ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη συστήματος υποστήριξης βάσεων δεδομένων πολλαπλών ενοίκων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΜΑΡΟΥΛΗ ΣΤΑΥΡΟΥ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2014

Η σελίδα αυτή είναι σκόπιμα λευκή.

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

Ανάπτυξη συστήματος υποστήριξης βάσεων δεδομένων πολλαπλών ενοίκων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΜΑΡΟΥΛΗ ΣΤΑΥΡΟΥ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Ιωάννης Βασιλείου	Ανδρέας Σταφυλοπάτης	Κώστας Κοντογιάννης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπλ. Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2014

(Υπογραφή)

.....

ΜΑΡΟΥΛΗΣ ΣΤΑΥΡΟΣ

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

Περίληψη

Οι αρχιτεκτονικές πολλαπλών ενοίκων χρησιμοποιούνται συχνά στις υπηρεσίες Software as a Service(SaaS), τόσο στο επίπεδο της εφαρμογής όσο και στις βάσεις δεδομένων που τις υποστηρίζουν. Στις αρχιτεκτονικές αυτές, πολλαπλοί πελάτες-ένοικοι μοιράζονται τους ίδιους υπολογιστικούς πόρους με αποτέλεσμα τη μείωση του κόστους των υπηρεσιών και την αυξημένη αξιοποίηση του διαθέσιμου υλικού(hardware). Παράλληλα, όμως, συνήθως αυξάνεται το αρχικό κόστος των εφαρμογών αυτών καθώς απαιτείται η ανάπτυξη πολύπλοκου κώδικα για την υποστήριξη πολλαπλών ενοίκων από το ίδιο στιγμιότυπο της εφαρμογής και από την ίδια βάση δεδομένων.

Στην παρούσα διπλωματική εργασία εστιάζουμε στις βάσεις δεδομένων πολλαπλών ενοίκων και αρχικά κάνουμε μια επισκόπηση διαφόρων τεχνικών που έχουν προταθεί για την υλοποίηση τους. Στη συνέχεια, συνδυάζοντας και τροποποιώντας κάποιες από τις τεχνικές αυτές, προτείνουμε μια τεχνική που προσπαθεί να αντιμετωπίσει το πρόβλημα της αντιστοίχισης των δεδομένων πολλαπλών ενοίκων σε κοινούς πίνακες μιας σχεσιακής βάσης δεδομένων. Ιδιαίτερη έμφαση δίνεται στην υποστήριξη της δυνατότητας για επεκτασιμότητα και προσαρμογή του σχήματος της βάσης στις ιδιαίτερες ανάγκες κάθε ενοίκου. Τέλος, χρησιμοποιώντας την τεχνική αυτή, αναπτύσσουμε ένα σύστημα που επιτρέπει την υποστήριξη βάσεων δεδομένων πολλαπλών ενοίκων από ένα σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων.

Λέξεις Κλειδιά: <<υπολογιστικό νέφος, multi-tenant βάσεις δεδομένων, multi-tenant εφαρμογές, Software as a Service>>

Η σελίδα αυτή είναι σκόπιμα λευκή.

Abstract

Multi-tenancy is often employed in Software as a Service applications, in both the application layer and the database layer. With multi-tenancy, multiple customers(tenants) share the same resources lowering operational costs and increasing the utilization of the available hardware. Nonetheless, the initial cost of deploying the application is often higher as complicated software structures need to be developed to support multiple tenants in the same application and database instance.

In this thesis, we focus on multi-tenant databases and start by examining several of the techniques used in implementing them. Then, combining and modifying some of those techniques, we present a schema-sharing technique that attempts to solve the problem of mapping the data of multiple tenants to the shared tables of a relational database. Particular emphasis is placed on the extensibility and customizability of the database schema to meet the demands of each tenant. Finally, using this technique, we develop a middleware layer that can be used to enable multi-tenancy in a standard relational database management system.

Keywords: << cloud computing, multi-tenant database, multi-tenant application, Software as a Service >>

Η σελίδα αυτή είναι σκόπιμα λευκή.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Ιωάννη Βασιλείου, για την υποστήριξη του και την ευκαιρία που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα. Επίσης, θα ήθελα να ευχαριστήσω θερμά τον κ. Γιώργο Παπαστεφανάτο για την καθοδήγηση και τις πολύτιμες υποδείξεις του στην εκπόνηση της παρούσας διπλωματικής εργασίας.
Τέλος, ευχαριστώ ιδιαίτερα την οικογένειά μου που με στήριξε σε όλη τη διάρκεια των σπουδών μου, καθώς και όλους όσους, με την ηθική και όχι μόνο υποστήριξη τους, συνέβαλαν στην ολοκλήρωση της παρούσας εργασίας.

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

1 Εισαγωγή 1

1.1 Υπολογιστικό Νέφος 1

1.2 Αρχιτεκτονικές πολλαπλών ενοίκων 2

1.2.1 Multi-tenant εφαρμογές 2

1.2.2 Multi-tenant βάσεις δεδομένων 3

1.3 Αντικείμενο διπλωματικής 5

1.4 Οργάνωση κειμένου 5

2 Σχετικές εργασίες 7

2.1 Προσεγγίσεις στη διαχείριση multi-tenant βάσεων δεδομένων 7

2.2 Τεχνικές αντιστοίχισης σχήματος στην προσέγγιση διαμοιραζόμενου σχήματος 10

2.3 Η περίπτωση της Salesforce.com 21

3 Η multi-tenant προσέγγιση του συστήματος 23

3.1 Τεχνική αντιστοίχισης στο φυσικό σχήμα 24

3.2 Πρωτεύοντα Κλειδιά 27

3.3 Επανεγγραφή ερωτημάτων 30

3.3.1 Ερωτήματα ανάκτησης δεδομένων 30

3.3.2 Ερωτήματα τροποποίησης δεδομένων 33

3.4 Έλεγχος Περιορισμών 37

3.4.1 Έλεγχος τύπων ιδιωτικών πεδίων 37

3.4.2 Περιορισμοί Αναφορικής Ακεραιότητας 38

3.4.3 Περιορισμοί Μοναδικότητας 38

4 Περιγραφή Συστήματος 41

4.1 Αρχιτεκτονική 41

4.2 Περιγραφή Δομικών Μονάδων 43

4.2.1 Metadata Manager *43*

4.2.2 Query Manager *45*

5 Σχεδίαση Συστήματος **47**

5.1 Αρχιτεκτονική 47

5.2 Περιγραφή Κλάσεων 49

5.2.1 multitenant.metadata.model *49*

5.2.2 multitenant.metadata.dal *52*

5.2.3 multitenant.metadata *54*

5.2.4 multitenant.query *58*

5.2.5 multitenant.exceptions *59*

5.3 Βάση Δεδομένων 60

5.3.1 Σχήμα μεταδεδομένων *61*

5.3.2 Φυσικό σχήμα διαμοιραζόμενων πινάκων *63*

6 Υλοποίηση **67**

6.1 Πλατφόρμες και προγραμματιστικά εργαλεία 67

6.1.1 Εργαλεία *67*

6.2 Λεπτομέρειες υλοποίησης 69

6.2.1 Λειτουργίες επανεγγραφής ερωτημάτων *69*

6.2.2 Λειτουργίες εξατομίκευσης σχήματος *80*

7 Έλεγχος **85**

7.1 Μεθοδολογία ελέγχου 85

7.2 Αναλυτική παρουσίαση ελέγχου 86

8 Επίλογος **103**

8.1 Σύνοψη και συμπεράσματα 103

8.2 Μελλοντικές επεκτάσεις 104

8.2.1 Υποστήριξη ευρετηρίων από ιδιωτικά πεδία *104*

8.2.2 Αλλαγή παραμέτρων συστήματος και αναδιάταξη δεδομένων *105*

9 Βιβλιογραφία 107

1

Εισαγωγή

1.1 Υπολογιστικό Νέφος

Μια από τις πιο δημοφιλείς τεχνολογίες τα τελευταία χρόνια είναι το Υπολογιστικό Νέφος(Cloud Computing). Το Υπολογιστικό Νέφος αφορά στη διάθεση υπολογιστικών πόρων και εφαρμογών ως υπηρεσίες μέσω του Διαδικτύου, η οποία έγινε δυνατή λόγω της μείωσης του κόστους αλλά και της αύξησης της ταχύτητας και της αξιοπιστίας των ευρυζωνικών συνδέσεων προς το Διαδίκτυο. Ένας πιο επίσημος ορισμός του Cloud Computing, από το Εθνικό Ινστιτούτο Τυποποιήσεων και Τεχνολογίας, είναι ο ακόλουθος:

> *«Το υπολογιστικό νέφος είναι η κατ' αίτηση διαδικτυακή κεντρική διάθεση υπολογιστικών πόρων (όπως δίκτυο, εξυπηρετητές, εφαρμογές και υπηρεσίες) με υψηλή ευελιξία, ελάχιστη προσπάθεια από τον χρήστη και υψηλή αυτοματοποίηση.»* [1]

Το Cloud Computing απαντάται σε τρεις κύριες μορφές:

- **Infrastructure as a Service(IaaS):** Ο πάροχος αυτής της μορφής του Cloud Computing, προσφέρει στους πελάτες του υπολογιστικούς πόρους όπως υπολογιστές(συνήθως εικονικά μηχανήματα) και χώρο αποθήκευσης δεδομένων.
- **Platform as a Service(PaaS)**: Εδώ παρέχεται στο χρήστη μια υπολογιστική πλατφόρμα που περιλαμβάνει συνήθως λειτουργικό σύστημα, περιβάλλον εκτέλεσης μιας ή περισσότερων γλωσσών προγραμματισμού, βάση δεδομένων και κάποιον

εξυπηρετητή ιστού(Web Server). Οι πλατφόρμες αυτές διευκολύνουν πολύ την ανάπτυξη και την διανομή προϊόντων λογισμικού, αφού οι προγραμματιστές δε θα ασχοληθούν καθόλου με την αγορά και τη διαχείριση του υλικού(hardware) καθώς και του λογισμικού που απαιτείται για την εκτέλεση των εφαρμογών τους.

- **Software as a Service (Saas):** Η τρίτη από τις εκδοχές του Υπολογιστικού Νέφους αναφέρεται σε Λογισμικό που προσφέρεται διαδικτυακά ως Υπηρεσία στο Νέφος. Οι χρήστες έχουν πρόσβαση μόνο στην εφαρμογή(και όχι στο υλικό και το λογισμικό που την υποστηρίζουν) και εξοικονομούν πόρους από την αγορά και συντήρηση λογισμικού, τη συντήρηση ακριβών εξυπηρετητών και εγκαταστάσεων αποθήκευσης δεδομένων.

1.2 Αρχιτεκτονικές πολλαπλών ενοίκων

Τη δεκαετία του 1990, η εξάπλωση του Διαδικτύου οδήγησε στην προσφορά υπηρεσιών φιλοξενίας και διαχείρισης εξειδικευμένου επιχειρησιακού λογισμικού κεντρικά από εταιρίες που ονομάζονταν Πάροχοι Υπηρεσιών Εφαρμογών(Application Service Providers). Οι Πάροχοι Υπηρεσιών Εφαρμογών διατηρούσαν τις απαραίτητες υποδομές(data centers) όπου φιλοξενούσαν λογισμικό όπως συστήματα ενδοεπιχειρησιακού σχεδιασμού των εταιριών-πελατών τους. Το μοντέλο του Software as a Service αποτελεί συνέχεια των παραπάνω υπηρεσιών, με τη διαφορά ότι οι πάροχοι του SaaS αναπτύσσουν τις δικές του εφαρμογές οι οποίες, ως επί το πλείστον, προσφέρονται μέσω του Διαδικτύου με χρήση ενός απλού πλοηγητή Ιστού[2]. Χαρακτηριστικό παράδειγμα είναι οι εφαρμογές πελατολογίου – CRM – που λειτουργούν στο cloud(π.χ. Salesforce.com). Μια εταιρία που δεν επιθυμεί να αγοράσει και να εγκαταστήσει σε δικά της υπολογιστικά συστήματα μια τέτοια εφαρμογή, μπορεί να ‘ενοικιάζει’ την εφαρμογή στο cloud.

1.2.1 Multi-tenant εφαρμογές

Στις εφαρμογές SaaS συνήθως χρησιμοποιείται ένα μοντέλο αρχιτεκτονικής όπου ονομάζεται ‘multi-tenancy’(μοντέλο πολλαπλών ενοικιαστών). Στο μοντέλο αυτό πολλοί ενοικιαστές(tenants) μοιράζονται το ίδιο στιγμιότυπο της εφαρμογής. Ως ενοικιαστή θεωρούμε την οντότητα που μισθώνει μια SaaS εφαρμογή. Ένας tenant μπορεί να είναι ένας ιδιώτης ή, συνήθως, ένας οργανισμός ή μια εταιρία. Οι multi-tenant αρχιτεκτονικές έρχονται σε αντιδιαστολή με τις παραδοσιακές αρχιτεκτονικές όπου σε κάθε ενοικιαστή αντιστοιχεί ένα στιγμιότυπο της εφαρμογής ή ακόμη και ξεχωριστό υπολογιστικό σύστημα[3].

Βασικό κίνητρο πίσω από τη χρήση των multi-tenant αρχιτεκτονικών είναι η δυνατότητα μείωσης του κόστους για κάθε ενοικιαστή και η ανάπτυξη οικονομιών κλίμακας. Πράγματι, η δημιουργία ενός στιγμιοτύπου μιας εφαρμογής έχει ένα ορισμένο αποτύπωμα στη μνήμη και στους επεξεργαστικούς πόρους ενός εξυπηρετητή, το οποίο όταν πολλαπλασιάζεται επί ένα μεγάλο αριθμό ενοικιαστών γίνεται αρκετά σημαντικό. Επιπλέον, πέραν του μειωμένου κόστους λειτουργίας, μια multi-tenant αρχιτεκτονική μπορεί να απλοποιήσει σημαντικά τον κύκλο ανάπτυξης του λογισμικού, αφού νέες εκδόσεις αρκεί να εγκατασταθούν σε ένα κοινό σύστημα και όχι για κάθε ενοικιαστή ξεχωριστά.

Παράλληλα, όμως, η χρήση μιας multi-tenant αρχιτεκτονικής προϋποθέτει συνήθως μεγαλύτερη προσπάθεια κατά το στάδιο ανάπτυξης της. Η έννοια του tenant έρχεται σε αντιδιαστολή με την έννοια του χρήστη στις παραδοσιακές εφαρμογές πολλαπλών χρηστών. Σε μια εφαρμογή πολλαπλών χρηστών κάθε χρήστης θεωρούμε ότι χρησιμοποιεί την ίδια εφαρμογή με περιορισμένες δυνατότητες για ρύθμιση της εφαρμογής στα μέτρα του. Αντίθετα, μια εφαρμογή πολλαπλών ενοικιαστών προϋποθέτει συνήθως μεγάλο περιθώριο προσαρμογής και ίσως και επέκτασης της, ώστε να ικανοποιεί τις ιδιαίτερες ανάγκες κάθε ενοικιαστή της[4]. Αυτή η απαίτηση επιβαρύνει το έργο των προγραμματιστών, οι οποίοι πρέπει να αναπτύξουν επιπλέον δομές στην εφαρμογή για την υποστήριξη της επεκτασιμότητας της, καθώς και για την αποθηκεύση των μετα-δεδομένων για κάθε ενοικιαστή(tenant). Επιπροσθέτως, λόγω της διαμοιραζόμενης πρόσβασης στους κοινούς πόρους της εφαρμογής, επιβάλλεται να ληφθούν μέτρα έτσι ώστε η χρήση της εφαρμογής από έναν tenant να μην επηρεάζει αρνητικά την απόδοση της εφαρμογής για κάποιον άλλον.

1.2.2 Multi-tenant βάσεις δεδομένων

Η multi-tenant προσέγγιση συνήθως επεκτείνεται και χρησιμοποιείται και στις βάσεις δεδομένων των multi-tenant εφαρμογών. Οι βάσεις δεδομένων πολλαπλών ενοικιαστών(multi-tenant databases) είναι βάσεις δεδομένων που εξυπηρετούν πολλούς ενοικιαστές ταυτόχρονα, μειώνοντας το κόστος αποθήκευσης και πρόσβασης στα δεδομένα και αξιοποιώντας αποδοτικότερα το διαθέσιμο υλικό και λογισμικό. Επίσης, διευκολύνουν σημαντικά τις εργασίες συντήρησης και αναβάθμισης στη βάση δεδομένων, εργασίες που παραδοσιακά θα έπρεπε να γίνουν για κάθε tenant ξεχωριστά. Σημαντικό κίνητρο για την επιλογή μιας multi-tenant προσέγγισης στις βάσεις δεδομένων μπορεί να αποτελέσει και η ευκολία με την οποία μπορούν να εφαρμοστούν τεχνικές συνάθροισης και ανάλυσης δεδομένων(data aggregation and analysis) σε αυτές[7]. Παρ’ότι οι τεχνικές αυτές μπορούν να εφαρμοστούν και σε δεδομένα από διαφορετικές βάσεις δεδομένων, η ανάλυση δεδομένων που μοιράζονται την ίδια βάση είναι ευκολότερη και αποδοτικότερη.

Όπως και στην περίπτωση της ίδιας της εφαρμογής, ο κάθε ενοικιαστής μπορεί να έχει διαφορετικές απαιτήσεις για το σχήμα της βάσης ή για την απόδοση του συστήματος που μισθώνει. Συνεπώς, και οι multi-tenant βάσεις δεδομένων πρέπει να προσφέρουν ένα βαθμό ευελιξίας ως προς τα δεδομένα που αποθηκεύει ο κάθε tenant σε αυτές. Εκτός αυτού, εγείρονται και ζητήματα ασφάλειας καθώς τα δεδομένα πολλών tenants αποθηκεύονται στην ίδια βάση. Αυτά τα ζητήματα πρέπει να αντιμετωπιστούν αποτελεσματικά έτσι ώστε να αποτρέπεται η πρόσβαση ενός tenant στα δεδομένα ενός άλλου. Ειδικότερα σε εφαρμογές που υποστηρίζουν σημαντικές επιχειρησιακές λειτουργίες, όπως τα Συστήματα Διαχείρισης Πελατειακών Σχέσεων(CRM), η ανεπαρκής εξασφάλιση της απομόνωσης(λογικής ή φυσικής) των δεδομένων μιας εταιρίας από κάποιας άλλης(πιθανώς ανταγωνιστικής), αποτελεί σημαντικό αποτρεπτικό παράγοντα για την επιλογή της συγκεκριμένης εφαρμογής.

Έχουν αναπτυχθεί αρκετές προσεγγίσεις και τεχνικές για την ανάπτυξη multi-tenant βάσεων δεδομένων. Ως επί το πλείστον, μιας και τα παραδοσιακά σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων(ΣΔΒΔ) δεν υποστηρίζουν τη multi-tenant σημασιολογία[5], οι μηχανικοί που αναπτύσσουν multi-tenant εφαρμοργές στο Υπολογιστικό Νέφος συνήθως ενσωματώνουν στην εφαρμογή τη λογική για την υποστήριξη πολλαπλών tenants.

Η προσέγγιση αυτή μπορεί να παρομοιαστεί με αντίστοιχες προσεγγίσεις που ακολουθούσαν οι προγραμματιστές για την ανάπτυξη εφαρμογών πολλαπλών χρηστών τις πρώτες μέρες των λειτουργικών συστημάτων, όταν αυτά δεν προσέφεραν εγγενή υποστήριξη τέτοιων εφαρμογών[8]. Οι προγραμματιστές αυτοί έπρεπε να συμπεριλάβουν στις εφαρμογές τους πολύπλοκο κώδικα για να υποστηρίξουν πολλαπλούς χρήστες, πράγμα που έκανε το έργο τους δυσκολότερο. Αυτό άλλαξε άρδην όταν τα λειτουργικά συστήματα ενσωμάτωσαν τη λογική υποστήριξης πολλαπλών χρηστών. Αντίστοιχα βήματα έχουν αρχίσει να γίνονται και για την υποστήριξη αρχιτεκτονικών πολλαπλών ενοικιαστών από τις βάσεις δεδομένων, όμως, προς το παρόν, τα περισσότερα ΣΔΒΔ δεν προσφέρουν εγγενή υποστήριξη multi-tenant βάσεων δεδομένων.

Αν και τα τελευταία χρόνια έχουν αρχίσει να χρησιμοποιούνται ευρέως συστήματα που δεν ακολουθούν το σχεσιακό μοντέλο(δηλαδή πινακοειδής αναπαράσταση των δεδομένων) και χαρακτηρίζονται γενικά ως NoSQL βάσεις δεδομένων, στα πλαίσια της παρούσας διπλωματικής εργασίας θα εστιάσουμε την προσοχή μας στην υλοποίηση σχεσιακών multi-tenant βάσεων δεδομένων. Αρχικά, το σχεσιακό μοντέλο εξακολουθεί να αποτελεί την καλύτερη επιλογή για κάποια είδη εφαρμογών(π.χ. συστήματα υποστήριξης επιχειρήσεων, CRM κ.λ.π.). Για παράδειγμα, οι περισσότερες NoSQL βάσεις δεδομένων δεν υποστηρίζουν πλήρεις ACID δοσοληψίες, κάτι που αποτελεί σημαντικό μειονέκτημα για εφαρμογές που απαιτούν τα δεδομένα τους να είναι συνεχώς σε συνεπή μορφή. Επιπλέον, τα σχεσιακά συστήματα, σε αντίθεση με τα περισσότερα είδη NoSQL συστημάτων(π.χ. key-value stores),

απαιτούν αυστηρό ορισμό και τήρηση ενός σχήματος για την αποθήκευση των δεδομένων. Η απαίτηση αυτή καθιστά σημαντικά δυσκολότερη την ανάπτυξη multi-tenant βάσεων δεδομένων σε ένα σχεσιακό σύστημα σε σχέση με ένα NoSQL σύστημα. Αυτό οφείλεται στο ότι στις multi-tenant βάσεις επιθυμούμε μεγαλύτερη ευελιξία στο σχήμα δεδομένων, το οποίο θα πρέπει να μπορεί να προσαρμόζεται στις ιδιαίτερες ανάγκες κάθε tenant. Έτσι, στην παρούσα διπλωματική εργασία θα εξετάσουμε και θα επιχειρήσουμε να αντιμετωπίσουμε το πρόβλημα αυτό, δηλαδή το πως μπορούμε, χρησιμοποιώντας ένα παραδοσιακό σχεσιακό ΣΔΒΔ, να υλοποιήσουμε multi-tenant βάσεις δεδομένων με δυνατότητα προσαρμογής του σχήματος από κάθε tenant.

1.3 Αντικείμενο διπλωματικής

Στην παρούσα διπλωματική εργασία, αφού κάνουμε μια επισκόπηση των μεθόδων που έχουν διατυπωθεί και εφαρμοστεί για την ανάπτυξη και υποστήριξη multi-tenant βάσεων δεδομένων, διατυπώνουμε ένα συνδυασμό και παραλλαγή κάποιων από τις μεθόδους αυτές και αναπτύσσουμε ένα σύστημα το οποίο, όταν λειτουργεί πάνω από ένα παραδοσιακό σχεσιακό ΣΔΒΔ(το οποίο δεν υποστηρίζει το multi-tenant μοντέλο), προσθέτει multi-tenant δυνατότητες σε αυτό. Το σύστημα αυτό αναπτύχθηκε με έμφαση στην ικανοποίηση των παρακάτω στόχων:

1. Υποστήριξη του multi-tenant μοντέλου με έμφαση στη μεγιστοποίηση του βαθμού αξιοποίησης των υπολογιστικών πόρων(αύξηση του αριθμού των tenants που υποστηρίζονται από δεδομένους υπολογιστικούς πόρους).
2. Δυνατότητα επέκτασης και προσαρμογής του σχήματος δεδομένων για κάθε tenant
3. Διαφάνεια του συτήματος. Το σύστημα θα πρέπει να προσφέρει μια διεπαφή που θα αποκρύπτει την υποκείμενη multi-tenant λογική του.
4. Διασφάλιση και διαχωρισμό των δεδομένων που ανήκουν σε κάθε tenant

1.4 Οργάνωση κειμένου

Παρακάτω περιγράφονται συνοπτικά τα κεφάλαια της παρούσας διπλωματικής εργασίας. Στο $2^{ο}$ κεφάλαιο πραγματοποιείται μια ανασκόπηση των διαφόρων προσεγγίσεων και τεχνικών που χρησιμοποιούνται στις multi-tenant βάσεις δεδομένων. Στο $3^{ο}$ κεφάλαιο διατυπώνουμε μια παραλλαγή κάποιων από τις υπάρχουσες τεχνικές, η οποία και θα χρησιμοποιείται από το

σύστημα που θα αναπτύξουμε και στο 4ο κεφάλαιο προχωράμε στην περιγραφή της γενικής αρχιτεκτονικής του συστήματος. Ακολουθεί η ανάλυση της σχεδίασης του, που πραγματοποιείται στο 5ο κεφάλαιο. Στο 6ο κεφάλαιο παρουσιάζουμε κάποια σημεία σχετικά με την υλοποίηση του συστήματος και περιγράφουμε τις πλατφόρμες και τα εργαλεία που χρησιμοποιήσαμε. Στη συνέχεια, στο 7ο κεφάλαιο, ακολουθεί η περιγραφή της διαδικασίας ελέγχου του συστήματος. Ο έλεγχος πραγματοποιήθηκε με την εκτέλεση κατάλληλων σεναρίων λειτουργίας σε μια διεπαφή ιστού που αναπτύξαμε ειδικά για τη διαδικασία ελέγχου. Τέλος, στο 8ο κεφάλαιο ολοκληρώνουμε τη διπλωματική αυτή εργασία συνοψίζοντας την και παραθέτοντας προτάσεις για μελλοντική επέκταση της.

.

2

Σχετικές εργασίες

Στο κεφάλαιο αυτό παρουσιάζονται οι τέσσερις κύριες προσεγγίσεις στην ανάπτυξη multi-tenant βάσεων δεδομένων και εξετάζονται τα πλεονεκτήματα και τα μειονεκτήματα κάθε προσέγγισης. Στη συνέχεια, δίνεται έμφαση στην τέταρτη προσέγγιση, όπου αναλύονται κάποιες τεχνικές που έχουν προταθεί για την επίτευξη της προσαρμοστικότητας του σχήματος δεδομένων στις ανάγκες κάθε tenant.

2.1 Προσεγγίσεις στη διαχείριση multi-tenant βάσεων δεδομένων

Κατά τη σχεδίαση μιας multi-tenant εφαρμογής, οι σχεδιαστές καλούνται να αποφασίσουν το βαθμό διαμοιρασμού των διαφόρων πόρων που απαιτεί η εφαρμογή. Συγκεκριμένα στο κομμάτι της αποθήκευσης των δεδομένων της εφαρμογής, οι τέσσερις κύριες προσεγγίσεις είναι οι παρακάτω[6],[10]:

1. **Διαφορετικό στιγμιότυπο του ΣΔΒΔ για κάθε tenant**

   Στην προσέγγιση αυτή κάθε tenant αντιστοιχίζεται σε ένα ξεχωριστό στιγμιότυπο του ΣΔΒΔ και πολλοί tenants μοιράζονται τον ίδιο φυσικό εξυπηρετητή, δηλαδή το ίδιο υπολογιστικό μηχάνημα. Η κοινή χρήση του ίδιου μηχανήματος γίνεται συνήθως

με χρήση τεχνικών εικονικοποίησης(virtualization), αν και μπορεί να γίνει και με τη χρήση κοινού λειτουργικού συστήματος στο οποίο τρέχουν πολλαπλά στιγμιότυπα του ΣΔΒΔ, καθένα από το οποίο αντιστοιχίζεται σε διαφορετικό χρήστη του λειτουργικού συτήματος. Εδώ, τα δεδομένα κάθε tenant είναι επαρκώς απομονωμένα από των υπολοίπων(ειδικότερα με τη χρήση τεχνικών virtualization) και κάθε tenant μπορεί εύκολα να τροποποιήσει το σχήμα του καθώς επιθυμεί. Όμως, η προσέγγιση αυτή είναι η ακριβότερη από τις τέσσερις, καθώς κάθε διαφορετικό στιγμιότυπο του ΣΔΒΔ έχει μεγάλο αποτύπωμα στη μνήμη.

2. **Διαφορετική βάση δεδομένων για κάθε tenant**

Στην προσέγγιση αυτή κάθε tenant αποκτά μια δική του βάση δεδομένων, στο ίδιο στιγμιότυπο του ΣΔΒΔ. Τα δεδομένα κάθε tenant είναι και εδώ απομονωμένα σε φυσικό επίπεδο από των υπολοίπων και κάθε tenant έχει πρόσβαση μόνο στα δικά του δεδομένα, αφού μπορεί να χρησιμοποιήσουμε τους μηχανισμούς ασφαλείας του ΣΔΒΔ για να αντιστοιχήσουμε ένα tenant με τη βάση του. Επιπλέον, η εξατομίκευση του σχήματος δεδομένων ενός tenant μπορεί να γίνει και εδώ πολύ εύκολα. Εντούτοις, η προσέγγιση αυτή, αν και φθηνότερη από την πρώτη, παραμένει ακριβή, και όσον αφορά στο κόστος συντήρησης αλλά και στο κόστος υλικού, αφού ο αριθμός των tenants που μπορούν να φιλοξενηθούν σε ένα εξυπηρετητή βάσεων δεδομένων περιορίζεται από τον αριθμό των βάσεων δεδομένων που αυτός μπορεί να υποστηρίξει. Παρά το αυξημένο κόστος, η προσέγγιση αυτή μπορεί να ενδείκνυται για εφαρμογές όπου η ασφάλεια των δεδομένων αλλά και η εύκολη επεκτασιμότητα του σχήματος είναι υψίστης σημασίας(π.χ. εφαρμογές του τραπεζικού τομέα).

3. **Διαφορετικό σχήμα για κάθε tenant**

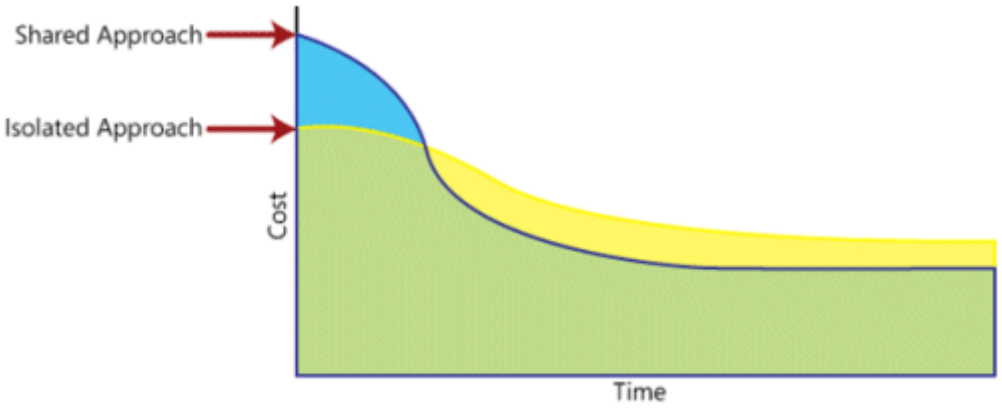
Στην τρίτη προσέγγιση, μια βάση δεδομένων μπορεί να περιέχει δεδομένα πολλαπλών ενοικιαστών, τα οποία όμως αποθηκεύονται σε ξεχωριστό σύνολο από πίνακες και κάθε σύνολο αντιστοιχίζεται σε ένα σχήμα της βάσης δεδομένων. Παρότι η δομή του σχήματος δεν χρησιμοποιείται από όλα τα Συστήματα Διαχείρισης Βάσεων Δεδομένων, είναι πολλά εκείνα που την υποστηρίζουν(π.χ. Postgresql, Microsoft SQL Server). Μια βάση δεδομένων μπορεί να περιέχει πολλά σχήματα, καθένα από τα οποία έχει ένα ξεχωριστό όνομα. Όπως και η πρώτη προσέγγιση, η προσέγγιση του διαφορετικού σχήματος είναι απλή στην υλοποίηση και μπορεί εύκολα να υποστηρίξει διαφορετικής μορφής δεδομένα για κάθε tenant. Η προσέγγιση αυτή παρέχει ένα μέτριο βαθμό απομόνωσης των δεδομένων, καθώς σε αντίθεση με τις βάσεις δεδομένων, τα σχήματα δε διαχωρίζονται αυστηρά. Συγκεκριμένα, το ΣΔΒΔ συνήθως επιτρέπει σε ένα χρήστη να έχει πρόσβαση σε

δεδομένα όλων των σχημάτων της βάσης δεδομένων με την οποία έχει συνδεθεί. Οπότε, πρέπει να εξασφαλιστεί με κάποιον άλλο τρόπο η απαγόρευση της πρόσβασης σε δεδομένα άλλων tenants. Η προσέγγιση αυτή μπορεί να υποστηρίξει μεγαλύτερο αριθμό ενοκιαστών σε κάθε εξυπηρετητή σε σχέση με τις πρώτες δύο προσεγγίσεις με συνέπεια την μείωση του κόστους της εφαρμογής.

4. **Διαμοιραζόμενο σχήμα**

   Στην τέταρτη προσέγγιση χρησιμοποιούμε την ίδια βάση δεδομένων και το ίδιο σχήμα σε αυτή για να αποθηκεύσουμε δεδομένα πολλαπλών ενοίκων. Με άλλα λόγια, ένας φυσικός πίνακας της βάσης δεδομένων μπορεί να περιέχει εγγραφές διαφορετικών ενοκιαστών, οι οποίες διαχωρίζονται από ένα πεδίο το οποίο περιέχει το αναγνωριστικό του αντίστοιχου tenant. Από τις τέσσερις προσεγγίσεις, η προσέγγιση αυτή μπορεί να υποστηρίξει το μεγαλύτερο αριθμό από tenants σε κάθε εξυπηρετητή. Παράλληλα, όμως, είναι και η προσέγγιση με τα μεγαλύτερα ρίσκα όσον αφορά στην ασφάλεια των δεδομένων κάθε tenant, καθώς και η προσέγγιση με τη μεγαλύτερη δυσκολία στην προσαρμογή του σχήματος δεδομένων κάθε tenant στις ιδιαίτερες ανάγκες του. Τα παραπάνω αντιμετωπίζονται συνήθως μέσα στη λογική της εφαμοργής με μεθόδους που θα παρουσιαστούν στην επόμενη παράγραφο. Άλλο πρόβλημα της συγκεκριμένης προσέγγισης είναι ότι η μεταφορά των δεδομένων ενός tenant σε κάποιον άλλο εξυπηρετητή προϋποθέτει την εκτέλεση ερωτημάτων στους κοινούς πίνακες αντί της απλής μεταφοράς αρχείων του λειτουργικού συστήματος, όπως στις τρεις πρώτες προσεγγίσεις. Επιπλέον, το ότι τα ερωτήματα ενός tenant στη βάση πρέπει να προσπελάσουν και τα δεδομένα όλων των υπολοίπων δυσχεραίνει το έργο του βελτιστοποιητή ερωτημάτων(query optimizer). Συγκεκριμένα, τα στατιστικά στοιχεία που διατηρεί ο βελτιστοποιητής αφορούν δεδομένα όλων των tenants. Για το λόγο αυτό, παρά το χαμηλότερο μετέπειτα κόστος της προσέγγισης διαμοιραζόμενου σχήματος, η αρχική ανάπτυξη της εφαρμογής είναι συνήθως ακριβότερη.

Στην Εικόνα 2-1([6]) βλέπουμε το κόστος ως συνάρτηση του χρόνου για την προσέγγιση διαμοιραζόμενου σχήματος(μπλε γραμμή) σε σχέση με το κόστος των υπόλοιπων προσεγγίσεων(κίτρινη γραμμή). Παρατηρούμε, όπως αναφέραμε και πριν, ότι παρά το υψηλότερο αρχικό κόστος της προσέγγισης διαμοιραζόμενου σχήματος, το μετέπειτα κόστος λειτουργίας της εφαρμογής είναι εμφανώς χαμηλότερο, γεγονός που κάνει την προσέγγιση αυτή πολύ ελκυστική. Έτσι, παρά τις παραπάνω προκλήσεις που εισάγει η χρήση κοινού σχήματος, πολλές εφαρμογές στο Υπολογιστικό Νέφος επιλέγουν αυτή την προσέγγιση.



Εικόνα 2-1

2.2 Τεχνικές αντιστοίχισης σχήματος στην προσέγγιση διαμοιραζόμενου σχήματος

Όπως είδαμε παραπάνω, η προσέγγιση διαμοιραζόμενου σχήματος, παρά τις τεχνικές δυσκολίες της, είναι πολύ ενδιαφέρουσα από οικονομικής απόψεως. Το γεγονός, όμως, ότι τα δεδομένα πολλαπλών ενοικιαστών αποθηκεύονται σε κοινούς πίνακες, μας αναγκάζει να κάνουμε διάκριση ανάμεσα στο λογικό σχήμα των δεδομένων ενός tenant και στο φυσικό σχήμα της βάσης δεδομένων που τα περιέχει. Η διάκριση αυτή γίνεται ακόμα πιο αναγκαία στην περίπτωση που το λογικό σχήμα κάθε tenant μπορεί να διαφοροποιείται από των υπολοίπων και να περιλαμβάνει πεδία και πίνακες που αφορούν μόνον εκείνον.

Στην απλούστερη προσέγγιση, όλοι οι tenants μοιράζονται τους ίδιους πίνακες και τα ίδια ακριβώς πεδία. Δηλαδή, δεν υπάρχει διαφοροποίηση στο λογικό σχήμα δεδομένων κάθε tenant. Εδώ, η μόνη απαίτηση είναι η δημιουργία σε κάθε πίνακα ενός πεδίου το οποίο περιέχει το αναγνωριστικό του συγκεκριμένου tenant. Το συγκεκριμένο πεδίο, που στα πλαίσια της εργασίας αυτής θα ονομάζουμε “tenant_id”, δεν είναι ορατό από τους tenants. Πράγματι, κάθε tenant δεν πρέπει, κατά την αλληλεπίδραση του με τη βάση δεδομένων, να τον απασχολεί το ότι τη μοιράζεται με άλλους tenants. Θεωρεί, δηλαδή, ότι είναι ο μοναδικός ιδιοκτήτης της βάσης δεδομένων. Έτσι, όλα τα ερωτήματα που εκτελεί σε αυτή θα πρέπει να μετασχηματίζονται κατάλληλα έτσι ώστε να επιστρέφουν ή να τροποποιούν εγγραφές που αφορούν μόνον εκείνον. Στον Πίνακα 2-1 βλέπουμε έναν απλό πίνακα ο οποίος περιέχει πληροφορίες σχετικά με έρευνες/ερωτηματολόγια. Ο πίνακας περιέχει εγγραφές πολλαπλών ενοικιαστών οι οποίες αντιστοιχίζονται στον κατάλληλο ενοικιαστή μέσω του πεδίου

“tenant_id”. Παρατηρούμε, επίσης, πως το πεδίο “survey_id”, που σε περίπτωση που ο πίνακας περιείχε δεδομένα ενός μόνο tenant θα αποτελούσε πρωτεύον κλειδί, εδώ περιέχει την ίδια τιμή παραπάνω από μια φορές. Με άλλα λόγια, το πρωτεύον κλειδί του πίνακα πρέπει να συμπεριλάβει και το πεδίο “tenant_id”, εφόσον αυτός αφορά πολλαπλούς ενοικιαστές. Στον Πίνακα 2-2 βλέπουμε το πως αντιλαμβάνεται ο tenant με αναγνωριστικό 1 τον ίδιο πίνακα.

tenant_id	survey_id	survey_title	description	end_date
1	1	Product #432 Launch	market research for new product	19/11/2013
2	1	Laptop vs tablet	null	5/12/2013
1	2	New-born Lion Name	Give a name to our new lion cub	10/1/2014
3	1	Customer Satisfaction	Yearly customer satisfaction survey	7/11/2013
2	2	Best Radio 2012	Radio station awards	20/1/2014

Πίνακας 2-1

survey_id	survey_title	description	end_date
1	Product #432 Launch	market research for new product	19/11/2013
2	New-born Lion Name	Give a name to our new lion cub	10/1/2014

Πίνακας 2-2

Η παραπάνω προσέγγιση, αν και απλή, προϋποθέτει ότι όλοι οι tenants “βλέπουν” το ίδιο ακριβώς “λογικό” σχήμα. Παρότι σε παραδοσιακές εφαρμογές του ιστού(π.χ. web mail) η προσέγγιση αυτή είναι επαρκής και χρησιμοποιείται πολύ συχνά, οι σύγχρονες εφαρμογές που προσφέρονται στο Νέφος απαιτούν, όπως προαναφέραμε, τη δυνατότητα τροποποίησης του σχήματος δεδομένων κάθε ενοικιαστή τους.

Για παράδειγμα, ας υποθέσουμε ότι στο βασικό και κοινό πίνακα ερευνών που είδαμε παραπάνω ο tenant με αναγνωριστικό "1" θέλει να προσθέσει δύο επιπλέον πεδία: ένα πεδίο με όνομα "is_open" που δείχνει αν μια έρευνα είναι ακόμα ενεργή, και ένα πεδίο με όνομα "version" στο οποίο αποθηκεύει την τρέχουσα έκδοση μιας έρευνας, σε περίπτωση που αυτή υπόκειται σε τροποποιήσεις. Αντίστοιχα, ο tenant με αναγνωριστικό "2" επιθυμεί να προσθέσει ένα πεδίο με όνομα "min_responses" στο οποίο θα αποθηκεύει τον ελάχιστο απαιτούμενο αριθμό από συμμετοχές σε μια έρευνα. Στους Πίνακες 2-3 και 2-4 βλέπουμε τους "λογικούς" πίνακες ερευνών, για τους tenants 1 και 2 αντίστοιχα, συμπεριλαμβανομένων και των πεδίων-επεκτάσεων τους.

survey_id	survey_title	description	end_date	is_open	version
1	Product #432 Launch	market research for new product	19/11/2013	TRUE	0.6
2	New-born Lion Name	Give a name to our new lion cub	10/1/2014	FALSE	1

Πίνακας 2-3

(λογικός πίνακας surveys για τον tenant 1)

survey_id	survey_title	description	end_date	min_responses
1	Laptop vs tablet	null	5/12/2013	100
2	Best Radio 2012	Radio station awards	20/1/2014	150

Πίνακας 2-4

(λογικός πίνακας surveys για τον tenant 2)

Αν δε χρησιμοποιούσαμε την προσέγγιση διαμοιραζόμενου σχήματος, τότε κάθε tenant θα είχε από έναν ιδιωτικό πίνακα ερευνών που θα τροποποιούσε καθώς επιθυμούσε. Η χρήση, όμως, κοινών πινάκων μας αναγκάζει να αναζητήσουμε άλλους τρόπους για την εξατομίκευση του λογικού σχήματος ενός tenant. Στη συνέχεια της παραγράφου αυτής

παρουσιάζουμε κάποιες από τις τεχνικές που έχουν προταθεί και εφαρμοσθεί για την αντιμετώπιση του παραπάνω προβλήματος[6] [9] [12] [11]. Οι τεχνικές αυτές αφορούν κυρίως τα σχεσιακά ΣΔΒΔ.

Γενικά Πεδία(Preallocated Fields)

Με την τεχνική αυτή, σε κάθε έναν από τους κοινούς πίνακες μπορούμε να προσθέσουμε έναν αριθμό από γενικά πεδία(preallocated fields) στα οποία κάθε tenant θα αποθηκεύει τις τιμές για τα πεδία-επεκτάσεις που ορίζει. Για παράδειγμα, στον Πίνακα 2-5 βλέπουμε τον ίδιο πίνακα ερευνών που είδαμε και στον Πίνακα 2-1. Στον Πίνακα 2-5, όμως, πέραν από τα βασικά και κοινά πεδία, έχουμε και δυο επιπλέον γενικά πεδία(preallocated fields). Ο tenant 1 χρησιμοποιεί τα πεδία αυτά για να αποθηκεύσει τα πεδία-επεκτάσεις του "is_open" και "version", ενώ ο tenant 2 χρησιμοποιεί μόνο το πρώτο από αυτά για το πεδίο-επέκταση του με όνομα "min_responses". Με άλλα λόγια, τα γενικά αυτά πεδία περιέχουν δεδομένα διαφορετικής σημασιολογίας για κάθε tenant. Επίσης, παρ'ότι κάθε γενικό πεδίο μπορεί να οριστεί ώστε να δέχεται δεδομένα συγκεκριμένου τύπου, είναι σύνηθες τα πεδία αυτά να δέχονται αλφαριθμητικά(strings). Έτσι, ενώ ένας tenant μπορεί να χρησιμοποιεί ένα γενικό πεδίο για να αποθηκεύει τιμές boolean, ένας άλλος μπορεί να αποθηκεύει αριθμητικές τιμές, οι οποίες τελικά αποθηκεύονται στη βάση ως αλφαριθμητικά.

tenant_id	survey_id	survey_title	description	end_date	prealloc_field_1	prealloc_field_1
1	1	Product #432 Launch	market research for new product	19/11/2013	TRUE	0.6
2	1	Laptop vs tablet	null	5/12/2013	100	null
1	2	New-born Lion Name	Give a name to our new lion cub	10/1/2014	FALSE	1
3	1	Customer Satisfaction	Yearly customer satisfaction survey	7/11/2013	null	null
2	2	Best Radio 2012	Radio station awards	20/1/2014	150	null

Πίνακας 2-5

Όπως αναφέρθηκε και στην αρχή της παραγράφου, οι tenants αντιλαμβάνονται και αλληλεπιδρούν μόνο με το "λογικό" τους σχήμα και δεν πρέπει να τους απασχολεί το πώς αντιστοιχίζεται αυτό στο φυσικό σχήμα της βάσης. Συνεπώς, το σύστημα πρέπει να διατηρεί μεταδεδομένα και να κάνει εκείνο την κατάλληλη αντιστοίχιση. Π.χ όταν κάποιο ερώτημα ενός tenant αναφέρεται σε πεδίο-επέκταση του, το σύστημα πρέπει να αντικαθιστά την αναφορά αυτή με το αντίστοιχο γενικό πεδίο(preallocated field).

Στην τεχνική αυτή, ο αριθμός των πεδίων-επεκτάσεων ενός tenant περιορίζεται από τον αριθμό των γενικών πεδίων που έχουν ορισθεί στον αντίστοιχο πίνακα. Αν και θα μπορούσαμε να ορίσουμε πολύ μεγάλο αριθμό από γενικά πεδία σε ένα διαμοιραζόμενο πίνακα, αυτό θα είχε το μειονέκτημα της ύπαρξης πολλών τιμών null σε εγγραφές ενοίκων που δεν τα χρησιμοποιούν.

Πίνακες Επέκτασης

Μια διαφορετική τεχνική, που επιτρέπει θεωρητικά απεριόριστο αριθμό από πεδία-επεκτάσεις χωρίς το πρόβλημα των πολλών τιμών null της προηγούμενης τεχνικής, είναι οι Πίνακες Επέκτασης. Εδώ, όλοι οι tenants μοιράζονται ένα βασικό πίνακα και τα πεδία-επεκτάσεις αυτών μπαίνουν σε ξεχωριστούς πίνακες επέκτασης. Στο παράδειγμα μας οι πίνακες επέκτασης θα ήταν οι παρακάτω:

survey_id	is_open	version
1	TRUE	0.6
2	FALSE	1

Πίνακας 2-6

(πίνακας επέκτασης του tenant 1 για τον πίνακα surveys)

survey_id	min_responses
1	100
2	150

Πίνακας 2-7

(πίνακας επέκτασης του tenant 2 για τον πίνακα surveys)

Όπως βλέπουμε, για κάθε tenant που επεκτείνει ένα βασικό πίνακα δημιουργείται ένας πίνακας επέκτασης με τα επιπλέον πεδία του. Κάθε τέτοιος πίνακας αντιστοιχεί σε ένα μόνο βασικό πίνακα. Σε περίπτωση που παραπάνω από ένας tenant μοιράζονται τα ίδια πεδία επέκτασης σε έναν βασικό πίνακα, θα μπορούσε ένας πίνακας επέκτασης να χρησιμοποιούνταν από όλους τους αντίστοιχους tenants. Τότε, θα έπρεπε να συμπεριλάβουμε σε αυτόν και ένα πεδίο "tenant_id".

Για την ανακατασκευή ενός λογικού πίνακα ενός tenant, πρέπει να εκτελέσουμε μια λειτουργία συνδέσμου(join) ανάμεσα σε έναν βασικό πίνακα και στον αντίστοιχο πίνακα επέκτασης του. Η λειτουργία συνδέσμου αυτή είναι απαραίτητη σε ερωτήματα που αναφέρονται και σε βασικά αλλά και σε πεδία επεκτάσεις και επιβαρύνει αρνητικά την απόδοση τους σε σχέση με την περίπτωση που όλα τα πεδία αποθηκεύονται σε ένα φυσικό πίνακα. Ένα ακόμα μειονέκτημα της τεχνικής αυτής είναι ότι για μεγάλο αριθμό ενοικιαστών με επεκτάσεις ο αριθμός των πινάκων στη βάση μεγαλώνει ανάλογα. Έτσι, ο αριθμός των ενοικιαστών που μπορούν να μοιράζονται την ίδια βάση περιορίζεται από το μέγιστο αριθμό πινάκων που αυτή μπορεί να υποστηρίξει.

Χρήση XML για επεκτάσεις

Στην τεχνική αυτή, σε ένα διαμοιραζόμενο πίνακα προσθέτουμε ένα πεδίο το οποίο περιέχει ένα αρχείο xml με τα επιπρόσθετα και ιδιωτικά για κάθε tenant πεδία, όπως φαίνεται στον Πίνακα 2-8. Παρότι με κάθε ΣΔΒΔ μπορούμε να αποθηκεύσουμε xml αρχεία(ακόμη και αν αυτά αποθηκεύονται ως αλφαριθμητικά ή CLOBs), ορισμένα συστήματα υποστηρίζουν ανάλυση και αναζήτηση σε τέτοια αρχεία(π.χ. pureXML του συστήματος DB2, PostgreSQL κ.α.) έτσι ώστε να μη φορτώνονται ολόκληρα κατά την εκτέλεση ενός ερωτήματος, αλλά μόνο τα κομμάτια τους που χρειάζονται. Παραλλαγή της τεχνικής αυτής, θα ήταν να μην αποθηκεύουμε τα xml αρχεία στους κοινούς πίνακες αλλά σε πίνακες επέκτασης όπως προτείνεται στο [13].

tenant_id	survey_id	survey_title	description	end_date	extension_xml
1	1	Product #432 Launch	market research for new product	19/11/2013	<extension> <is_open>true</is_open> <version>0.6</version> </extension>
2	1	Laptop vs tablet	null	5/12/2013	<extension> <min_responses>100 </ min_responses > </extension>
1	2	New-born Lion Name	Give a name to our new lion cub	10/1/2014	<extension> <is_open>false</is_open> <version>1</version> </extension>
3	1	Customer Satisfaction	Yearly customer satisfaction survey	7/11/2013	null
2	2	Best Radio 2012	Radio station awards	20/1/2014	<extension> <min_responses>150 </ min_responses > </extension>

Πίνακας 2-8

Γενικό Πίνακας(Universal Table)

Στην τεχνική αυτή διατηρούμε ένα φυσικό πίνακα στη βάση δεδομένων στον οποίο αποθηκεύουμε τα δεδομένα όλων των λογικών πινάκων που χρησιμοποιούν οι tenants. Ο πίνακας αυτός περιέχει ένα πολύ μεγάλο αριθμό από γενικά πεδία αλφαριθμητικού τύπου(varchar), καθώς και πεδία που ορίζουν σε ποιο λογικό πίνακα και σε ποιον tenant ανήκει μια εγγραφή. Το πλήθος των γενικών πεδίων επιλέγεται μεγάλο, ώστε στον πίνακα να μπορούν να αποθηκευθούν και εγγραφές αρκετά ευρέων λογικών πινάκων. Στο πρώτο γενικό πεδίο του γενικού αυτού πίνακα, αντιστοιχίζεται η πρώτη στήλη κάθε λογικού πίνακα και ούτω καθεξής. Για εγγραφές λογικών πινάκων με πλήθος στηλών μικρότερο από το πλήθος των γενικών πεδίων, τα πεδία που δε χρησιμοποιούνται παίρνουν τιμή NULL. Κάθε tenant μπορεί να επεκτείνει ένα λογικό πίνακα όπως επιθυμεί, καθώς και να ορίζει δικούς του ιδιωτικούς λογικούς πίνακες. Το σύστημα πρέπει να διατηρεί μεταδεδομένα για τους λογικούς πίνακες που έχουν ορισθεί, για τις στήλες αυτών, καθώς και για την αντιστοίχιση

των στηλών στα γενικά πεδία. Επίσης, επειδή όλα τα δεδομένα αποθηκεύονται στη βάση σε αλφαριθμητική μορφή, πρέπει το σύστημα να μεριμνά για τον έλεγχο των τύπων τους.

Η τεχνική αυτή, σε αντίθεση με την τεχνική των πινάκων επέκτασης, έχει το πλεονέκτημα ότι για μια εγγραφή ενός λογικού πίνακα δε χρειάζεται να ανακατασκευαστεί μέσω μιας λειτουργίας συνδέσμου. Επίσης, πέραν από τους κοινούς πίνακες οι tenants μπορούν να ορίσουν και δικούς τους ιδιωτικούς, οι οποίοι, επειδή αντιστοιχίζονται στο γενικό πίνακα, δεν αυξάνουν τον αριθμό των φυσικών πινάκων της βάσης.

Εν τούτοις, ειδικότερα για στενούς λογικούς πίνακες, αναγκαζόμαστε να διατηρούμε στις εγγραφές τους πολλές τιμές NULL. Οι τιμές αυτές, παρότι αντιμετωπίζονται αρκετά αποδοτικά από τα σύγχρονα ΣΔΒΔ, καταλαμβάνουν επιπλέον αποθηκευτικό χώρο. Συν τοις άλλοις, το γεγονός ότι ένα γενικό πεδίο μπορεί να περιέχει τιμές όχι μόνο από διαφορετικούς tenants, αλλά και από διαφορετικούς πίνακες, καθιστά ανώφελη τη δημιουργία ευρετηρίου για αυτό. Στον Πίνακα 2-9 βλέπουμε πώς θα έμοιαζε ένας τέτοιος γενικός πίνακας για το παράδειγμα μας.

tenant_id	table_id	custom_field_1	custom_field_2	custom_field_3	custom_field_4	custom_field_5	...	custom_field_N
1	1	1	Product #432 Launch	market research for new product	19/11/2013	TRUE	...	null
2	1	1	Laptop vs tablet	null	5/12/2013	100	...	null
1	2	1	Eleni Tsaroucha	Filippidou 12	Peristeri	2105767890	...	null
1	1	2	New-born Lion Name	Give a name to our new lion cub	10/1/2014	FALSE	...	null
3	1	1	Customer Satisfaction	Yearly customer satisfaction survey	7/11/2013	null	...	null
2	2	1	Miltos Katsos	Skoufa 27	Athens	2103233456	...	null
1	2	2	Maria Leonardou	Evrou 42	Athens	2103013223	...	null
2	1	2	Best Radio 2012	Radio station awards	20/1/2014	150	...	null

Πίνακας 2-9

(παράδειγμα universal table)

Pivot Tables

Εδώ, κάθε πεδίο ενός λογικού πίνακα αποθηκεύεται σε μια ξεχωριστή εγγραφή ενός pivot table. Ένας τέτοιος πίνακας περιέχει πεδία μεταδεδομένων για τον tenant, τη στήλη, τη γραμμή και το λογικό πίνακα. Μπορούμε να ορίσουμε είτε ένα pivot table αλφαριθμητικού τύπου(varchar), όπου θα αποθηκεύουμε(μετά από μετατροπή) δεδομένα κάθε τύπου, είτε ένα pivot table για κάθε υποστηριζόμενο τύπο δεδομένων. Η δεύτερη προσέγγιση έχει το πλεονέκτημα της υποστήριξης ευρετηρίων του ΣΔΒΔ.

Σημαντικό μειονέκτημα της τεχνικής αυτής, είναι ότι για κάθε τιμή πραγματικών δεδομένων της βάσης πρέπει να αποθηκεύουμε και ένα μεγάλο αριθμό από μεταδεδομένα. Επιπλέον, για την ανακατασκευή ενός λογικού πίνακα ν στηλών πρέπει να εκτελέσουμε ν-1 λειτουργίες συνδέσμου(joins) στα κατάλληλα pivot tables και για κατάλληλες συνθήκες στα μεταδεδομένα. Μείωση των απαιτούμενων λειτουργιών συνδέσμου μπορούμε να έχουμε με το να αποθηκεύουμε σε pivot tables μόνο τις τιμές πεδίων επέκτασης, ενώ τα κοινά πεδία θα αποθηκεύονται σε διαμοιραζόμενους φυσικούς πίνακες της βάσης.

Chunk Tables

Στο [12] προτείνεται η τεχνική των Chunk Tables(Πίνακες κομματιών). Τα chunk tables μοιάζουν με τα pivot tables που είδαμε παραπάνω, με τη διαφορά ότι αντί για ένα πεδίο δεδομένων έχουν ένα συνδυασμό από πεδία διαφόρων τύπων. Στα πεδία αυτά μπορούν να ορίζονται και ευρετήρια.

Ένας λογικός πίνακας διαμερίζεται κάθετα σε υποσύνολα στηλών του, και κάθε υποσύνολο αποκτά ένα αναγνωριστικό(chunk_id) και αντιστοιχίζεται σε κατάλληλο chunk table. Η διαμέριση αυτή πρέπει να γίνεται με τέτοιο τρόπο ώστε να ελαχιστοποιείται ο αριθμός των κομματιών. Τελικά, μια εγγραφή ενός λογικού πίνακα, διασπάται σε κομμάτια, και κάθε κομμάτι αποθηκεύεται στο αντίστοιχο chunk table. Για την ανακατασκευή ολόκληρης της εγγραφής απαιτούνται τόσες λειτουργίες συνδέσμου όσα και τα κομμάτια στα οποία έχει διασπαστεί.

Κάθε chunk table, περιέχει πεδία μεταδεδομένων που δείχνουν σε ποιον tenant, αλλά και σε ποιό chunk ενός λογικού πίνακα, ανήκει κάθε εγγραφή του. Ο συνδυασμός των αναγνωριστικών για τον tenant, τον πίνακα και της γραμμής(row) σε αυτόν, ορίζει

μονοσήμαντα μια εγγραφή ενός λογικού πίνακα. Τα ίδια πεδία, με τη συμπερίληψη του αναγνωριστικού του chunk, αποτελούν το πρωτεύον κλειδί κάθε chunk table.

Σε σχέση με την τεχνική των pivot tables, η τεχνική αυτή μειώνει το λόγο μεταδεδομένων προς πραγματικά δεδομένα, ενώ σε σχέση με την τεχνική του Γενικού Πίνακα(Universal Table) προσθέτει τη δυνατότητα δημιουργίας ευρετηρίων και μπορεί να αποφύγει κάποιες από τις περιττές τιμές null. Παρ'όλα αυτά, ο σχεδιαστής μιας multi-tenant βάσης δεδομένων που χρησιμοποιεί την τεχνική αυτή, πρέπει να αποφασίσει για τον αριθμό και το μέγεθος των chunk tables, αλλά και τους τύπους δεδομένων που αυτά θα περιέχουν. Οι αποφάσεις αυτές, πρέπει να ληφθούν με γνώμονα την ελαχιστοποίηση των λειτουργιών συνδέσμου.

Στους πίνακες 2-10 και 2-11, βλέπουμε πως θα μπορούσαν να αποθηκεύονται τα δεδομένα του πίνακα ερευνών(μαζί με τις επεκτάσεις κάθε tenant) που είδαμε παραπάνω, με τη χρήση της τεχνικής των Chunk Tables.

tenant_id	table_id	chunk	row	numeric	string
1	1	1	1	1	Product #432 Launch
1	1	2	1	0.6	market research for new product
2	1	1	1	1	Laptop vs tablet
2	1	2	1	100	null
1	1	1	2	2	New-born Lion Name
1	1	2	2	1	Give a name to our new lion cub
3	1	1	1	1	Customer Satisfaction
3	1	2	1	null	Yearly customer satisfaction survey
2	1	1	2	2	Best Radio 2012
2	1	2	2	150	Radio station awards

Πίνακας 2-10

tenant_id	table_id	chunk	row	date	boolean
1	1	3	1	19/11/2013	TRUE
2	1	3	1	5/12/2013	null
1	1	3	2	10/1/2014	FALSE
3	1	3	1	7/11/2013	null
2	1	3	2	20/1/2014	null

Πίνακας 2-11

Φυσικά, σε ένα πραγματικό σύστημα, τα chunk tables θα έπρεπε να επιλεχθούν αρκετά μεγαλύτερα για τη βελτίωση της απόδοσης του(όπως φαίνεται και από τα πειραματικά αποτέλέσματα που παρουσιάζονται στο [12]).

Chunk Folding

Στην τεχνική αυτή, που αποτελεί παραλλαγή της τεχνικής των Chunk Tables, ένας λογικός πίνακας διασπάται κάθετα σε ένα συνηθισμένο φυσικό πίνακα και σε κομμάτια(chunks) που αποθηκεύονται σε κατάλληλα Chunk Tables. Για τους κοινούς πίνακες των tenants, τα κοινά πεδία αυτών τοποθετούνται στο φυσικό πίνακα, ενώ οι επεκτάσεις τους μπαίνουν στα chunk tables. Όπως και πριν, η επιλογή της κατάλληλης διαμέρισης και η αντιστοίχιση κάθε κομματιού στο κατάλληλο chunk table, πρέπει να γίνεται προσεχτικά ώστε να βελτιστοποιείται η απόδοση του συστήματος.

.

2.3 Η περίπτωση της Salesforce.com

Η "salesforce.com", πέραν από το ομώνυμο Σύστημα Διαχείρισης Πελατειακών Σχέσεων(CRM) που προσφέρει μέσω του Νέφους, διαθέτει στους πελάτες της και μια πλατφόρμα(PaaS) ανάπτυξης και φιλοξενίας multi-tenant εφαρμογών. Τόσο η πλατφόρμα αυτή, όσο και το CRM σύστημα, χρησιμοποιούν τη multi-tenant προσέγγιση και σε επίπεδο εφαρμογής αλλά και στην αποθήκευση των δεδομένων. Στο [14] αναλύεται η αρχιτεκτονική της πλατφόρμας αυτής και στην παράγραφο αυτή παρουσιάζουμε σύντομα την προσέγγιση και τις τεχνικές που χρησιμοποιεί σχετικά με την αποθήκευση των δεδομένων.

Κατ'αρχάς, η πλατφόρμα αυτή χρησιμοποιεί την προσέγγιση διαμοιραζόμενου σχήματος. Οι λογικοί πίνακες κάθε tenant αντιστοιχίζονται σε κοινούς φυσικούς πίνακες μέσω ενός συνδυασμού κάποιων από τις τεχνικές που εξετάσαμε πριν. Συγκεκριμένα, τα δεδομένα των tenants αποθηκεύονται σε ένα ευρύ γενικό πίνακα(universal table). Στον πίνακα αυτό αποθηκεύονται οι εγγραφές όλων των λογικών πινάκων που διαθέτουν οι tenants και το σύστημα διατηρεί πίνακες μεταδεδομένων για να αντιστοιχήσει κάθε λογικό σχήμα στο φυσικό σχήμα της βάσης. Για τη μείωση των λειτουργιών εισόδου-εξόδου, το σύστημα αποθηκεύει προσωρινά(caching) τα μεταδεδομένα που φορτώνει στη μνήμη. Επειδή η χρήση ενός μόνο γενικού πίνακα με πεδία αλφαριθμητικού τύπου καθιστά αναποτελεσματικά(όπως είδαμε κατά την ανάλυση της τεχνικής του Γενικού Πίνακα) τα ευρετήρια του ΣΔΒΔ,

δημιουργούνται στη βάση και πίνακες(pivot tables) που χρησιμοποιούνται ως ευρετήρια ή για τον έλεγχο περιορισμών(π.χ. περιορισμοί μοναδικότητας ή ξένου κλειδιού) σε πεδία που ορίζουν οι tenants. Π.χ. όταν σε ένα πεδίο ορίζεται ένα ευρετήριο, οι τιμές που δίνονται στο πεδίο αυτό, πέραν από το γενικό πίνακα, αποθηκεύονται και σε ένα πίνακα ευρετηρίου(index pivot table) στο σωστό τύπο δεδομένων(και όχι μόνο ως αλφαριθμητικά). Στον πίνακα αυτόν, ορίζεται ένα ευρετήριο του ΣΔΒΔ και, κατά την εκτέλεση ενός ερωτήματος, το σύστημα μπορεί να χρειαστεί να ανατρέξει πρώτα σε εκείνον και μετά στο γενικό πίνακα. Για την αποδοτική χρήση των ευρετηρίων αυτών, το σύστημα διαθέτει ένα πολύπλοκο βελτιστοποιητή ερωτημάτων, πέραν από το βελτιστοποιητή που διαθέτει το χρησιμοποιούμενο ΣΔΒΔ. Οι λειτουργίες ορισμού δεδομένων(DDL) που εκτελεί κάθε tenant, δεν αντιστοιχούν σε λειτουργίες DDL στο ΣΔΒΔ. Αντ'αυτού, κάθε τέτοια λειτουργία οδηγεί σε αλλαγές μόνο στους πίνακες μεταδεδομένων. Κάθε εγγραφή σε ένα λογικό πίνακα αποκτά και ορίζεται μονοσήμαντα από ένα αναγνωριστικό τύπου GUID(globally unique identifier). Έτσι, αποφεύγεται η χρήση πολύπλοκων πρωτευόντων κλειδιών.

3

Η multi-tenant προσέγγιση του συστήματος

Όπως προαναφέραμε, στα πλαίσια της παρούσας διπλωματικής εργασίας, αναπτύσσουμε ένα σύστημα που θα επιτρέπει την υποστήριξη multi-tenant βάσεων δεδομένων από ένα σχεσιακό ΣΔΒΔ το οποίο, γενικά, είναι προσανατολισμένο σε βάσεις δεδομένων ενός tenant. Παρότι θα εξετάσουμε με λεπτομέρεια τα διάφορα συστατικά του συστήματος σε επόμενα κεφάλαια, στο κεφάλαιο αυτό παρουσιάζουμε τη γενική προσέγγιση του και διατυπώνουμε μια παραλλαγή κάποιων από τις τεχνικές που εξετάσαμε στο Κεφάλαιο 2 για την εξατομίκευση του σχήματος στις multi-tenant βάσεις δεδομένων.

Αρχικά, αν και κάθε προσέγγιση από τις τέσσερις που εξετάσαμε στην παράγραφο 2.1 μπορεί να ενδείκνυται σε συγκεκριμένες εφαρμογές, η προσέγγιση που επιλέξαμε να ακολουθήσουμε για το σύστημα μας είναι η προσέγγιση διαμοιραζόμενου σχήματος. Η προσέγγιση αυτή, όπως είδαμε, μπορεί να μεγιστοποιήσει τον αριθμό των tenants που μοιράζονται τους ίδιους φυσικούς υπολογιστικούς πόρους. Με άλλα λόγια, είναι η πιο οικονομικά ενδιαφέρουσα προσέγγιση. Επίσης, ενώ οι υπόλοιπες προσεγγίσεις μπορούν να χρησιμοποιηθούν με μικρό προγραμματιστικό κόπο, όταν χρησιμοποιείται διαμοιραζόμενο σχήμα πρέπει να δημιουργηθεί ένα πολύπλοκο επίπεδο λογισμικού. Αυτό το πρόβλημα προσπαθούμε να αντιμετωπίσουμε με το σύστημα που αναπτύσσουμε. Συγκεκριμένα, το σύστημα θα μπορεί να χρησιμοποιηθεί από εφαρμογές που χρησιμοποιούν κοινό σχήμα για τη βάση δεδομένων τους, χωρίς να απαιτείται να αναπτυχθεί εκ νέου η λογική αντιστοίχισης των δεδομένων των tenants στους κοινούς πίνακες της βάσης.

3.1 Τεχνική αντιστοίχισης στο φυσικό σχήμα

Όταν τα δεδομένα πολλαπλών tenants αποθηκεύονται στους ίδιους πίνακες αναγκαζόμαστε να διαφοροποιήσουμε την έννοια του λογικού σχήματος ενός tenant από το φυσικό σχήμα το οποίο υπάρχει στη βάση. Οι τεχνικές αντιστοίχισης που είδαμε στην παράγραφο 2.3 κυμαίνονται από τη χρήση κοινών πινάκων χωρίς τη δυνατότητα εξατομίκευσης του σχήματος, μέχρι τη χρήση γενικών δομών(universal table, chunk tables κ.λ.π.) όπου αποθηκεύονται όλοι οι λογικοί πίνακες των tenants.

Αν και υπάρχουν εφαρμογές που μπορεί να μην απαιτούν δυνατότητα εξατομίκευσης του σχήματος δεδομένων, το σύστημα πρέπει εν γένει να μπορεί να υποστηρίξει και εξατομίκευση κοινών πινάκων, αλλά και δυνατότητα δημιουργίας νέων πινάκων από κάθε tenant. Οι τεχνικές που αποθηκεύουν κάθε εγγραφή σε κάποια γενική δομή(π.χ. universal table), παρά το ότι ικανοποιούν τις παραπάνω απαιτήσεις, έχουν το μειονέκτημα ότι μετατρέπουν τη βάση σε μια απλή αποθήκη δεδομένων που δε μπορεί να αξιοποιήσει αποτελεσματικά τη λειτουργικότητα που προσφέρει το ΣΔΒΔ. Από την άλλη, τεχνικές όπως τα preallocated fields ή η χρήση xml επιτυγχάνουν περιορισμένη εξατομίκευση, καθώς είτε περιορίζουν τον αριθμό των πεδίων-επεκτάσεων είτε απαιτούν τη δημιουργία ενός φυσικού πίνακα για κάθε λογικό πίνακα ενός tenant. Το τελευταίο σημαίνει ότι αν κάθε tenant δημιουργεί δικούς του ιδιωτικούς πίνακες, ο αριθμός των tenants σε κάθε βάση θα περιορίζεται από το μέγιστο αριθμό πινάκων που αυτή μπορεί να υποστηρίξει.

Αρχικά, διαπιστώνουμε ότι κάθε εφαρμογή, όση δυνατότητα για εξατομίκευση και να προσφέρει, συνήθως απαιτεί ένα σύνολο από βασικούς πίνακες και πεδία αυτών για να λειτουργήσει. Οι πίνακες και τα πεδία αυτά είναι κοινά για κάθε ενοικιαστή της εφαρμογής. Αντί να αντιστοιχήσουμε τους πίνακες αυτούς σε κάποια γενική δομή, επιλέγουμε να χρησιμοποιήσουμε ένα φυσικό πίνακα της βάσης για κάθε έναν από αυτούς. Οι πίνακες αυτοί, που στο εξής θα τους αποκαλούμε **βασικούς πίνακες(standard tables)**, μπορούν να αξιοποιήσουν σε μεγάλο βαθμό τη λειτουργικότητα που προσφέρει το ΣΔΒΔ για δημιουργία ευρετηρίων, έλεγχο τύπων, έλεγχο περιορισμών κ.λ.π. Ένας βασικός πίνακας είναι ένας κανονικός πίνακας της βάσης, αντίστοιχος με τον πίνακα που θα υπήρχε σε μια single-tenant βάση δεδομένων. Η μόνη αναγκαία διαφορά είναι η ύπαρξη ενός πεδίου που δηλώνει τον tenant στον οποίο ανήκει μια εγγραφή. Παράδειγμα ενός βασικού πίνακα είναι ο πίνακας ερευνών που είδαμε στον Πίνακα 2-1.

Για την επέκταση ενός βασικού πίνακα η τεχνική που προτείνουμε συνδυάζει τα preallocated fields με μια παραλλαγή της τεχνικής των chunk tables. Παρά τους περιορισμούς που είδαμε ότι έχει η τεχνική των preallocated fields, εν τούτοις, προοσφέρει ένα απλό τρόπο για τη δημιουργία πεδίων-επεκτάσεων σε ένα βασικό πίνακα. Επιπλέον, επειδή τα πεδία αυτά

αποθηκεύονται μαζί με τα κοινά πεδία του πίνακα, δε χρειάζεται κάποια λειτουργία συνδέσμου για τη φόρτωση ολόκληρης της σχέσης. Έτσι, κατά τη δημιουργία της multi-tenant βάσης, ο σχεδιαστής θα μπορεί να προσθέσει σε κάθε βασικό πίνακα έναν αριθμό από γενικά πεδία. Ο αριθμός των πεδίων αυτών θα καθορίζεται από το σχεδιαστή και μπορεί να διαφέρει από πίνακα σε πίνακα. Ένας βασικός πίνακας μπορεί να έχει ένα μεγάλο πλήθος από γενικά πεδία, ενώ ένας άλλος μπορεί να μην έχει κανένα. Τα γενικά αυτά πεδία θα είναι τύπου varchar(αλφαριθμητικά). Με άλλα λόγια, κάθε τιμή που θα αποθηκεύεται σε αυτά, θα μετατρέπεται πρώτα σε αλφαριθμητικό και ο αντίστοιχος έλεγχος θα πρέπει να γίνεται από το σύστημα μας.

Για την περίπτωση που ο αριθμός των γενικών πεδίων δεν αρκεί για τα πεδία-επεκτάσεις ενός tenant, θα χρησιμοποιήσουμε μια δομή ανάλογη με τα chunk tables. Η ίδια δομή θα χρησιμοποιείται και για την αποθήκευση ιδιωτικών πινάκων που μπορεί να ορίσει κάθε tenant. Είδαμε ότι με την τεχνική των chunk tables μπορούμε να έχουμε πολλούς τέτοιους πίνακες, καθένας με διαφορετικό πλήθος και συνδυασμό τύπων πεδίων. Όμως, δεν υπάρχει γενικά κάποιος τρόπος για τον προσδιορισμό αυτών των χαρακτηριστικών. Επίσης, κακή επιλογή τους μπορεί να οδηγήσει σε μη αποδοτική διαμέριση των λογικών σχέσεων σε chunks και συνεπώς σε αύξηση των απαιτούμενων joins και τελικά σε μείωση της απόδοσης του συστήματος.

Στην τεχνική που προτείνουμε, αποφεύγουμε τις παραπάνω δυσκολίες με το να χρησιμοποιήσουμε ένα chunk table με γενικά πεδία τύπου varchar. Δηλαδή, όπως και στα γενικά πεδία(preallocated fields) σε ένα βασικό πίνακα, όλες οι τιμές στα πεδία του chunk table θα μετατρέπονται και θα αποθηκεύονται ως αλφαριθμητικά. Έτσι, η μόνη επιλογή του σχεδιαστή ως προς το chunk table, θα είναι η επιλογή του μεγέθους του, δηλαδή του πλήθους των γενικών πεδίων του. Σε αντίθεση με την αρχική τεχνική των Chunk Tables, που προτείνεται στο [12], η τεχνική που θα ακολουθήσουμε επιτρέπει, επιπλέον, ευκολότερη αλλαγή στη μορφή του chunk table(ουσιαστικά αλλαγή του μεγέθους του), αν διαπιστωθεί πιθανή βελτίωση από τέτοια αλλαγή.

Συνεπώς, μια λογική σχέση ενός tenant μπορεί να διαμερίζεται κάθετα και κάποια από τα πεδία της να αποθηκεύονται σε ένα βασικό πίνακα και κάποια άλλα στο Chunk Table. Αν τα πεδία δε χωράνε σε μια εγγραφή του Chunk Table, το σύστημα θα μπορεί να δημιουργεί παραπάνω από ένα chunks για κάθε σχέση. Τελικά, δηλαδή, για την ανακατασκευή ολόκληρης της σχέσης μπορεί να απαιτηθούν λειτουργίες συνδέσμου ανάμεσα σε δυο μόνο φυσικούς πίνακες της βάσης: σε ένα βασικό πίνακα και στο Chunk Table. Βέβαια, αν μια σχέση αποτελείται απο πολλαπλά chunks θα απαιτούνται αντίστοιχες λειτουργίες συνδέσμου του Chunk Table με τον εαυτό του(self-join), με κατάλληλες τιμές του πεδίου μεταδεδομένων που θα καθορίζει το κάθε chunk σε μια σχέση.

Στο [12], που παρουσιάζεται η αρχική τεχνική των Chunk Tables, σημειώνεται ότι μπορεί πολλαπλά chunks μιας λογικής σχέσης να αποθηκεύονται στο ίδιο chunk table, με αποτέλεσμα πιθανώς καλύτερη χρήση της ενδιάμεσης μνήμης του ΣΔΒΔ όταν φορτώνεται η σχέση. Στην παραλλαγή που διατυπώνουμε εμείς, το επιχείρημα αυτό αποκτά μεγαλύτερη ισχύ, καθώς όλα τα chunks θα αποθηκεύονται σε ένα μόνο chunk table και πολλαπλά chunks μεταφράζονται απλά σε παραπάνω self-joins(σύνδεσμος ενός πίνακα με τον εαυτό του). Συνεπώς, ειδικότερα αν εξασφαλίσουμε ότι τα chunks κάθε λογικής εγγραφής αποθηκεύονται σειριακά το ένα μετά το άλλο, είναι πιθανότερο να χρειαστούμε λιγότερες αναγνώσεις σελίδων για να φορτωθεί μια εγγραφή από το αν είχαμε πολλαπλά chunk tables.

Το ότι αποθηκεύουμε τις τιμές όλων των ιδιωτικών πεδίων σε μορφή VARCHAR συνεπάγεται αδυναμία δημιουργίας ευρετηρίων του ΣΔΒΔ σε αυτά. Έτσι, όπως και στην τεχνική του Γενικού Πίνακα που χρησιμοποιείται από την πλατφόρμα salesforce.com, αν θέλουμε να υποστηρίζουμε ευρετήρια σε τέτοια πεδία, θα πρέπει και στην τεχνική μας να δημιουργήσουμε εμείς κατάλληλο μηχανισμό. Εντούτοις, στα κοινά πεδία των βασικών πινάκων(τα οποία θα είναι συνήθως και τα σημαντικότερα για την multi-tenant εφαρμογή που χρησιμοποιεί τη βάση) μπορούμε να ορίσουμε κανονικά ευρετήρια του ΣΔΒΔ.

Η επιλογή του αριθμού των γενικών πεδίων σε ένα βασικό πίνακα, αλλά και του μεγέθους του Chunk Table, θα αποτελεί, όπως είπαμε, επιλογή του σχεδιαστή της multi-tenant βάσης. Το σύστημα που υλοποιούμε, δηλαδή, θα δέχεται τις συγκεκριμένες τιμές ως παραμέτρους. Μεγαλύτερες τιμές για αυτές τις παραμέτρους μπορεί να μεταφράζονται σε πολλές τιμές null για tenants που δεν απαιτούν τόσα πολλά πεδία-επεκτάσεις. Από την άλλη, μικρότερες τιμές θα οδηγούν σε παραπάνω λειτουργίες συνδέσμου για tenants που επιθυμούν να ορίσουν περισσότερα πεδία. Αν και ο προσδιορισμός των τιμών των παραμέτρων θα πρέπει να γίνει αφού ληφθούν υπόψιν οι προβλεπόμενες ανάγκες των tenants για εξατομίκευση, εν τούτοις, είναι προτιμότερο να δοθούν υψηλότερες τιμές. Από τα πειραματικά αποτελέσματα που παρουσιάζονται στο [12] σχετικά με τα Chunk Tables συμπεραίνουμε ότι μεγαλύτερα chunk tables οδηγούν σε πολύ καλύτερη απόδοση, η οποία συναγωνίζεται την απόδοση από τη χρηση συμβατικών πινάκων. Η βελτίωση αυτή σχετίζεται με την ελαχιστοποίηση των λειτουργιών συνδέσμου, που σε γενικές γραμμές έχουν σχετικά μεγάλο υπολογιστικό κόστος. Τα αποτελέσματα αυτά μπορούν να μεταφερθούν αυτούσια και στη δική μας περίπτωση του ενός chunk table με πεδία τύπου varchar.

3.2 Πρωτεύοντα Κλειδιά

Αν το σύστημα υποστήριζε μόνο βασικούς πίνακες, τότε η μόνη αλλαγή που θα απαιτούνταν στο πρωτεύον κλειδί ενός πίνακα ώστε να υποστήριζε πολλαπλούς tenants, θα ήταν η συμπερίληψη και του πεδίου "tenant_id". Το ότι, όμως, επεκτάσεις βασικών πινάκων αλλά και ιδιωτικές σχέσεις των tenants αποθηκεύονται στο ίδιο chunk table, μας αναγκάζει να χρησιμοποιήσουμε ένα ενιαίο τρόπο για το μοναδικό προσδιορισμό κάθε λογικής εγγραφής της βάσης(είτε ανήκει σε βασικό πίνακα είτε σε ιδιωτικό).

Η πρώτη και πιο φυσιολογική επιλογή για τα πρωτεύοντα κλειδιά θα ήταν η επιλογή που αναφέραμε όταν αναλύαμε την αρχική τεχνική των Chunk Tables(Παρ. 2.2). Δηλαδή, κάθε λογική εγγραφή ενός πίνακα, είτε βασικού είτε ιδιωτικού, να ορίζεται μονοσήμαντα από το συνδυασμό των αναγνωριστικών για τον tenant και τον πίνακα, αλλά και από ένα πεδίο row, που διακρίνει τις εγγραφές μέσα σε ένα λογικό πίνακα. Η χρήση, όμως, συνδυασμού πεδίων ως πρωτεύον κλειδί, πέραν από πιθανές επιπτώσεις στην απόδοση, δυσχεραίνει τη δημιουργία συσχετίσεων ανάμεσα στους πίνακες. Πράγματι, για κάθε εξάρτηση ξένου κλειδιού θα έπρεπε να χρησιμοποιούμε ολόκληρο το συνδυασμό πεδίων.

Μια διαφορετική προσέγγιση θα ήταν η χρήση για κάθε εγγραφή ενός αναγνωριστικού μοναδικού ανάμεσα στις εγγραφές όλων των σχέσεων όλων των tenants. Το αναγνωριστικό αυτό μπορεί να είναι ένα αναγνωριστικό τύπου GUID(Globally Unique Identifier). Ένα GUID είναι μια τιμή των 128 bit η οποία παριστάνεται συνήθως ως 32 δεκαεξαδικά ψηφία χωρισμένα σε ομάδες από άνω παύλες[15]. Οι αριθμοί αυτοί παράγονται από αλγόριθμους που επιλέγονται ώστε η πιθανότητα επανεμφάνισης της ίδιας τιμής να είναι από πολύ μικρή έως μηδαμινή. Τα GUIDs έχουν αρκετές χρήσεις. Για παράδειγμα, χρησιμοποιούνται ως πρωτεύοντα κλειδιά σε κατανεμημένες βάσεις δεδομένων, έτσι ώστε να εξασφαλιστεί ότι η τιμή σε ένα κλειδί είναι μοναδική σε κάθε κόμβο του κατανεμημένου συστήματος.

Όταν τα GUIDs χρησιμοποιούνται ως πρωτεύοντα κλειδιά σε σχέσεις, μπορεί να επηρεάσουν αρνητικά την απόδοση της βάσης κατά την εισαγωγή νέων εγγραφών. Αυτό συμβαίνει διότι οι περισσότεροι αλγόριθμοι παράγουν τυχαίες τιμές με αποτέλεσμα οι εγγραφές να πρέπει να τοποθετηθούν σε κάποια τυχαία θέση μέσα στη σχέση αντί στο τέλος της. Για την αντιμετώπιση του προβλήματος αυτού, έχουν αναπτυχθεί αρκετοί αλγόριθμοι παραγωγής ακολουθιακών GUIDs οι οποίοι, παράλληλα, καθιστούν απίθανη την παραγωγή της ίδιας τιμής παραπάνω από μια φορά.

Χρησιμοποιώντας, λοιπόν, αναγνωριστικά τύπου GUID, προσφέρουμε στο σύστημα μας ένα απλούστερο τρόπο μοναδικού προσδιορισμού κάθε εγγραφής και διευκολύνουμε σημαντικά τη δημιουργία εξαρτήσεων ξένου κλειδιού. Έτσι, για τη συσχέτιση δυο σχέσεων αρκεί η

δημιουργία ενός πεδίου σε μία από αυτές που θα περιέχει ένα τέτοιο αναγνωριστικό το οποίο θα δείχνει σε μια εγγραφή της άλλης σχέσης.

Έχοντας αναφερθεί και στα πρωτεύοντα κλειδιά των σχέσεων, μπορούμε τώρα να δούμε ένα παράδειγμα αντιστοίχισης μιας λογικής σχέσης στο φυσικό σχήμα της βάσης. Στον Πίνακα 3-1 βλέπουμε πώς αντιλαμβάνεται το βασικό πίνακα "surveys" ο tenant 1. Στον πίνακα αυτόν, που υπάρχει στο λογικό σχήμα όλων των tenants, τα πεδία "survey_title", "description" και "end_date" είναι κοινά για όλους τους tenants. Τα υπόλοιπα πεδία("is_open", "version", "company", "summary") τα έχει προσθέσει ο tenant 1 και είναι ιδιωτικά για αυτόν.

guid	survey_title	description	end_date	is_open	version	company	summary
16eddf6f-9bc4-11e3-baa4-45cae7864bef	Product #432 Launch	market research for new product	19/2/14	TRUE	0.6	FrozenYo Co.	null
9f4c315c-9bc5-11e3-9d40-39b8e19902c6	New-born Lion Name	Give a name to our new lion cub	5/3/14	FALSE	1	Safari Zoo Park	null

Πίνακας 3-1

Στους πίνακες 3-2 και 3-3 βλέπουμε το πώς αποθηκεύονται τα δεδομένα αυτά στους φυσικούς πίνακες της βάσης. Στον 3-2 βλέπουμε το φυσικό βασικό πίνακα "surveys" που, εκτός από τα βασικά πεδία, έχει και δυο γενικές στήλες(prealloc_field_1 και prealloc_field_2). Στον 3-3 βλέπουμε το Chunk table της βάσης. Όταν ένας tenant προσθέτει ένα ιδιωτικό πεδίο σε ένα βασικό πίνακα θα πρέπει πρώτα να ελέγχουμε αν αυτός έχει κάποια ελεύθερη γενική στήλη(δηλαδή γενική στήλη που δε χρησιμοποιείται ήδη από τον tenant αυτό). Αν υπάρχει, θα πρέπει να αντιστοιχίζουμε το νέο ιδιωτικό πεδίο σε αυτή. Αν όχι, τότε μόνο θα αντιστοιχίζουμε το νέο πεδίο σε κάποια στήλη ενός chunk. Τα πεδία "is_open" και "version" αποθηκεύονται στις δυο γενικές στήλες του βασικού πίνακα "surveys". Τα υπόλοιπα πεδία-επεκτάσεις αποθηκεύονται στο Chunk Table, σε chunk που ορίσαμε(με αριθμό chunk 1) για αυτόν τον πίνακα και για αυτόν τον tenant. Για την ανακατασκευή ολόκληρου του λογικού πίνακα "surveys" για τον tenant 1, θα πρέπει να κάνουμε join ανάμεσα στο φυσικό πίνακα "surveys" και στο Chunk Table(με κατάλληλες συνθήκες στις στήλες μεταδεδομένων chunk_num, table_id, tenant_id και με συνθήκη συνδέσμου στη στήλη guid).

guid	tenant_id	survey_title	description	end_date	prealloc_field_1	prealloc_field_2
16eddf6f-9bc4-11e3-baa4-45cae7864bef	1	Product #432 Launch	market research for new product	19/2/14	TRUE	0.6
9f4c315c-9bc5-11e3-9d40-39b8e19902c6	1	New-born Lion Name	Give a name to our new lion cub	5/3/14	FALSE	1
e4ba0f62-9bc8-11e3-8fb1-6704bf635b3f	2	Laptop vs tablet	null	10/6/11	100	null
e4ba365b-9bc8-11e3-9b35-f3a6b61fd0ad	2	Best Radio 2012	radio station awards	22/12/12	150	null

Πίνακας 3-3

guid	chunk_num	table_id	tenant_id	custom_field_1	custom_field_2	custom_field_3	custom_field_4
16eddf6f-9bc4-11e3-baa4-45cae7864bef	1	2	1	FrozenYo Co.	null	null	null
3aae3b07-9bd0-11e3-b493-	1	6	2	Nikos Papadopoulos	Aristotelous 32	Athens	2103421300
9f4c315c-9bc5-11e3-9d40-39b8e19902c6	1	2	1	Safari Zoo Park	null	null	null
baae88cb-9bd0-11e3-9ef5-47031bc9939b	1	6	2	Maria Louka	Smirnis 21	Peristeri	2105723450

Πίνακας 3-3

3.3 Επανεγγραφή ερωτημάτων

3.3.1 Ερωτήματα ανάκτησης δεδομένων

Η διαφοροποίηση του λογικού σχήματος από το φυσικό σημαίνει ότι ένα SQL ερώτημα που στέλνει στη βάση ένας tenant, και το οποίο αναφέρεται στο λογικό του σχήμα, δε μπορεί να εκτελεσθεί αυτούσιο από αυτή. Θα πρέπει να ξαναγράφεται από το σύστημα σε μορφή που να αντιστοιχεί στο πραγματικό σχήμα που διαθέτει η βάση.

Έστω, για παράδειγμα, ότι ο tenant 1 στέλνει στη βάση το ακόλουθο ερώτημα ανάκτησης δεδομένων(select query):

```
SELECT   *
FROM  surveys s
WHERE  s.is_open = 'true'
```

Στο ερώτημα αυτό βλέπουμε ότι αναφέρεται η σχέση “surveys”. Η σχέση αυτή, όπως είδαμε στον Πίνακα 3-1, αν και υπάρχει για όλους τους tenants, για τον tenant 1 έχει επεκταθεί με αποτέλεσμα να χρησιμοποιούνται και τα γενικά πεδία του αντίστοιχου βασικού πίνακα, αλλά και το Chunk Table. Για την επανεγγραφή του ερωτήματος ως προς το φυσικό σχήμα αρκεί να αντικαταστήσουμε την αναφορά στη σχέση “surveys” με ένα υποερώτημα που ανακατασκευάζει όλο το λογικό πίνακα “surveys” μαζί με τα πεδία-επεκτάσεις του συγκεκριμένου tenant. Επιπλέον, ακόμα και αν δεν είχε επεκταθεί ο πίνακας, θα έπρεπε να προστεθεί μια συνθήκη για την επιλογή μόνο των εγγραφών του tenant που έστειλε το ερώτημα. Μιας και τα δεδομένα πολλαπλών ενοίκων αποθηκεύονται σε κοινούς πίνακες, η συνθήκη αυτή είναι πολύ σημαντική αφού προσφέρει ένα είδος λογικής απομόνωσης ανάμεσα στα δεδομένα διαφορετικών tenants. Το ερώτημα αποκτά τελικά την ακόλουθη μορφή:

```
SELECT *
FROM
   (SELECT  surveys.guid, surveys.survey_title,
        surveys.description,  surveys.end_date,
        CAST(surveys.prealloc_field_1 AS BOOLEAN) AS is_open,
        surveys.prealloc_field_2 AS VERSION,
        chunk1.custom_field_1  AS company,
        chunk1.custom_field_2  AS summary
    FROM   surveys,
              chunk_table AS chunk1
    WHERE surveys.tenant_id = 1 AND
              chunk1.tenant_id = 1 AND
              chunk1.table_id = 2 AND
              chunk1.chunk_num = 1 AND
              surveys.guid = chunk1.guid) AS s
WHERE s.is_open = 'true'
```

Παρατηρούμε ότι η αρχική αναφορά στον πίνακα “surveys” αντικαθίσταται από ένα υποερώτημα select στο οποίο εκτελείται μια λειτουργία συνδέσμου ανάμεσα στον αντίστοιχο βασικό πίνακα και στο Chunk Table. Η λογική σχέση έχει ένα μόνο chunk, οπότε το Chunk Table συνδέεται μόνο μια φορά. Τα γενικά πεδία που χρησιμοποιούνται, είτε ανήκουν στο βασικό πίνακα, είτε στο Chunk Table, μετονομάζονται, στα πλαίσια του ερωτήματος, ώστε να μπορούν να αναφερθούν με το όνομα του αντίστοιχου πεδίου-επέκτασης. Επίσης, μιας και είναι τύπου VARCHAR, ζητείται να μετατραπούν στον τύπο δεδομένων του πεδίου-επέκτασης. Μπορούμε επιπλέον να δούμε τις συνθήκες που αφορούν τις στήλες μεταδεδομένων(tenant_id, table_id, chunk_num) αλλά και τη συνθήκη συνδέσμου η οποία αφορά μόνο το πρωτεύον κλειδί guid των εγγραφών.

Το υποερώτημα που αντικαθιστά τη λογική σχέση “surveys” αυξάνει την πολυπλοκότητα του αρχικού ερωτήματος. Εν τούτοις, όπως αποδεικνύεται στο [17](και αναφέρεται στο [12]), το φώλιασμα υποερωτημάτων με μόνο συζευκτικά κατηγορήματα(όπως στην περίπτωση μας) μπορεί πάντα να αφαιρεθεί από ένα βελτιστοποιητή ερωτημάτων ο οποίος έχει ως έξοδο ένα

επίπεδο ερώτημα ισοδύναμο με το αρχικό. Αν και δεν υποστηρίζεται αυτή η λειτουργία από τους βελτιστοποιητές όλων των ΣΔΒΔ, το σύστημα που θα χρησιμοποιήσουμε εμείς(Postgresql) την υποστηρίζει. Για συστήματα που δε μπορούν να αφαιρέσουν το φώλιασμα θα έπρεπε να το κάναμε εμείς κατά την επανεγγραφή του αρχικού ερωτήματος. Αυτό, όμως, θα αύξανε πολύ την πολυπλοκότητα της επανεγγραφής.

Αν αντικαθιστούμε κάθε αναφορά σε μια λογική σχέση με ένα υποερώτημα που την ανακατασκευάζει ολόκληρη, ενδέχεται να οδηγηθούμε σε ερωτήματα με περιττά joins. Για παράδειγμα, ας εξετάσουμε το ακόλουθο ερώτημα:

```
SELECT   s.survey_title
FROM    surveys s
WHERE  s.is_open = 'true'
```

Το ερώτημα αυτό είναι σχεδόν ίδιο με το προηγούμενο, με τη διαφορά ότι αντί να ζητείται να επιστραφούν όλες οι στήλες του πίνακα “surveys”, ζητείται μόνο η στήλη “survey_title”. Και τα δυο πεδία του πίνακα “surveys” που αναφέρονται στο ερώτημα βρίσκονται στο βασικό πίνακα και όχι σε κάποιο chunk. Οπότε, η εκτέλεση μιας λειτουργίας συνδέσμου με το Chunk Table κρίνεται περιττή και θα μπορούσαμε να επανεγγράψουμε το ερώτημα ως εξής:

```
SELECT s.survey_title
FROM
    (SELECT surveys.survey_title,
                CAST(surveys.prealloc_field_1 AS BOOLEAN) AS is_open
     FROM surveys
     WHERE surveys.tenant_id = 1) AS s
WHERE s.is_open = 'true'
```

Παρατηρούμε ότι το φωλιασμένο υποερώτημα είναι πολύ απλούστερο από αυτό του προηγούμενου ερωτήματος και ότι δεν πραγματοποιείται κάποιο join. Τη λογική αυτή, θα πρέπει να ακολουθήσουμε γενικά στην επανεγγραφή των ερωτημάτων τύπου select.

3.3.2 Ερωτήματα τροποποίησης δεδομένων

Το γεγονός ότι μια λογική σχέση μπορεί να μην αποθηκεύεται αυτούσια σε ένα μόνο φυσικό πίνακα, σημαίνει ότι ερωτήματα τροποποίησης δεδομένων ενδέχεται να επηρεάσουν παραπάνω από έναν πίνακα.

Κατά την εισαγωγή μιας εγγραφής, το σύστημα θα πρέπει να εισάγει μια εγγραφή σε ένα βασικό πίνακα(αν η αντίστοιχη λογική σχέση είναι διαμοιραζόμενη), καθώς και τόσες εγγραφές στο Chunk Table, όσες και τα chunks από τα οποία αποτελείται η σχέση. Για παράδειγμα, έστω ότι ο tenant 1 επιθυμεί να εκτελέσει την παρακάτω εισαγωγή:

```
INSERT INTO surveys
VALUES
('Customer Satisfaction',
'Yearly customer satisfaction survey',
‘Mar 5 2014', false, 1.1,
'Dot Telecom Co.', NULL);
```

Ο πίνακας “surveys”, αν και διαμοιραζόμενος, για τον tenant 1 περιέχει, όπως είδαμε και πεδία-επεκτάσεις. Το σύστημα, αφού παράξει ένα μοναδικό αναγνωριστικό GUID για τη νέα εγγραφή, θα μετατρέψει το ερώτημα στην ακόλουθη μορφή:

```
INSERT INTO surveys
(guid, tenant_id, survey_title, description, end_date,
prealloc_field_1, prealloc_field_2)
VALUES
('e79a50a3-9bdb-11e3-80ce-639e57237a8b',
1,
'Customer Satisfaction',
'Yearly customer satisfaction survey',
'Mar 5 2014',
```

```
check_null(CAST(FALSE AS BOOLEAN)),
CAST(1.1 AS NUMERIC));

INSERT INTO chunk_table
(guid, chunk_num, table_id, tenant_id,
custom_field_1, custom_field_2,
custom_field_3, custom_field_4)
VALUES
('e79a50a3-9bdb-11e3-80ce-639e57237a8b', 1, 2, 1,
'Dot Telecom Co.',
NULL, NULL, NULL);
```

Έχουμε τελικά δύο εισαγωγές, μία στο βασικό πίνακα “surveys” και άλλη μια στο Chunk Table. Οι νέες εγγραφές θα πρέπει να έχουν τις σωστές τιμές στις στήλες μεταδεδομένων. Οι συναρτήσεις “CAST” και “CHECK_NULL” που εμφανίζονται αφορούν τον έλεγχο του τύπου των δεδομένων και περιορισμών τύπου not-null αντίστοιχα. Να σημειώσουμε ότι αν για κάποιο λόγο αποτύχει κάποια από τις παραπάνω εισαγωγές, θα πρέπει να απορριφθούν όλες. Με άλλα λόγια, πρέπει να εκτελούνται στα πλαίσια μιας δοσοληψίας.

Ερωτήματα διαγραφής εγγραφών ή τροποποίησης ήδη υπαρχουσών εγγραφών αντιμετωπίζονται με παρόμοιο τρόπο από το σύστημα. Συγκεκριμένα, θα πρέπει να βρίσκουμε το σύνολο των εγγραφών που επηρεάζονται και να διαγράφουμε ή να τροποποιούμε τις αντίστοιχες εγγραφές στο φυσικό σχήμα.

Έστω, για παράδειγμα, ότι ο tenant 1 στέλνει στο σύστημα το ακόλουθο ερώτημα τροποποίησης που αφορά τη σχέση “surveys”:

```
UPDATE surveys s
SET s.is_open = 'false'
WHERE s.end_date < 'Feb 20 2014'
```

Στο ερώτημα αυτό, ζητείται να αλλάξει η τιμή του πεδίου “is_open” για τις εγγραφές του πίνακα “surveys” που ικανοποιούν τη συνθήκη end_date < 'Feb 20 2014'. Το πεδίο “is_open” είναι πεδίο-επέκταση του tenant 1 και όπως είδαμε αποθηκεύεται σε κατάλληλο chunk στο Chunk Table. Συνεπώς, δε χρειάζεται να τροποποιήσουμε καμία εγγραφή στο βασικό πίνακα “surveys”, αλλά μόνο τις κατάλληλες εγγραφές του Chunk Table.

Για την εύρεση του συνόλου των εγγραφών που θα επηρεαστεί μπορούμε να ακολουθήσουμε αρκετούς τρόπους. Ο πιο απλός είναι σε κάθε ένα από τα ερωτήματα τροποποίησης που θα προκύψουν μετά από την επανεγγραφή να βάλουμε στο “WHERE” κομμάτι τους, ένα φωλιασμένο υποερώτημα ανάκτησης (subselect query) που θα επιλέγει τα αναγνωριστικά των εγγραφών που επηρεάζονται.

Το υποερώτημα αυτό μπορεί να προκύψει χρησιμοποιώντας το μηχανισμό επανεγγραφής ερωτημάτων ανάκτησης δεδομένων που περιγράψαμε στην 3.3.1. Συγκεκριμένα, δημιουργούμε αρχικά ένα ερώτημα select, στο οποίο ζητούμε να επιστραφεί το πεδίο guid. Στο “FROM” κομμάτι του ερωτήματος βάζουμε το όνομα του λογικού πίνακα που τροποποιεί το αρχικό ερώτημα τροποποίησης, και στο “WHERE” κομμάτι βάζουμε το αντίστοιχο κομμάτι του αρχικού ερωτήματος. Στο παράδειγμα μας, δηλαδή, θα δημιουργήσουμε το ακόλουθο select ερώτημα:

```
SELECT s.guid
FROM surveys s
WHERE s.end_date < 'Feb 20 2014'
```

Το ερώτημα αυτό θα το τροφοδοτήσουμε στο μηχανισμό επανεγγραφής και θα προκύψει το ακόλουθο ερώτημα:

```
SELECT s.guid
FROM (SELECT surveys.guid, surveys.end_date
        FROM surveys
        WHERE surveys.tenant_id = 1) AS s
WHERE s.end_date < 'Feb 20 2014')
```

Το αρχικό ερώτημα τροποποίησης θα μετατραπεί τελικά στο ακόλουθο:

```
UPDATE chunk_table
SET custom_field_1 = CAST('false' AS BOOLEAN)
WHERE chunk_num = 1 AND
        guid IN (SELECT s.guid
                FROM (SELECT surveys.guid, surveys.end_date
                        FROM surveys
                          WHERE surveys.tenant_id = 1) AS s
                WHERE s.end_date < 'Feb 20 2014'));
```

Η προσέγγιση αυτή, σε περίπτωση που ένα ερώτημα τροποποίησης αντιστοιχεί σε πολλαπλά στο φυσικό σχήμα, αναγκαζει το ΣΔΒΔ να υπολογίζει το σύνολο των αναγνωριστικών για κάθε ένα από αυτά. Θα ήταν επομένως καλύτερα να υπολογίζαμε μια φορά το σύνολο αυτό και να χρησιμοποιούσαμε τα αποτελέσματα σε κάθε ένα από τα παραγόμενα ερωτήματα τροποποίησης.

Για παράδειγμα, παρατηρούμε το παραπάνω πρόβλημα στο ακόλουθο ερώτημα διαγραφής του tenant 1:

```
DELETE FROM surveys
WHERE surveys.is_open = 'false';
```

Το ερώτημα αυτό μεταφράζεται στα ακόλουθα δυο ερωτήματα διαγραφής στο φυσικό σχήμα, ένα στο βασικό πίνακα “surveys” και ένα στο Chunk Table.

```
DELETE FROM surveys
WHERE guid IN
        (SELECT surveys.guid
         FROM (SELECT surveys.guid,
                        CAST(surveys.prealloc_field_1 AS BOOLEAN) AS is_open
                FROM surveys
```

```
            WHERE surveys.tenant_id = 1) AS surveys
    WHERE surveys.is_open = 'false');


DELETE FROM chunk_table
WHERE guid IN
        (SELECT surveys.guid
         FROM (SELECT surveys.guid,
                        CAST(surveys.prealloc_field_1 AS BOOLEAN) AS is_open
                FROM surveys
                WHERE surveys.tenant_id = 1) AS surveys
        WHERE surveys.is_open = 'false');
```

Αν υπολογίζαμε πρώτα το σύνολο των GUIDs και το χρησιμοποιούσαμε και στα δυο ερωτήματα διαγραφής, θα γλιτώναμε το διπλό υπολογισμό του. Για παράδειγμα, θα μπορούσαμε να ορίσουμε ένα προσωρινό πίνακα μέσα στη δοσοληψία που θα εκτελούσαμε τα ερωτήματα διαγραφής και στον πίνακα αυτό να αποθηκεύαμε το σύνολο των GUIDs. Ο προσωρινός αυτός πίνακας θα ήταν ορατός μόνο στα πλαίσια της συγκεκριμένης δοσοληψίας.

3.4 Έλεγχος Περιορισμών

3.4.1 Έλεγχος τύπων ιδιωτικών πεδίων

Οι τιμές που θα δίνουμε σε ένα ιδιωτικό πεδίο θα αποθηκεύονται τελικά ως varchar. Για τη μετατροπή από τον τύπο varchar στον κανονικό τύπο του πεδίου μπορούμε να χρησιμοποιήσουμε το μηχανισμό μετατροπής τύπων του ΣΔΒΔ. Στα περισσότερα τέτοια συστήματα η μετατροπή γίνεται μέσω της κλήσης μιας συνάρτησης(μέσα σε ένα ερώτημα sql). Στην Postgresql(την οποία θα χρησιμοποιήσουμε και για το σύστημα μας) η συνάρτηση αυτή είναι της μορφής CAST(expr AS datatype).

Την ίδια συνάρτηση θα πρέπει να χρησιμοποιήσουμε και κατά την ανάθεση τιμής σε ένα πεδίο. Αν και τα περισσότερα ΣΔΒΔ κάνουν αυτόματα τη μετατροπή από κάποιο τύπο στον τύπο varchar, αυτό θα μας επέτρεπε να δώσουμε τιμή διαφορετικού τύπου σε ένα πεδίο χωρίς

να δημιουργηθεί σφάλμα. Για παράδειγμα, θα μπορούσαμε να δώσουμε μια μη αριθμητική τιμή σε πεδίο αριθμητικού τύπου και να μην απορριφθεί από το σύστημα. Για τον έλεγχο, λοιπόν, της ορθότητας του τύπου θα χρησιμοποιήσουμε τη συνάρτηση "CAST" για να μετατρέψουμε την έκφραση ανάθεσης που δίνεται προς στον τύπο δεδομένων του πεδίου. Αν αποτύχει η μετατροπή, απορρίπτεται η δοσοληψία και επιστρέφεται μήνυμα σφάλματος.

3.4.2 Περιορισμοί Αναφορικής Ακεραιότητας

Το γεγονός ότι ένας βασικός πίνακας περιέχει εγγραφές πολλαπλών tenants, αλλά και το ότι ένας tenant μπορεί να ορίζει δικούς του πίνακες, καθιστά το μηχανισμό ελέγχου αναφορικής ακεραιότητας του ΣΔΒΔ ακατάλληλο στην περίπτωση μας. Για το λόγο αυτό, πρέπει να διατηρούμε στα μεταδεδομένα του συστήματος πληροφορίες για όλες τις συσχετίσεις και να πραγματοποιούμε εμείς τον έλεγχο αναφορικής ακεραιότητας. Δηλαδή, σε κάθε τροποποίηση ενός πεδίου που περιέχει ένα ξένο κλειδί προς κάποιο πίνακα, θα πρέπει να ελέγχουμε ότι η νέα τιμή αντιστοιχεί σε υπαρκτό αναγνωριστικό εγγραφής του σωστού πίνακα και του σωστού tenant.

3.4.3 Περιορισμοί Μοναδικότητας

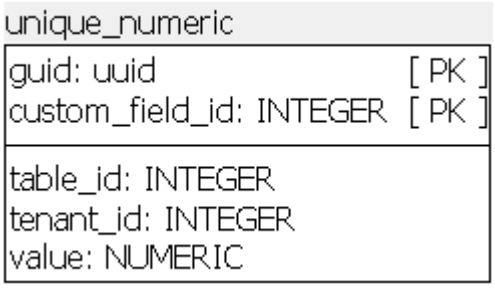
Με τους περιορισμούς μοναδικότητας απαγορεύουμε σε ένα πεδίο να αποκτά παραπάνω από μια φορά την ίδια τιμή. Σε μια βάση δεδομένων ενός tenant το ΣΔΒΔ επιτρέπει τη δημιουργία και τον έλεγχο τέτοιων περιορισμών. Στις multi-tenant βάσεις που θα υποστηρίζει το σύστημα μας αυτός ο μηχανισμός δεν αρκεί.

Αν θέλουμε ένα από τα κοινά πεδία ενός βασικού πίνακα να παίρνει μοναδικές τιμές, τότε αρκεί να ορίσουμε ένα περιορισμό μοναδικότητας στο συνδυασμό του πεδίου αυτού με το πεδίο tenant_id. Έτσι, ενώ θα απαγορεύονται διπλές τιμές για κάθε tenant, θα επιτρέπεται σε διαφορετικούς tenants να χρησιμοποιούν την ίδια τιμή.

Από την άλλη, το ότι πολλαπλά πεδία-επεκτάσεις, πιθανώς και από διαφορετικές λογικές σχέσεις, μπορούν να αποθηκεύονται στην ίδια γενική στήλη μας αναγκάζει να βρούμε κάποιον άλλο τρόπο ελέγχου της μοναδικότητας των τιμών τους. Το πρόβλημα εντείνεται και από την αποθήκευση κάθε τέτοιου πεδίου ως varchar.

Για την αντιμετώπιση του παραπάνω προβλήματος προτείνουμε την ακόλουθη διαδικασία: κάθε τιμή ενός πεδίου-επέκτασης με περιορισμό μοναδικότητας θα αποθηκεύεται σε ένα κατάλληλο πίνακα ο οποίος θα διαθέτει ένα ευρετήριο μοναδικότητας(unique index) που θα

απαγορεύει πολλαπλές τιμές για το ίδιο λογικό πεδίο. Η τιμή του πεδίου-επέκτασης θα αποθηκεύεται στον πίνακα αυτό στον κατάλληλο τύπο δεδομένων και όχι γενικά ως varchar. Συγκεκριμένα, θα έχουμε έναν τέτοιο πίνακα για κάθε υποστηριζόμενο τύπο δεδομένων από ιδιωτικά πεδία. Για παράδειγμα, έστω ο τύπος δεδομένων numeric. Ο πίνακας ελέγχου μοναδικότητας για αυτόν θα έχει την ακόλουθη μορφή:



Στον πίνακα αυτόν θα έχουμε ευρετήριο μοναδικότητας(unique index) στο συνδυασμό των πεδίων “custom_field_id” και “value”. Το “custom_field_id” περιέχει το αναγνωριστικό του ιδιωτικού πεδίου και το πεδίο “value” την τιμή του πεδίου αυτού για την εγγραφή με αναγνωριστικό guid. Κάθε φορά που εισάγουμε μια εγγραφή σε ένα λογικό πίνακα θα κοιτάμε αν έχει ιδιωτικά πεδία με περιορισμό μοναδικότητας. Για κάθε τέτοιο πεδίο θα εισάγουμε στον κατάλληλο πίνακα ελέγχου μοναδικότητας μια εγγραφή με κατάλληλες τιμές στις στήλες μεταδεδομένων και τιμή στη στήλη “value” την τιμή που παίρνει το πεδίο στην νέα εγγραφή. Τελικά, το ευρετήριο μοναδικότητας θα μας απαγορεύει να δώσουμε διπλές τιμές σε ένα τέτοιο πεδίο. Αλλαγή στην τιμή τέτοιων πεδίων(μέσω update) θα πρέπει να αλλάζουν και τις αντίστοιχες τιμές στους πίνακες μοναδικότητας. Επίσης, διαγραφή εγγραφών ενός πίνακα με μοναδικά πεδία, θα συνεπάγεται και διαγραφή από τους πίνακες μοναδικότητας των αντίστοιχων εγγραφών.

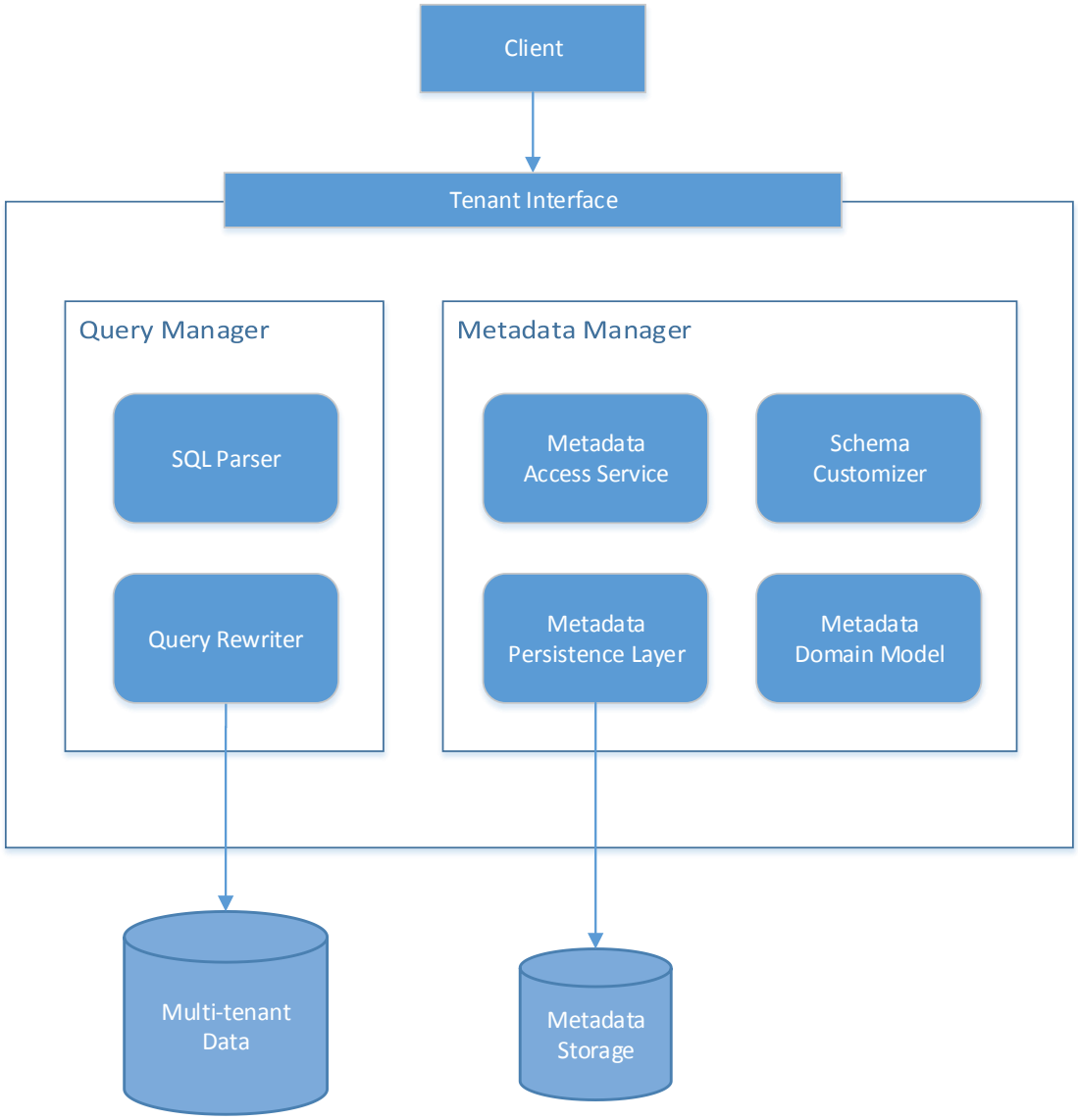
4

Περιγραφή Συστήματος

Στο προηγούμενο κεφάλαιο είδαμε ότι το σύστημα που θα αναπτύξουμε θα προσφέρει τη δυνατότητα υποστήριξης multi-tenant βάσεων δεδομένων με χρήση διαμοιραζόμενου σχήματος. Εξηγήσαμε, επίσης, την τεχνική που θα χρησιμοποιήσουμε για την αντιστοίχιση στο φυσικό σχήμα, καθώς και τις προκλήσεις που αυτή συνεπάγεται. Στο κεφάλαιο αυτό περιγράφουμε τη γενική αρχιτεκτονική του συστήματος, καθώς και τη λειτουργία των βασικών δομικών του μονάδων.

4.1 Αρχιτεκτονική

Το σύστημα πρέπει να προσφέρει μια διεπαφή που θα αποκρύπτει τη multi-tenant λογική της βάσης και θα αναλαμβάνει εκείνο την αντιστοίχιση του λογικού σχήματος κάθε tenant στο διαμοιραζόμενο φυσικό σχήμα. Για την αντιστοίχιση αυτή, θα πρέπει να διατηρεί ένα πλήθος από μεταδεδομένα. Για παράδειγμα, θα πρέπει να διατηρεί πληροφορία σχετικά με τα πεδία-επεκτάσεις ενός tenant και το πώς αυτά αντιστοιχίζονται στα γενικά πεδία(ενός βασικού πίνακα ή του Chunk Table). Τα μεταδεδομένα αυτά θα χρησιμοποιούνται κυρίως στην επανεγγραφή των ερωτημάτων που στέλνουν στο σύστημα οι tenants ως προς το φυσικό σχήμα της βάσης δεδομένων. Στην Εικ. 4-1 βλέπουμε τη γενική αρχιτεκτονική του συστήματος.



Εικόνα 4-1

Βλέπουμε ότι το σύστημα αποτελείται από δυο βασικά δομικά μέρη:

- **Metadata Manager**

  Ο Metadata Manager είναι το υποσύστημα εκείνο που διαχειρίζεται τα μεταδεδομένα.

- **Query Manager**

  Η μονάδα αυτή αναλαμβάνει την ανάλυση, την επανεγγραφή και την τελική εκτέλεση ενός ερωτήματος.

Επίσης, στο σχήμα φαίνεται ότι το σύστημα αλληλεπιδρά με δυο βάσεις δεδομένων:

- **Metadata Storage**

  Εδώ, διατηρούνται από το σύστημα τα μεταδεδομένα που απαιτούνται για την αντιστοίχιση του λογικού σχήματος ενός tenant στο φυσικό.

- **Multi-tenant data**

  Ως "Multi-tenant data" θεωρούμε τα πραγματικά δεδομένα που αποθηκεύουν οι tenants. Συνεπώς, εδώ περιέχονται οι βασικοί πίνακες της multi-tenant βάσης, το Chunk Table, αλλά και πίνακες για τον έλεγχο περιορισμών μοναδικότητας.

Οι βάσεις αυτές διαχωρίζονται στο σχήμα απλά για να γίνει φανερή η διαφορά στη σκοπιμότητα τους.

Τέλος, παρατηρούμε ότι το σύστημα μεσολαβεί ανάμεσα στο ΣΔΒΔ και σε μια εφαρμογή-πελάτη που χρησιμοποιεί μια multi-tenant βάση δεδομένων. Πρόσβαση στη βάση γίνεται μόνο μέσω της διεπαφής "**Tenant Interface**".

4.2 Περιγραφή Δομικών Μονάδων

4.2.1 Metadata Manager

Η μονάδα αυτή είναι υπεύθυνη για τη διατήρηση και την ανάκτηση των μεταδεδομένων. Κάθε λειτουργία του συστήματος απαιτεί πρόσβαση στα μεταδεδομένα και συνεπώς η δομική αυτή μονάδα χρησιμοποιείται από όλες τις υπόλοιπες.

4.2.1.1 Metadata Domain Model

Εδώ βρίσκουμε κλάσεις που αναπαριστούν τους διαφορετικούς τύπους μεταδεδομένων. Για παράδειγμα, θα πρέπει να υπάρχουν κλάσεις που να αντιπροσωπεύουν λογικούς πίνακες, πεδία κ.λ.π.

4.2.1.2 Metdadata Persistence Layer

Το επίπεδο "Metadata Persistence Layer" αναλαμβάνει τη διατήρηση και την ανάκτηση των μεταδεδομένων από τη βάση. Τα μεταδεδομένα θα διατηρούνται σε μια σχεσιακή βάση και επομένως θα πρέπει, μιας και θα χρησιμοποιήσουμε μια αντικειμενοστρεφή γλώσσα προγραμματισμού(συγκεκριμένα τη Java), να αντιστοιχίζονται σε μορφή αντικειμένου για χρήση από το σύστημα και συγκεκριμένα στις κλάσεις αντικειμένων που ορίζονται στο

“Metadata Domain Model”. Για την αντιστοίχιση αυτή ενδείκνυται η χρήση κάποιου εργαλείου αντικειμενοσχεσιακής απεικόνισης.

4.2.1.3 Metadata Access Service

Η υπομονάδα αυτή προσφέρει μια διεπαφή για πρόσβαση στα μεταδεδομένα του συστήματος από τις υπόλοιπες δομικές μονάδες. Όταν ένας tenant αλληλεπιδρά με το σύστημα, η υπομονάδα “Metadata Access Service”, καλώντας μεθόδους της υπομονάδας “Metadata Persistence Layer”, φορτώνει από τη βάση μεταδεδομένων τα απαραίτητα μεταδεδομένα για το συγκεκριμένο tenant. Για να αποφύγουμε περιττές προσβάσεις στη βάση δεδομένων, η υπομονάδα “Metadata Access Service” πρέπει να διατηρεί τα μεταδεδομένα που φορτώνει σε μια προσωρινή μνήμη(cache). Οι εγγραφές από την cache θα αντικαθίστανται όταν αυτή γεμίζει με χρήση πολιτικής LRU(least recently used). Με άλλα λόγια, θα διώχνεται η εγγραφή με τα μεταδεδομένα του tenant εκείνου που έχει να χρησιμοποιήσει το σύστημα τον περισσότερο χρόνο.

4.2.1.4 Schema Customizer

Ο Schema Customizer αναλαμβάνει την εξατομίκευση του σχήματος ενός tenant.

Η γλώσσα SQL περιλαμβάνει, πέραν από δυνατότητες ανάκτησης και ενημέρωσης δεδομένων, και δυνατότητες δημιουργίας και τροποποίησης σχημάτων και σχεσιακών πινάκων[16]. Υπενθυμίζουμε ότι οι multi-tenant βάσεις δεδομένων που θα υποστηρίζει το σύστημα θα μπορούν να έχουν έναν αριθμό από κοινούς βασικούς πίνακες και πεδία αυτών, τα οποία θα ορίζονται από το σχεδιαστή της βάσης και δε θα επιτρέπεται να τροποποιηθούν από κάποιον tenant. Εντούτοις, ένας tenant θα μπορεί να επεκτείνει ένα βασικό πίνακα, καθώς και να ορίσει μια δική του ιδιωτική σχέση. Επιπλέον, θα μπορεί να τροποποιήσει τα προσωπικά του πεδία ή πίνακες(διαγραφή, μετονομασία κ.λ.π.). Αυτές οι λειτουργίες δείξαμε ότι δε θα αντιστοιχούν σε δημιουργία ενός φυσικού πεδίου ή πίνακα στη βάση δεδομένων. Αντ’αυτού, θα δημιουργούνται ή θα τροποποιούνται εγγραφές σε πίνακες μεταδεδομένων οι οποίες θα περιγράφουν την αντιστοίχιση των λογικών αυτών πεδίων ή πινάκων στο φυσικό σχήμα. Αν και οι λειτουργίες αυτές θα μπορούσαν να εκκινούνται από κάποιον tenant μέσω ερωτημάτων SQL(πχ CREATE TABLE, ALTER TABLE κ.λ.π.), τα οποία το σύστημα θα μεταφράζει στις κατάλληλες ενέργειες στα μεταδεδομένα, θεωρούμε καλύτερη προσέγγιση την εξατομίκευση του σχήματος μέσω κλήσεων κατάλληλων μεθόδων. Έτσι, θα γίνεται άμεσα κατανοητό ποιές λειτουργίες επιτρέπονται(π.χ. με μέθοδο deleteCustomField γίνεται φανερό ότι επιτρέπεται διαγραφή μόνο πεδίου-επέκτασης) και ποιές όχι.

4.2.2 Query Manager

Ο "Query Manager" αποτελεί, όπως είδαμε, το κομμάτι εκείνο του συστήματος που είναι υπεύθυνο για τις λειτουργίες ανάκτησης και τροποποίησης των δεδομένων στη multi-tenant βάση. Λαμβάνει το ερώτημα ενός tenant και, χρησιμοποιώντας τα κατάλληλα μεταδεδομένα που θα ζητήσει από το Metadata Access Service, το τροποποιεί ώστε να αντιστοιχίζεται στο φυσικό σχήμα. Τελικά, το στέλνει προς εκτέλεση στο χρησιμοποιούμενο ΣΔΒΔ και, αφού λάβει τα αποτελέσματα του ερωτήματος, τα προωθεί στον tenant.

Το ερώτημα που στέλνει ο tenant πρέπει αρχικά να αναλυθεί συντακτικά για να προσδιοριστεί το είδος, καθώς και η δομή του. Για παράδειγμα, σε ένα ερώτημα ανάκτησης πρέπει να προσδιοριστούν οι πίνακες που αναφέρονται καθώς και ποια πεδία τους έτσι ώστε, χρησιμοποιώντας τα κατάλληλα μεταδεδομένα, να αντικατασταθούν από τα αντίστοιχα υποερωτήματα ανακατασκευής ενός λογικού πίνακα(βλέπε ενότητα 3.3). Η ανάλυση αυτή πραγματοποιείται από τον SQL Parser ο οποίος, μετά την ανάλυση, στέλνει το ερώτημα στον Query Rewriter που το μετασχηματίζει για εκτέλεση στο φυσικό σχήμα.

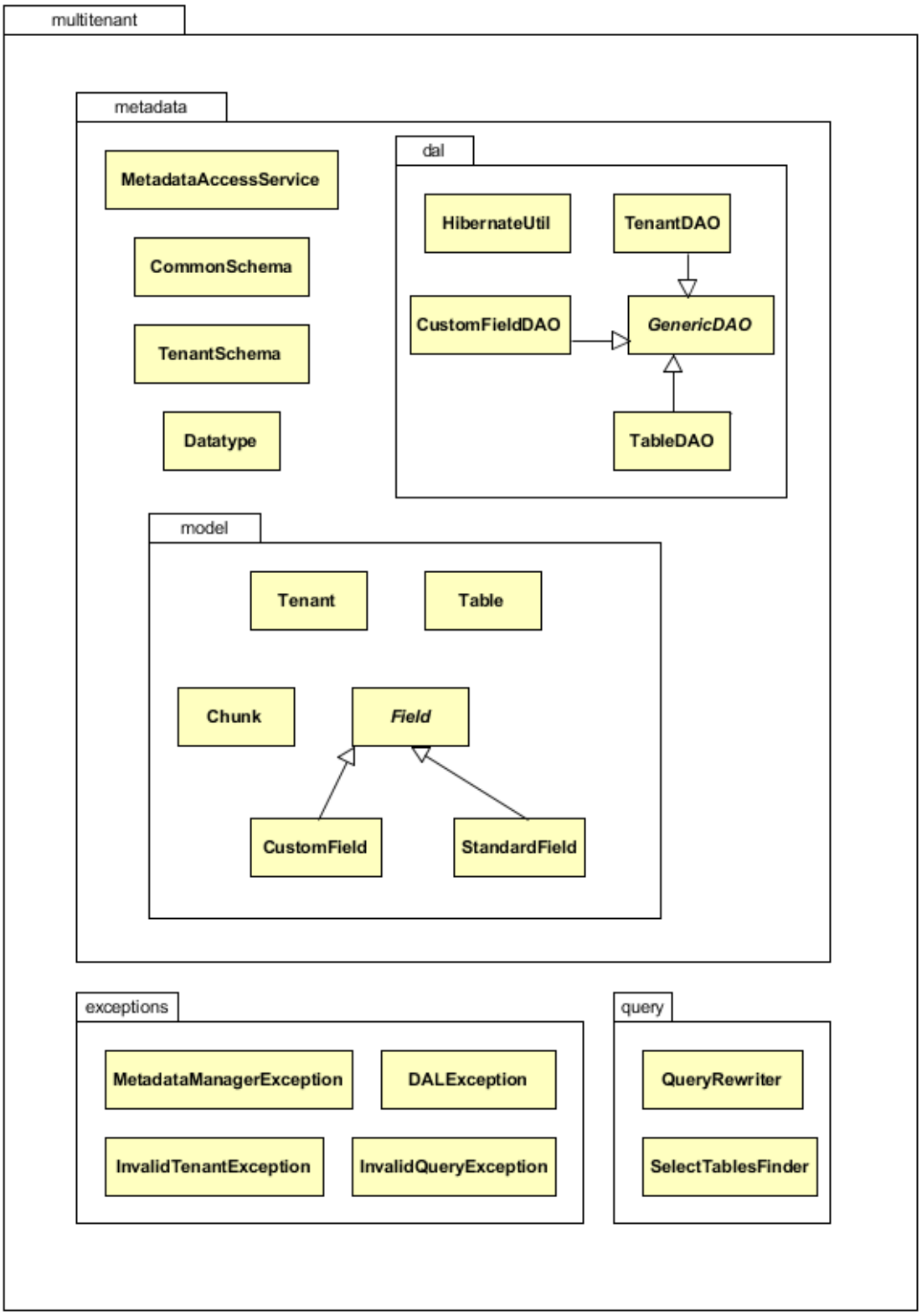
5

Σχεδίαση Συστήματος

Στο κεφάλαιο αυτό, θα ασχοληθούμε με τη σχεδίαση του συστήματος. Αρχικά, εξετάζουμε τις κλάσεις από τις οποίες θα αποτελείται το σύστημα. Στη συνέχεια παρουσιάζουμε το σχεσιακό σχήμα που θα χρησιμοποιήσουμε για την αποθήκευση των μεταδεδομένων καθώς και τη μορφή που θα έχουν οι πίνακες που θα περιέχουν τα δεδομένα της multi-tenant βάσης.

5.1 Αρχιτεκτονική

Στην εικόνα 5-1 βλέπουμε συνοπτικά την ιεραρχία πακέτων από τα οποία αποτελείται το σύστημα και τις κλάσεις που αυτά περιέχουν.


multitenant
metadata
MetadataAccessService
CommonSchema
TenantSchema
Datatype
dal
HibernateUtil
TenantDAO
CustomFieldDAO
GenericDAO
TableDAO
model
Tenant
Table
Chunk
Field
CustomField
StandardField
exceptions
MetadataManagerException
DALException
InvalidTenantException
InvalidQueryException
query
QueryRewriter
SelectTablesFinder

Εικόνα 5-1

5.2 Περιγραφή Κλάσεων

Στην ενότητα αυτή περιγράφουμε, με διαγράμματα κλάσεων της UML, τις κλάσεις του συστήματος. Οι κλάσεις παρουσιάζονται οργανωμένες σε υποενότητες, όπου κάθε υποενότητα αντιστοιχεί σε ένα πακέτο κλάσεων του συστήματος.

5.2.1 multitenant.metadata.model

Σε αυτό το πακέτο κλάσεων βρίσκουμε κλάσεις που αντιστοιχούν στις βασικές οντότητες μεταδεδομένων που κρατάμε στο σύστημα. Λόγω μεγέθους, το αντίστοιχο διάγραμμα κλάσεων μοιράζεται στις εικόνες 5-2 και 5-3 και γι'αυτό το λόγο παραλείπονται οι συσχετίσεις και οι εξαρτήσεις ανάμεσα σε αυτές τις κλάσεις.

5.2.1.1 Tenant

Η κλάση "Tenant" αναπαριστά έναν tenant και προσφέρει μεθόδους για ανάκτηση και για ανάθεση των πεδίων που χαρακτηρίζουν έναν tenant. Τέτοια πεδία είναι το αναγνωριστικό ενός tenant, το συνθηματικό και ο κώδικός πρόσβασης στο σύστημα, το όνομα της αντίστοιχης εταιρίας ή οργανισμού κ.λ.π.

5.2.1.2 Table

Η κλάση "Table" αναπαριστά ένα λογικό πίνακα της βάσης δεδομένων. Ένα Table μπορεί να αντιστοιχεί είτε σε ένα βασικό πίνακα, είτε σε ένα ιδιωτικό πίνακα ενός tenant. Σε κάθε περίπτωση, ένα στιγμιότυπο της κλάσης αυτής αφορά έναν tenant. Έτσι, ακόμα και αν αντιστοιχεί σε ένα βασικό πίνακα, ένα Table θα περιέχει πληροφορία για τα πεδία-επεκτάσεις του συγκεκριμένου tenant σε αυτόν. Η κλάση αυτή περιλαμβάνει μεθόδους για επιστροφή των πεδίων του αντίστοιχου πίνακα, καθώς και μεθόδους για προσθήκη, διαγραφή και μετονομασία ενός πεδίου.

5.2.1.3 Field

Η κλάση "Field" είναι μια αφηρημένη κλάση, δηλαδή δε μπορούν να δημιουργηθούν στιγμιότυπα της. Χρησιμοποιείται για να δηλώσουμε ότι οι κλάσεις που θα την κληρονομούν θα πρέπει να μοιράζονται κάποια πεδία και μεθόδους. Ένα πεδίο πρέπει να έχει ένα "λογικό" όνομα το οποίο θα επιστρέφεται μέσω της *getFieldName(),* καθώς και ένα "φυσικό" όνομα το οποίο επιστρέφεται μέσω της *getPhysicalName().* Το φυσικό όνομα αντιστοιχεί στο όνομα της πραγματικής στήλης στη βάση δεδομένων στην οποία θα αποθηκεύεται το πεδίο αυτό.

Με την “getTable” παίρνουμε τον πίνακα που περιέχει το πεδίο αυτό, ενώ με την “getTargetTable” παίρνουμε τον πίνακα στον οποίο «δείχνει» το πεδίο, αν αποτελεί ένα ξένο κλειδί. Τέλος, τα πεδία ενός πίνακα έχουν μια τιμή που ορίζει τη σειρά με την οποία θα εμφανίζονται στον πίνακα. Η τιμή αυτή επιστρέφεται μέσω της “getFieldOrder”. Τα αντικείμενα της Field, θα μπορούν να συγκριθούν μεταξύ τους ως προς την τιμή αυτή.

5.2.1.4 StandardField

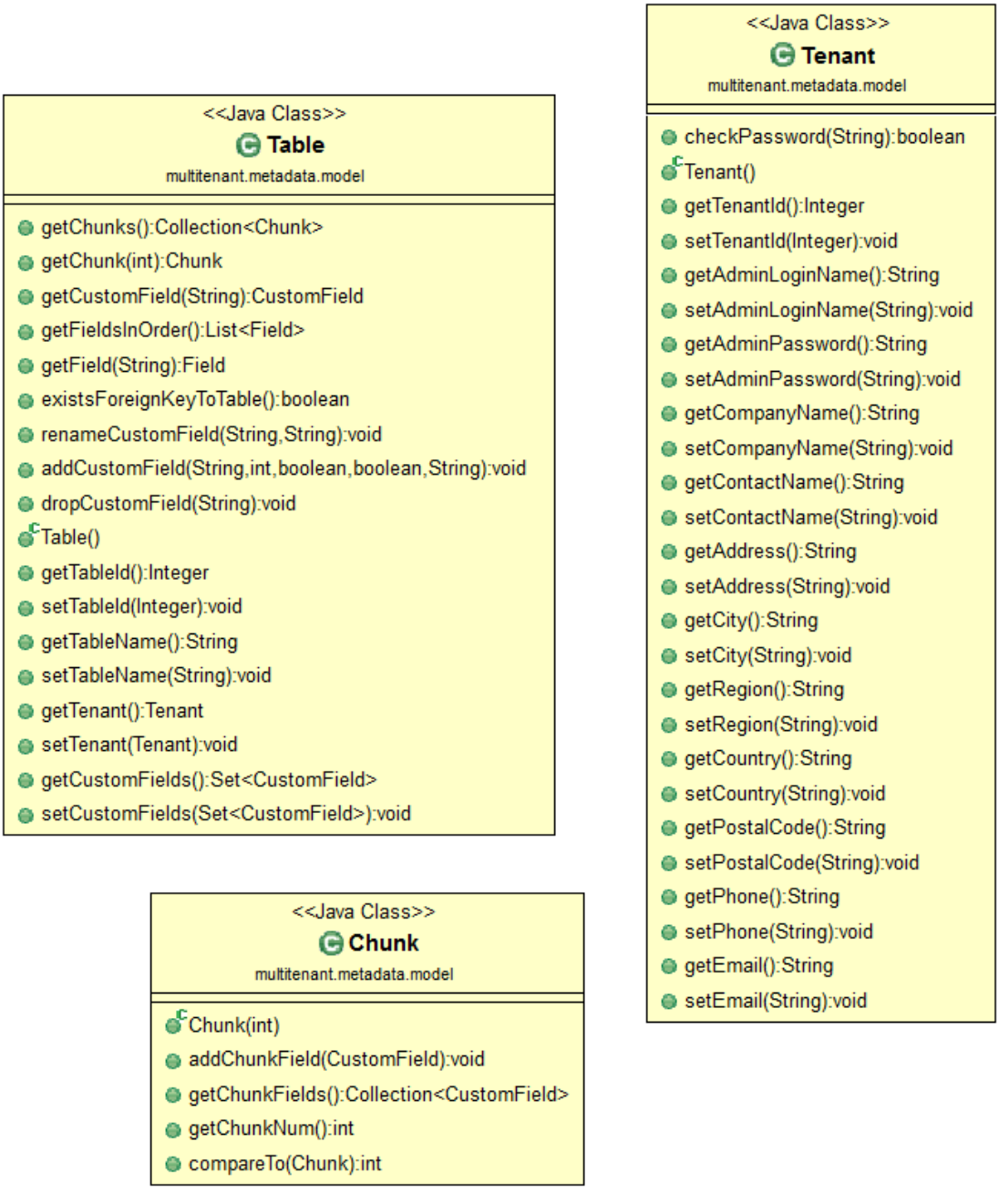
Ένα “StandardField”, που κληρονομεί την κλάση “Field”, αναπαριστά ένα βασικό πεδίο ενός βασικού πίνακα. Δηλαδή, και το λογικό αλλά και το φυσικό όνομα του πεδίου αυτού ταυτίζονται.

5.2.1.5 CustomField

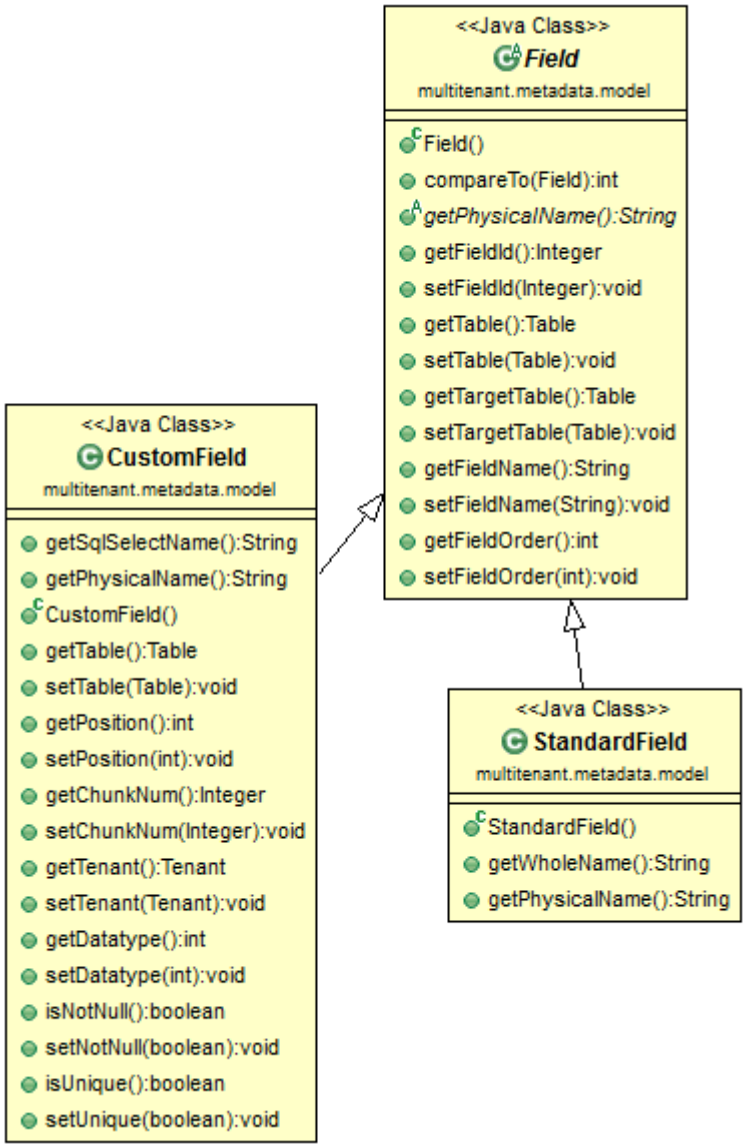
Όπως και η κλάση “StandardField”, η κλάση “CustomField” κληρονομεί την αφηρημένη κλάση “Field”. Ένα CustomField αναπαριστά ένα πεδίο-επέκταση ενός tenant που μπορεί να αντιστοιχεί είτε σε ένα γενικό πεδίο ενός βασικού πίνακα, είτε σε κάποια στήλη ενός chunk, ανάλογα με το αν η μέθοδος *getChunkNum()* επιστρέφει null ή τον αριθμό του αντίστοιχου chunk. Σε κάθε περίπτωση, η μέθοδος *getPosition()* προσδιορίζει τον αύξοντα αριθμό του αντίστοιχου γενικού πεδίου του βασικού πίνακα ή της στήλης του Chunk Table. Μιας και ο έλεγχος τύπων, αλλά και περιορισμών μοναδικότητας ή τύπου not-null, θα πραγματοποιείται από το σύστημα για τα ιδιωτικά πεδία, η κλάση CustomField περιλαμβάνει σχετική πληροφορία(getDatatype() ,isNotNull(), isUnique())

5.2.1.6 Chunk

Ένα Table μπορεί να έχει έναν αριθμό από στιγμιότυπα της κλάσης “Chunk”. Ένα τέτοιο στιγμιότυπο αναπαριστά ένα chunk μιας λογικής σχέσης(είτε βασικής είτε ιδιωτικής ενός tenant) και περιέχει ένα υποσύνολο από τα ιδιωτικά πεδία ενός λογικού πίνακα.


<<Java Class>>
Table
multitenant.metadata.model
getChunks():Collection<Chunk>
getChunk(int):Chunk
getCustomField(String):CustomField
getFieldsInOrder():List<Field>
getField(String):Field
existsForeignKeyToTable():boolean
renameCustomField(String,String):void
addCustomField(String,int,boolean,boolean,String):void
dropCustomField(String):void
Table()
getTableId():Integer
setTableId(Integer):void
getTableName():String
setTableName(String):void
getTenant():Tenant
setTenant(Tenant):void
getCustomFields():Set<CustomField>
setCustomFields(Set<CustomField>):void
<<Java Class>>
Tenant
multitenant.metadata.model
checkPassword(String):boolean
Tenant()
getTenantId():Integer
setTenantId(Integer):void
getAdminLoginName():String
setAdminLoginName(String):void
getAdminPassword():String
setAdminPassword(String):void
getCompanyName():String
setCompanyName(String):void
getContactName():String
setContactName(String):void
getAddress():String
setAddress(String):void
getCity():String
setCity(String):void
getRegion():String
setRegion(String):void
getCountry():String
setCountry(String):void
getPostalCode():String
setPostalCode(String):void
getPhone():String
setPhone(String):void
getEmail():String
setEmail(String):void
<<Java Class>>
Chunk
multitenant.metadata.model
Chunk(int)
addChunkField(CustomField):void
getChunkFields():Collection<CustomField>
getChunkNum():int
compareTo(Chunk):int

Εικόνα 5-2



Εικόνα 5-3

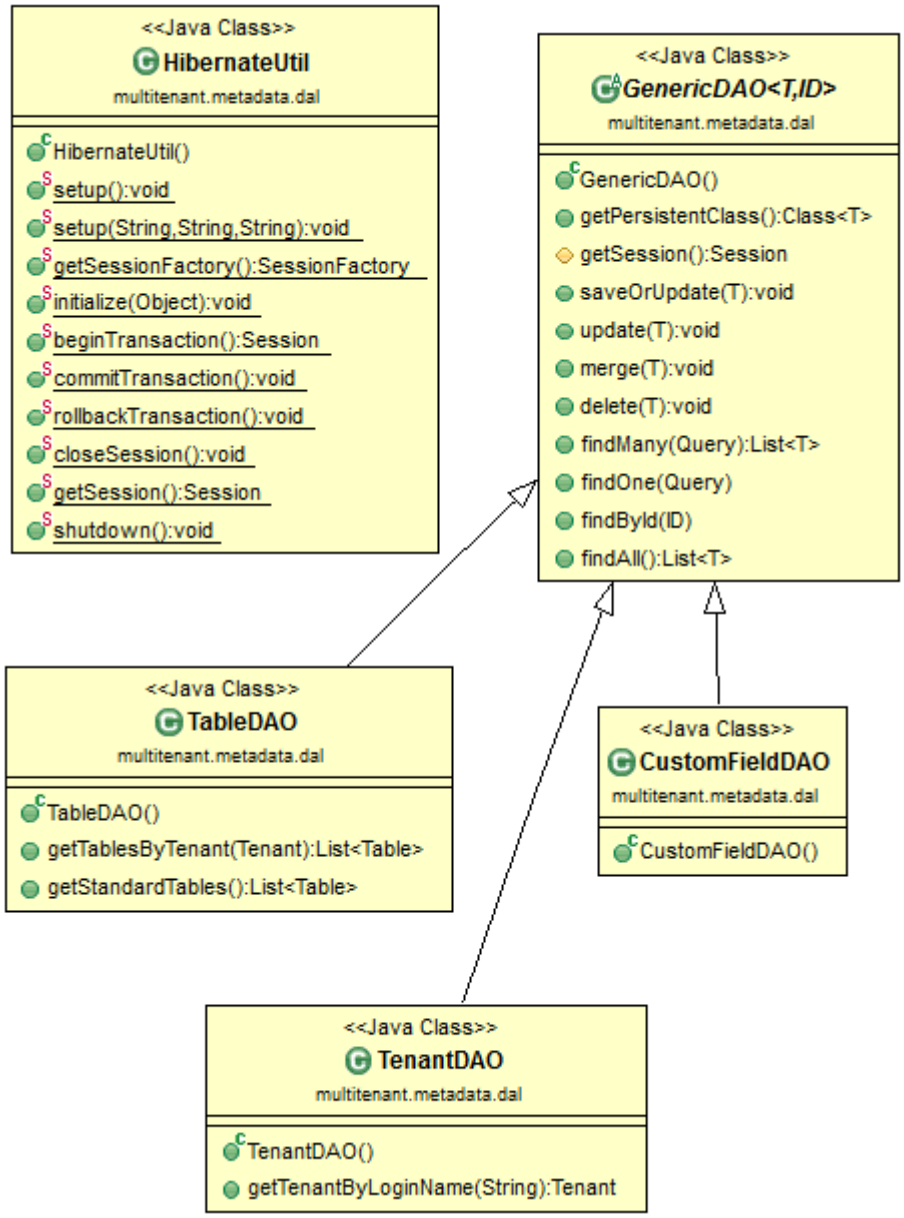
5.2.2 multitenant.metadata.dal

Σε αυτό το πακέτο κλάσεων, υπάρχουν κλάσεις που πραγματοποιούν τη διατήρηση και την ανάκτηση από τη βάση των μεταδεδομένων που περιγράψαμε προηγουμένως. Αναφέραμε και πριν ότι για τη διατήρηση των μεταδεδομένων στη βάση ενδείκνυται η χρήση λογισμικού αντικειμενοσχεσιακής απεικόνισης. Ως αντικειμενοσχεσιακή απεικόνιση (Object-Relational Mapping ή ORM) ορίζεται ένας αυτοματοποιημένος τρόπος απεικόνισης του μοντέλου αντικειμένων(object model) μιας αντικειμενοστραφούς εφαρμογής στη σχεσιακή βάση στην

οποία αποθηκεύονται τελικά τα δεδομένα. Η απεικόνιση αυτή γίνεται χρησιμοποιώντας μετά-δεδομένα (metadata) για την περιγραφή του τρόπου της διασύνδεσης. Ουσιαστικά, τα ORM συστήματα αναλαμβάνουν το επίπεδο DAL(data access layer) της εφαρμογής και επιτρέπουν την εύκολη αποθήκευση ολόκληρων γράφων από συνδεδεμένα αντικείμενα σε μια σχεσιακή βάση δεδομένων. Το ORM διαχειρίζεται μόνο του αυτή τη διαδικασία, διαμορφώνοντας αυτόματα τα κατάλληλα SQL ερωτήματα προς τη βάση. Το βασικό πλεονεκτημα από τη χρήση ενός τέτοιου εργαλείου είναι η μείωση του χρόνου και του κόστους ανάπτυξης και συντήρησης του λογισμικού, καθώς διαφορετικά ο προγραμματιστής θα έπρεπε να αναπτύξει εξ'αρχής ένα επίπεδο το οποίο θα δημιουργεί ερωτήματα sql και θα αντιστοιχεί τα δεδομένα της βάσης σε αντικείμενα της εφαρμογής. Το εργαλείο ORM το οποίο επιλέξαμε για την ανάπτυξη του συστήματος είναι το Hibernate.

Παρά το γεγονός ότι η χρήση εργαλείων αντικειμενοσχεσιακής απεικόνισης μας επιτρέπει να αποφύγουμε την ανάπτυξη ενός Data Access Layer(αφού αρκεί να ορίσουμε με κατάλληλο τρόπο τα μεταδεδομένα της απεικόνισης και να χρησιμοποιήσουμε τις μεθόδους του εργαλείου), εντούτοις, είναι καλή πρακτική η δημιουργία κλάσεων DAO(Database Access Objects). Οι κλάσεις αυτές διαχωρίζουν τη λογική του εργαλείου από το υπόλοιπο πρόγραμμα και το κάνουν πιο ευέλικτο και καλύτερα δομημένο. Έτσι, λοιπόν, δημιουργούμε μια γενική αφηρημένη κλάση DAO, τη GenericDAO που θα περιλαμβάνει γενικές μεθόδους για φόρτωση και αποθήκευση των οντοτήτων που θα αντιστοιχίζουμε στη βάση. Για κάθε μια από τις οντότητες αυτές θα δημιουργήσουμε μια κλάση DAO που θα κληρονομεί τη GenericDAO και θα προσθέτει και επιπλέον μεθόδους για πιο εξειδικευμένες λειτουργίες φόρτωσης και διατήρησης μεταδεδομένων. Για παράδειγμα, βλέπουμε τη μέθοδο getTenantByLoginName(string) της κλάσης TenantDAO που φορτώνει τον tenant που αντιστοιχεί στο συγκεκριμένο συνθηματικό πρόσβασης. Επίσης, η κλάση TableDAO προσφέρει μεθόδους για ανάκτηση των Table ενός tenant ή γενικά των Table που αντιστοιχούν στους βασικούς πίνακες της multi-tenant βάσης.

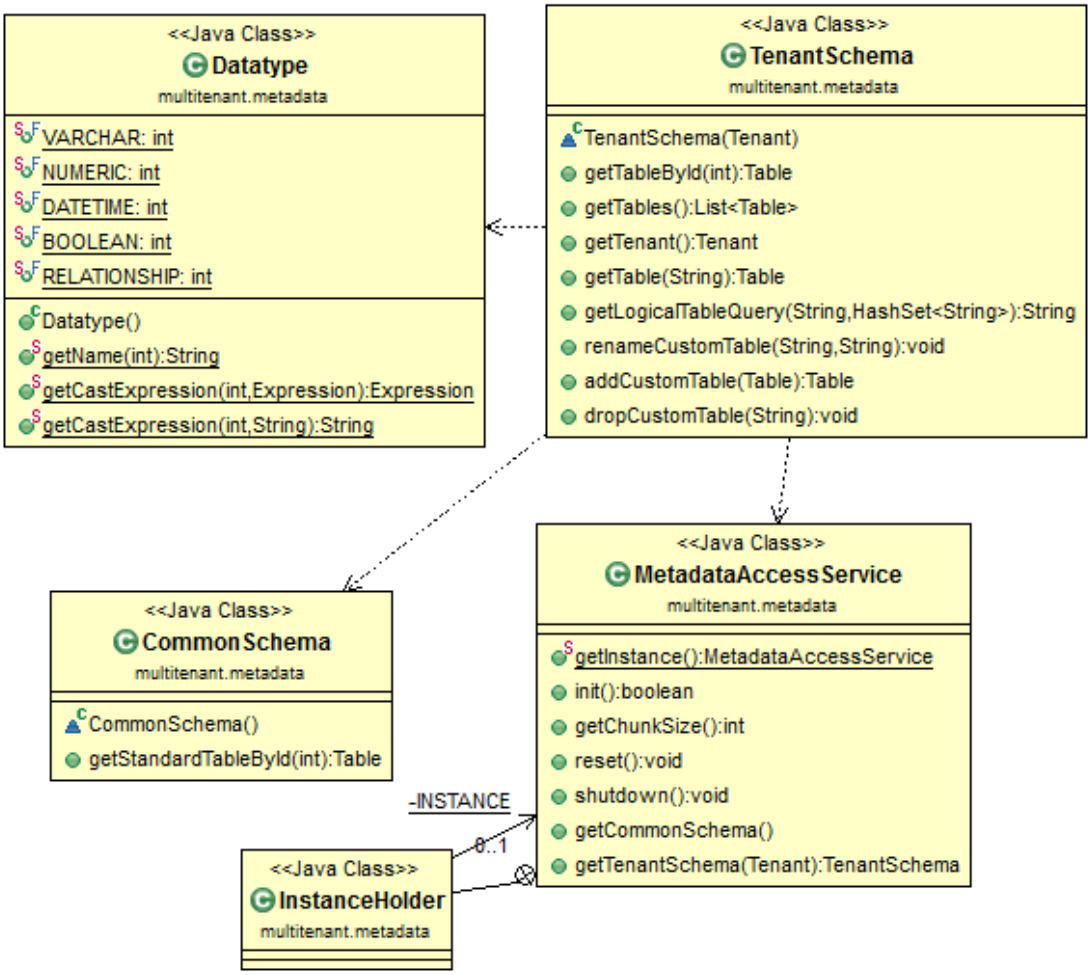
Τέλος, η κλάση HibernateUtil αφορά τη διαχείριση κάποιων παραμέτρων του ORM εργαλείου και την οργάνωση των αλληλεπιδράσεων του με τη βάση δεδομένων μέσω δοσοληψιών.



Εικόνα 5-4

5.2.3 multitenant.metadata

Εδώ βρίσκονται κλάσεις που είτε ομαδοποιούν κατάλληλα τις βασικές οντότητες μεταδεδομένων που περιγράψαμε στην παράγραφο 5.2.1, είτε προσφέρουν μια διεπαφή για πρόσβαση στα μεταδεδομένα.



Εικόνα 5-5

5.2.3.1 CommonSchema

Η κλάση "CommonSchema" αφορά το κοινό σχήμα των tenants. Περιέχει, με άλλα λόγια, μια λίστα με στιγμιότυπα της κλάσης Table, που αντιπροσωπεύουν τους βασικούς διαμοιραζόμενους πίνακες της βάσης. Κάθε τέτοιο Table θα περιέχει τη λίστα με τα αντίστοιχα στιγμιότυπα της κλάσης StandardField. Η κλάση CommonSchema στόχο έχει το να μη χρειάζεται να φορτώνουμε και να διατηρούμε στη μνήμη για κάθε tenant την πληροφορία για τα κοινά πεδία.

5.2.3.2 TenantSchema

Η κλάση αυτή ομαδοποιεί τα μεταδεδομένα που αφορούν το λογικό σχήμα ενός tenant. Δημιουργώντας ένα στιγμιότυπο της κλάσης αυτής, φορτώνουμε από τη βάση τα αντίστοιχα

μεταδεδομένα χρησιμοποιώντας μεθόδους του πακέτου multitenant.metadata.dao. Η κλάση αυτή είναι πολύ σημαντική καθώς χρησιμοποιείται και για την επανεγγραφή του ερωτήματος ενός tenant και για την εξατομίκευση του σχήματος του. Ακολουθεί μια σύντομη περιγραφή των πιο σημαντικών μεθόδων της κλάσης αυτής:

- *getTableById(tableId)*

  Η μέθοδος αυτή επιστρέφει το Table που αντιστοιχεί στο συγκεκριμένο αναγνωριστικό tableId. Το Table αυτό μπορεί να αντιπροσωπεύει είτε ένα βασικό πίνακα είτε έναν ιδιωτικό πίνακα του συγκεκριμένου tenant.

- *getTables()*

  Με τη μέθοδο αυτή επιστρέφεται μια λίστα με όλους τους πίνακες (βασικούς και ιδιωτικούς) που έχει στο σχήμα του ο tenant.

- *getTable(tableName)*

  Δίνοντας το όνομα ενός πίνακα, παίρνουμε το αντίστοιχο Table. Η μέθοδος αυτή είναι ιδιαιτέρως χρήσιμη για την επανεγγραφή ενός ερωτήματος, καθώς το Table που επιστρέφει περιέχει όλη την πληροφορία που αφορά τον πίνακα αυτόν και που είναι απαραίτητη κατά την επανεγγραφή.

- *getLogicalTableQuery*

  Τη χρησιμοποιούμε για να πάρουμε ένα ερώτημα select που ανακατασκευάζει έναν λογικό πίνακα, όπως είδαμε στην 3.3. Επίσης, για να αποφύγουμε τα περιττά joins, δίνουμε στη μέθοδο αυτή ένα σύνολο με όλα τα πεδία του πίνακα που χρειαζόμαστε στο τελικό ερώτημα.

Οι τρεις τελευταίες μέθοδοι που θα εξετάσουμε αφορούν την εξατομίκευση του σχήματος ενός tenant:

- *renameCustomTable(oldTableName, newTableName)*

  Η μέθοδος αυτή μετονομάζει έναν ιδιωτικό πίνακα. Σημειώνουμε ότι δε θα υπάρχει δυνατότητα για μετονομασία, και γενικά για τροποποίηση, ενός βασικού πίνακα.

- *addCustomTable(table)*

  Χρησιμοποιώντας τη μέθοδο αυτή προσθέτουμε στο λογικό σχήμα ενός tenant ένα νέο ιδιωτικό πίνακα που περιγράφεται από το αντικείμενο table.

- *dropCustomTable(tableName)*

  Η μέθοδος αυτή διαγράφει τον ιδιωτικό πίνακα με το όνομα tableName του tenant.

5.2.3.3 MetadataAccessService

Η κλάση αυτή αποτελεί τη διεπαφή πρόσβασης στα μεταδεδομένα του συστήματος. Επίσης, η κλάση αυτή ελέγχει(αρχικοποιεί κ.λ.π.) και όλα τα απαραίτητα υποσυστήματα που απαιτούνται για τη διατήρηση των μεταδεδομένων. Μπορεί να υπάρχει μόνο ένα στιγμιότυπο της κλάσης αυτής, με άλλα λόγια ακολουθεί το σχεδιαστικό μοτίβο singleton. Με τη μέθοδο *getCommonSchema()* επιστρέφεται ένα στιγμιότυπο της κλάσης "CommonSchema". Το στιγμιότυπο αυτό θα δημιουργείται κατά την εκκίνηση του συστήματος, όπου και θα φορτώνεται από τη βάση πληροφορία για τους κοινούς πίνακες. Μέσω της κλάσης αυτής θα προσφέρεται και πρόσβαση στα μεταδεδομένα που αφορούν το λογικό σχήμα ενός tenant. Συγκεκριμένα, με τη μέθοδο *getTenantSchema(tenant)* θα επιστρέφεται ένα αντικείμενο της κλάσης TenantSchema για τον συγκεκριμένο tenant. Το αντικείμενο αυτό θα αναζητείται πρώτα σε μια προσωρινή μνήμη(cache), και αν δε βρεθεί εκεί θα δημιουργείται, φορτώνοντας παράλληλα και τα απαραίτητα μεταδεδομένα από τη βάση. Η cache θα χρησιμοποιείται για να αποφύγουμε πολλαπλές προσβάσεις στη βάση δεδομένων για ανάκτηση μεταδεδομένων σχετικών με έναν tenant. Τέλος, η MetadataAccessService προσφέρει και μεθόδους για αρχικοποίηση, επανεκκίνηση καθώς και τερματισμό της υπηρεσίας.

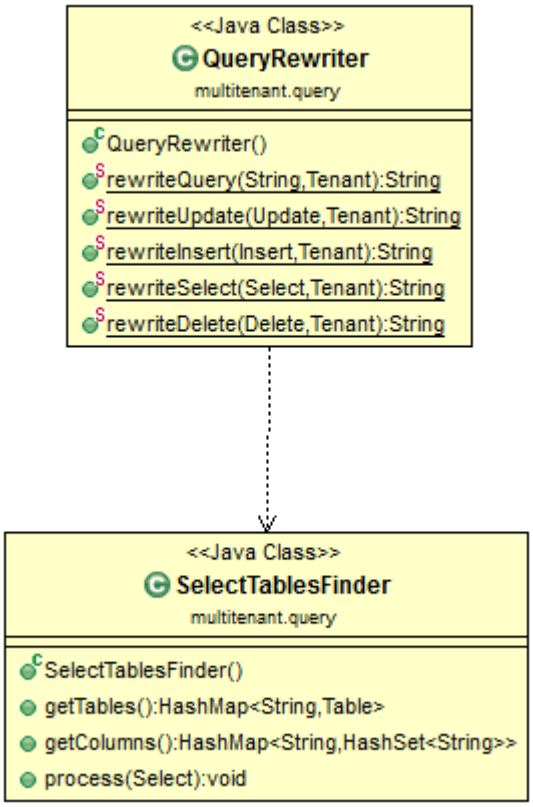
5.2.3.4 Datatype

Ένα ιδιωτικό πεδίο είδαμε ότι θα αποθηκεύεται τελικά στη βάση ως αλφαριθμητικό. Εντούτοις, μπορεί να ορίζεται και ως άλλου τύπου δεδομένων και απλά να αποθηκεύεται ως αλφαριθμητικό μετά από μετατροπή. Κάθε υποστηριζόμενος τύπος δεδομένων για ιδιωτικά πεδία θα αντιστοιχίζεται σε έναν ακέραιο αριθμό. Βλέπουμε στο διάγραμμα ότι υπάρχουν σταθερές που αντιστοιχούν στους ακόλουθους πέντε τύπους δεδομένων: varchar, numeric, datetime, boolean και relationship. Για κάθε έναν από αυτούς του τύπους θα πρέπει να υπάρχει μια αντιστοίχιση σε έναν τύπο δεδομένων του ΣΔΒΔ που θα χρησιμοποιηθεί. Δεν είναι απαραίτητο το όνομα του τύπου όπως θα χρησιμοποιείται στα πλαίσια του συστήματος(και τελικά και από κάθε tenant) να ταυτίζεται με το όνομα του αντίστοιχου τύπου του ΣΔΒΔ. Για παράδειγμα ο τύπος δεδομένων relationship στην πραγματικότητα θα πρέπει να αντιστοιχεί σε έναν τύπο δεδομένων που αναπαριστά GUIDs. Ο τύπος δεδομένων αυτός χρησιμοποιείται για πεδία που θα περιέχουν ένα ξένο κλειδί προς κάποιο άλλο πίνακα. Επειδή, στο σύστημα μας τα ξένα κλειδιά θα είναι τύπου GUID, χρησιμοποιούμε το όνομα relationship για τον τύπο αυτό για να γίνεται φανερός ο σκοπός του.

Η μέθοδος *getName(int)* επιστρέφει το όνομα του τύπου δεδομένων που αντιστοιχεί στον ακέραιο int, ενώ η μέθοδος *getCastExpression(int, expr)* επιστρέφει μια έκφραση(σε σύνταξη της sql) που μετατρέπει την έκφραση expr στον τύπο δεδομένων που αντιστοιχεί στον

ακέραιο int. Η έκφραση αυτή χρησιμοποιείται κατά την επανεγγραφή ενός ερωτήματος για τον έλεγχο και τη μετατροπή του τύπου των ιδιωτικών πεδίων(βλέπε και παρ. 3.4.1)..

5.2.4 multitenant.query



Εικόνα 5-6

5.2.4.1 QueryRewriter

Η κλάση αυτή είναι υπεύθυνη για την επανεγγραφή των ερωτημάτων. Η μέθοδος *rewriteQuery(query, tenant)* παίρνει ένα ερώτημα και χρησιμοποιώντας ένα συντακτικό αναλυτή για τη γλώσσα Sql το μετατρέπει σε μορφή που να μπορεί να το επεξεργαστεί(λεπτομέρειες θα εξετάσουμε στο Κεφ. 6). Αφού ολοκληρώσει την ανάλυση και προσδιοριστεί το είδος του ερωτήματος(select, insert, delete ή update), καλεί μια από τις υπόλοιπες μεθόδους(π.χ. rewriteSelect) για την πραγματοποίηση της επανεγγραφής. Οι

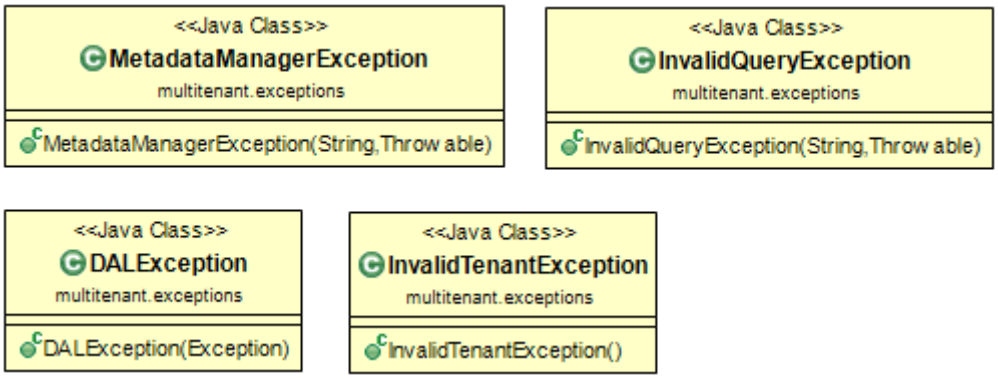
μέθοδοι αυτές χρησιμοποιώντας τα μεταδεδομένα για το σχήμα του tenant(τα οποία λαμβάνουν μέσω της MetadataAccessService) δημιουργούν και επιστρέφουν το ερώτημα που θα εκτελεστεί τελικά από τη βάση.

5.2.4.2 SelectTablesFinder

Με την κλάση αυτή, παίρνουμε ένα ερώτημα ανάκτησης(select) και βρίσκουμε τους πίνακες αλλά και τα πεδία αυτών που αναφέρονται στο ερώτημα. Η ανάλυση αυτή είναι απαραίτητη για την επανεγγραφή ενός ερωτήματος select(βλέπε και Παρ. 3.3), και γι'αυτό η κλάση αυτή χρησιμοποιείται από τη μέθοδο "rewriteSelect(select)" της κλάσης QueryRewriter. Η μέθοδος *"process(select)"* που πραγματοποιεί την παραπάνω ανάλυση δέχεται το ερώτημα σε κατάλληλη μορφή αντικειμένου, δηλαδή αφού έχει προηγηθεί συντακτική ανάλυση του. Τελικά, με τις μεθόδους "getTables()" και "getColumns()" λαμβάνουμε τα ονόματα των πίνακων και των πεδίων που αναφέρονται στο ερώτημα.

5.2.5 multitenant.exceptions

Στο πακέτο multitenant.exceptions, υπάρχουν τέσσερις κλάσεις που αντιπροσωπεύουν ειδικές περιπτώσεις εξαιρέσεων.



Εικόνα 5-7

- MetadataManagerException: αφορά γενικά σε σφάλμα του συστήματος διαχείρισης μεταδεδομένων και περιέχει λεπτομέρειες για το είδος του σφάλματος.
- InvalidQueryException: δημιουργείται σε περιπτώσεις που ένα λογικό ερώτημα που στέλνεται στο σύστημα δεν είναι έγκυρο. Τέτοιες περιπτώσεις περιλαμβάνουν

ερωτήματα που δεν ακολουθούν τους κανόνες σύνταξης της SQL, που περιέχουν αναφορές σε μη ορισμένα πεδία και πίνακες, που παραβιάζουν κάποιο περιορισμό(π.χ. not null constraint) κ.λ.π.

- DALException: αφορά σε σφάλμα του επιπέδου αντικειμενοσχεσιακής απεικόνισης των μεταδεδομένων. Με άλλα λόγια, περιέχει μια από τις εξαιρέσεις που μπορούν να ριχθούν από το εργαλείο ORM που θα χρησιμοποιήσουμε, δηλαδή το Hibernate. Η "DALException" χρησιμοποιείται μόνο εσωτερικά από το σύστημα και τελικά οδηγεί στην ρίψη μιας εξαίρεσης τύπου "MetadataManagerException".
- InvalidTenantException: προκύπτει μετά από σφάμα ταυτοποίησης των στοιχείων σύνδεσης ενός tenant με το σύστημα.

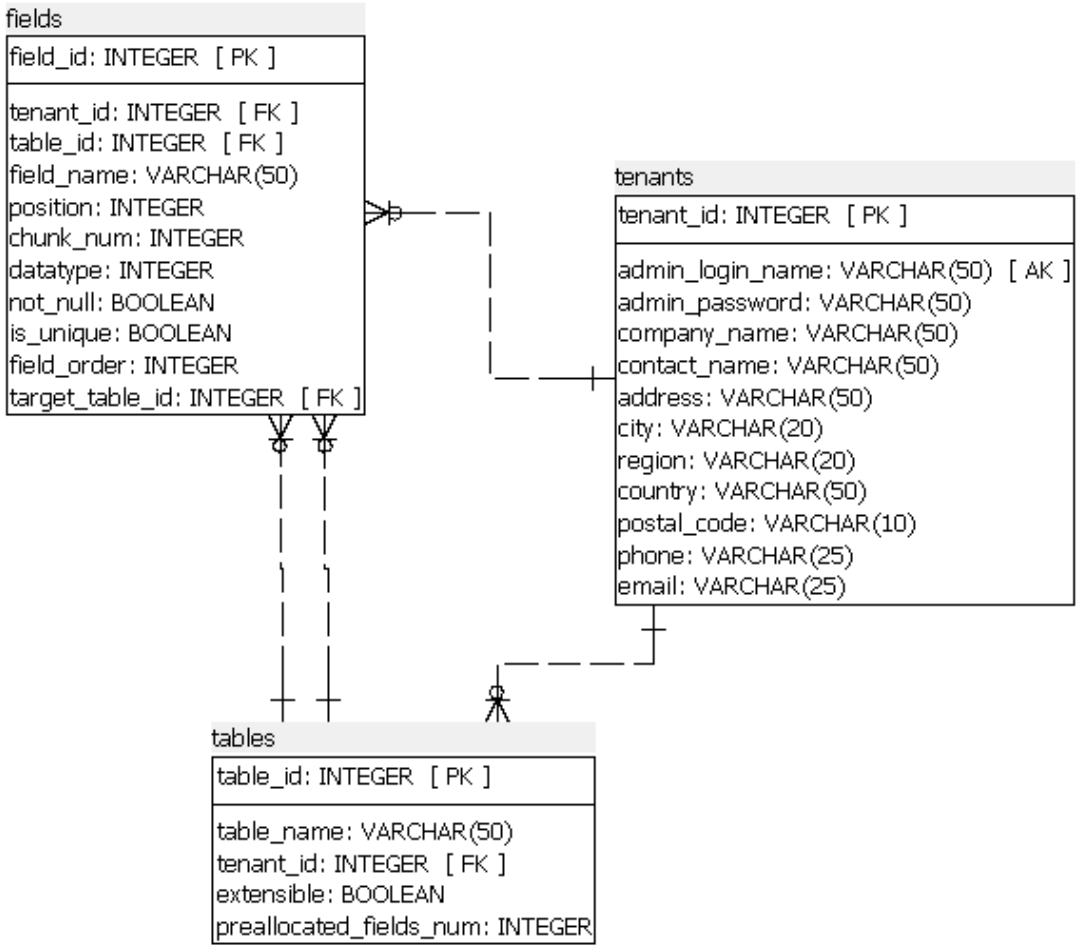
5.3 Βάση Δεδομένων

Στην ενότητα αυτή παρουσιάζουμε το σχεσιακό σχήμα των μεταδεδομένων που θα χρησιμοποιεί το σύστημα, καθώς και την απαιτούμενη μορφή των πινάκων στη multi-tenant βάση. Είδαμε ότι το σύστημα θα διατηρεί και θα φορτώνει τα μεταδεδομένα από τη βάση μέσω του εργαλείου αντικειμενοσχεσιακής απεικόνισης Hibernate. Αντ'αυτού, τα πραγματικά δεδομένα των tenants θα ανακτώνται μέσω ερωτημάτων sql τα οποία θα επανεγγράφονται από το σύστημα και θα εκτελούνται στη βάση. Επειδή, όπως θα δούμε στο Κεφ. 6, κάποιες λειτουργίες εξατομίκευσης απαιτούν στα πλαίσια μιας δοσοληψίας πρόσβαση και στα μεταδεδομένα αλλά και στα δεδομένα των tenants, είναι απλούστερο και αποδοτικότερο τα δυο σχεσιακά σχήματα να μοιράζονται την ίδια βάση δεδομένων.

Τα σχεσιακά σχήματα που θα παρουσιάσουμε σχεδιάσθηκαν για το σύστημα PostgreSQL, αν και εύκολα μπορούν να προσαρμοσθούν και για χρήση από άλλα σχεσιακά ΣΔΒΔ. Επίσης, μιας και η PostgreSQL υποστηρίζει τη δημιουργία πολλαπλών σχημάτων στην ίδια βάση[19], επιλέξαμε τα μεταδεδομένα να αποθηκεύονται σε διαφορετικό σχήμα. Κάθε τέτοιο σχήμα αποτελεί πρακτικά ένα χώρο ονομάτων(namespace). Έτσι, δε θα απαγορεύεται η δημιουργία στη multi-tenant βάση βασικού πίνακα με όνομα ίδιο με κάποιου πίνακα μεταδεδομένων.

5.3.1 Σχήμα μεταδεδομένων

Στην εικόνα 5-8 βλέπουμε το σχεσιακό διάγραμμα του σχήματος μεταδεδομένων.



Εικόνα 5-8

Πίνακας “tables”

Στον πίνακα αυτό διατηρούμε την πληροφορία σχετικά με τους πίνακες της βάσης. Διατηρούμε εγγραφές τόσο για τους βασικούς πίνακες, όσο και για ιδιωτικούς πίνακες των tenants. Ακολουθεί η περιγραφή των πεδίων της σχέσης:

table_id	το αναγνωριστικό του πίνακα
table_name	το όνομα του πίνακα

tenant_id	το πεδίο αυτό περιέχει το αναγνωριστικό του tenant αν ο πίνακας είναι ιδιωτικός, αλλιώς δεν έχει τιμή(null)
extensible	σε περίπτωση που ο πίνακας είναι βασικός, το πεδίο αυτό ορίζει αν μπορεί να επεκταθεί από τους tenants
preallocated_fields_num	αν ο πίνακας είναι βασικός, το πεδίο αυτό προσδιορίζει τον αριθμό των γενικών του πεδίων

Πίνακας "tenants"

Στον πίνακα αυτό αποθηκεύουμε την πληροφορία για τους tenants. Η σημασία των περισσότερων πεδίων είναι προφανής. Αρκεί να αναφέρουμε ότι ως admin_login_name και ως admin_password εννοούμε το αναγνωριστικό και τον κωδικό πρόσβασης αντίστοιχα για σύνδεση στο σύστημα και διαχείριση του σχήματος του tenant.

Πίνακας "fields"

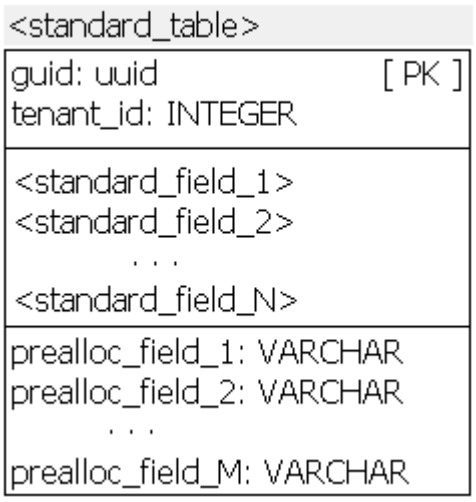
Εδώ αποθηκεύουμε την πληροφορία σχετικά με τα πεδία της βάσης. Αποθηκεύουμε και τα κοινά αλλά και τα ιδιωτικά πεδία των tenants.

field_id	το αναγνωριστικό του πεδίου
tenant_id	το αναγνωριστικό του tenant στον οποίο ανήκει το πεδίο
table_id	το αναγνωριστικό του πίνακα που περιέχει το πεδίο
field_name	το όνομα του πεδίου
position	ο αριθμός της γενικής στήλης που θα αποθηκεύεται ένα ιδιωτικό πεδίο είτε στο γενικό πίνακα(αν αποθηκεύεται στο chunk table) είτε σε κάποιο βασικό
chunk_num	αν το πεδίο είναι ιδιωτικό και αποθηκεύεται στο chunk table, το πεδίο αυτό δηλώνει τον αριθμό του αντίτοιχου chunk(μια λογική

	σχέση μπορεί να διαμερίζεται σε πολλαπλά chunks)
datatype	ένας ακέραιος που αντιπροσωπεύει τον τύπο δεδομένων του πεδίου
not_null	αν η boolean αυτή τιμή είναι αληθής, τότε το πεδίο δε θα μπορεί να πάρει τιμή null
is_unique	ύπαρξη ή όχι περιορισμού μοναδικότητας
field_order	η σειρά του πεδίου στον πίνακα
target_table_id	σε περίπτωση που το συγκεκριμένο πεδίο δείχνει σε κάποιον πίνακα, δηλαδή περιέχει ένα ξένο κλειδί, το πεδίο target_table_id περιέχει το αναγνωριστικό αυτού του πίνακα

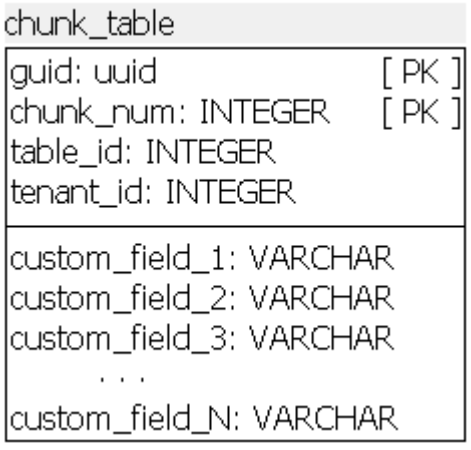
5.3.2 Φυσικό σχήμα διαμοιραζόμενων πινάκων

Στην παράγραφο αυτή θα παρουσιάσουμε το σχεσιακό διάγραμμα των πινάκων που θα περιέχουν τα πραγματικά δεδομένα της multi-tenant βάσης. Στην Εικ. 5-9 βλέπουμε τη μορφή που θα πρέπει να έχει ένας βασικός πίνακας. Αρχικά, θα πρέπει να έχει δυο υποχρεωτικά πεδία: το πεδίο guid και το πεδίο tenant_id. Μετά, θα ορίζονται τα βασικά πεδία, όπου θα δηλώνονται με το κανονικό τους όνομα. Τέλος, ακολουθούν τα γενικά πεδία τα οποία πρέπει να ονομάζονται prealloc_field_n, όπου n ο αριθμός του συγκεκριμένου γενικού πεδίου. Η αρίθμηση θα ξεκινάει από το 1 μέχρι και τον αριθμό που δηλώθηκε στο πεδίο “prealloc_fields_num” της αντίστοιχης εγγραφής του πίνακα μεταδεδομένων “tables”.



Εικόνα 5-9

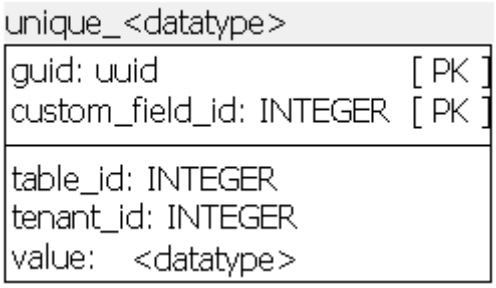
Στην Εικ. 5-10 φαίνεται η μορφή που πρέπει να έχει ο πίνακας chunk_table. Αρχικά, υπάρχουν τα πεδία τα οποία καθορίζουν το αναγνωριστικό της λογικής εγγραφής(guid) στην οποία ανήκει μια εγγραφή στο chunk_table, καθώς και τον αριθμό του αντίστοιχου κομματιού(chunk). Υπάρχουν, επίσης, πεδία που δείχνουν τον πίνακα στον οποίο ανήκει η εγγραφή καθώς και τον tenant. Το πρωτεύον κλειδί του chunk_table είναι ο συνδυασμός του guid και του chunk_num, μιας και μια λογική εγγραφή μπορεί να αποτελείται από πολλαπλά chunks. Ακολουθούν τα πεδία που θα περιέχουν τα κανονικά δεδομένα, τα οποία θα ονομάζονται custom_field_n, όπου n ο αύξων αριθμός της γενικής στήλης στο chunk_table.



Εικόνα 5-10

Πέραν από το ευρετήριο που θα αφορά το πρωτεύον κλειδί του chunk_table, θα πρέπει να διατηρούμε και ένα ευρετήριο για το πεδίο table_id για αποδοτικότερη εκτέλεση των ερωτημάτων καθώς και των λειτουργιών εξατομίκευσης(π.χ. διαγραφή ιδιωτικού πίνακα).

Τέλος, επαναλαμβάνουμε για λόγους πληρότητας τη μορφή των πινάκων ελέγχου μοναδικότητας ιδιωτικών πεδίων. Θα υπάρχει ένας τέτοιος πίνακας για κάθε τύπο δεδομένων που θα μπορούμε να ορίσουμε σε ένα ιδιωτικό πεδίο. Στην παρούσα σχεδίαση του συστήματος με τους πέντε τύπους δεδομένων που αναφέραμε στην 5.2.3, θα έχουμε πέντε τέτοιους πίνακες. Το σχεσιακό σχήμα καθενός από αυτούς θα είναι όπως αυτό που φαίνεται στην Εικ. 5-11, με τη διαφορά ότι και στο όνομα του πίνακα, αλλά και στον τύπο δεδομένων του πεδίου value, θα πρέπει να αντικαταστήσουμε τον αντίστοιχο τύπο. Επαναλαμβάνουμε ότι σε κάθε τέτοιο πίνακα θα υπάρχει ένα ευρετήριο μοναδικότητας(unique index) στο συνδυασμό custom_field_id και value. Επίσης, θα υπάρχει και ένα κανονικό ευρετήριο στο πεδίο table_id(π.χ. για αποδοτικότερη διαγραφή των εγγραφών κατά τη διαγραφή ενός ιδιωτικού πίνακα)



Εικόνα 5-91

6

Υλοποίηση

Στο κεφάλαιο αυτό, αφού αναφερθούμε αρχικά στα εργαλεία και στις πλατφόρμες που χρησιμοποιήσαμε για την ανάπτυξη του συστήματος, παρουσιάζουμε κάποια σημαντικά ζητήματα σχετικά με την υλοποίηση του.

6.1 Πλατφόρμες και προγραμματιστικά εργαλεία

Στην παρούσα ενότητα αναφέρονται τα προγραμματιστικά εργαλεία και οι πλατφόρμες ανάπτυξης λογισμικού που χρησιμοποιήσαμε.

Η ανάπτυξη του συστήματος έγινε με την αντικειμενοστρεφή γλώσσα προγραμματισμού Java (Java SE 7), χρησιμοποιώντας την πλατφόρμα ανάπτυξης Eclipse.

6.1.1 Εργαλεία

6.1.1.1 Hibernate

[20]

Το Hibernate είναι ένα ανοιχτού κώδικα ORM εργαλείο για τη Java που υποστηρίζει τη διαφανή διατήρηση των Plain Old Java Objects (POJOs) σε πίνακες μιας σχεσιακής βάσης δεδομένων. Η απεικόνιση των POJOs στους πίνακες πραγματοποιείται μέσω αρχείων XML ή

μέσω των Java Annotations. Το Hibernate παρέχει επίσης μηχανισμό δημουργίας ερωτημάτων στη βάση και ανάκτησης των αποτελεσμάτων στη μορφή αντικειμένων που αυτά απεικονίζονται. Με τη χρήση του, οι εφαρμογές είναι εύκολα μεταφέρσιμες σε άλλα υποστηριζόμενα από αυτό ΣΔΒΔ.

Στο σύστημα μας χρησιμοποιήσαμε την έκδοση 4.2.5 του Hibernate.

6.1.1.2 PostgreSQL

[19]

Το ΣΔΒΔ που χρησιμοιήσαμε είναι η PostgreSQL(v9.3)[19]. Η PostgreSQL αποτελεί ένα ανοιχτού κώδικα αντικειμενο—σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων. Είναι ACID συμβατή (ACID compliant), έχει ολοκληρωμένη υποστήριξη για foreign keys, joins, views, triggers, και stored procedures (σε διάφορες γλώσσες προγραμματισμού) και συμπεριλαμβάνει τα περισσότερα SQL92 και SQL99 data types. Προσπαθήσαμε να χρησιμοποιήσουμε μόνο τις σχεσιακές ιδιότητες της Postgresql, έτσι ώστε το σύστημα να είναι σχετικά εύκολο να τροποποιηθεί για να λειτουργήσει και με κάποιο άλλο ΣΔΒΔ(που είναι καθαρά σχεσιακό και όχι αντικειμενο-σχεσιακό). Στη μεταφερσιμότητα αυτή συμβάλλει το εργαλείο ORM που χρησιμοποιήσαμε για τη διατήρηση των μεταδεδομένων, και που μπορεί να λειτουργήσει με ελάχιστες τροποποιήσεις με διάφορα ΣΔΒΔ. Εντούτοις, αναπτύξαμε κάποιες stored procedures με τη γλώσσα PL/pgSQL που δεν υποστηρίζεται γενικά από άλλα συστήματα, μιας και δημιουργήθηκε εξ'αρχής για την PostgreSQL. Αυτές οι συναρτήσεις θα έπρεπε να ξαναγραφούν σε κάποια γλώσσα που να υποστηρίζει το εκάστοτε ΣΔΒΔ.

6.1.1.3 Ehcache

[21]

Η Ehcache είναι μια βιβλιοθήκη ανοιχτού κώδικα για δημιουργία και διαχείριση γρήγορων και thread-safe προσωρινών χώρων αποθήκευσης(caches) στη Java. Μπορούν να υποστηριχθούν τόσο caches στη μνήμη όσο και στο δίσκο, καθώς και caches που μοιράζονται σε πολλαπλούς εξυπηρετητές. Η χρήση μιας cache μπορεί να βελτιώσει σημαντικά την απόδοση μιας εφαρμογής, καθώς μειώνει τις απαραίτητες προσβάσεις στη βάση δεδομένων. Εμείς θα χρησιμοποιήσουμε τη βιβλιοθήκη αυτή για τη δημιουργία μιας cache για τα μεταδεδομένα των tenants, όπως εξηγήσαμε σε προηγούμενο κεφάλαιο.

6.1.1.4 JsqlParser

[22]

Ένα SQL ερώτημα κάποιου tenant είδαμε ότι πριν επανεγγραφεί από το σύστημα πρέπει να αναλυθεί συντακτικά και να μετατραπεί σε μορφή κατάλληλη για επεξεργασία. Την ανάλυση αυτή την κάνουμε με χρήση της βιβλιοθήκης JSqlParser. Η βιβλιοθήκη αυτή, που είναι ανοιχτού κώδικα, αναλύει συντακτικά ένα sql ερώτημα και το μεταφράζει σε μια ιεραρχία κλάσεων της Java. Μπορούμε να πλοηγηθούμε σε αυτή την ιεραρχία μέσω του σχεδιαστικού μοτίβου "Visitor". Ο συντακτικός αναλυτής της βιβλιοθήκης έχει δημιουργηθεί με τη γεννήτρια συντακτικών αναλυτών JavaCC, και υποστηρίζει τις περισσότερες από τις δομές της γλώσσας SQL που χρειαζόμαστε για το σύστημα μας.

6.1.1.5 Java UUID Generator

[23]

Πρόκειται μια μια ανοιχτού-κώδικα βιβλιοθήκη για παραγωγή αναγνωριστικών GUID(που αναφέρονται και ως UUID) σύμφωνα με το "IETF UUID specification". Η βιβλιοθήκη αυτή μπορεί να παράγει και τους τρεις επίσημους(όπως ορίζονται στο παραπάνω specification) τύπους τέτοιων αναγνωριστικών. Στα πλαίσια του συστήματος χρησιμοποιήσαμε τη γεννήτρια που παρέχει η βιβλιοθήκη για παραγωγή ακολουθιακών GUIDs.

6.2 Λεπτομέρειες υλοποίησης

Έχοντας παρουσιάσει τη γενική αρχιτεκτονική και σχεδίαση του συστήματος, στην ενότητα αυτή, θα παρουσιάσουμε κάποια κομμάτια του συστήματος που αξίζουν ιδιαίτερη αναφορά προκειμένου να γίνει κατανοητός ο τρόπος υλοποίησής τους.

6.2.1 Λειτουργίες επανεγγραφής ερωτημάτων

Αν και εξετάσαμε την επανεγγραφή ερωτημάτων για την τεχνική αντιστοίχισης που χρησιμοποιούμε στο σύστημα σε προηγούμενο κεφάλαιο, στην παράγραφο αυτή, παρουσιάζουμε το πως υλοποιούμε τις λειτουργίες αυτές στις αντίστοιχες μεθόδους των κλάσεων του συστήματος.

Αρχικά, ένα ερώτημα δίνεται ως παράμετρος(ως String) στη μέθοδο "rewriteQuery(String query, Tenant tenant)" της κλάσης QueryRewriter. Η μέθοδος αυτή, καλώντας μεθόδους της βιβλιοθήκης JSqlParser, αναλύει συντακτικά το ερώτημα και το μετατρέπει σε κατάλληλη μορφή αντικειμένου ανάλογα με τον τύπο του ερωτήματος. Για παράδειγμα, αν πρόκειται για ένα ερώτημα τύπου select, τότε θα δημιουργηθεί ένα αντικείμενο της κλάσης Select που θα

αντιστοιχεί στο συγκεκριμένο ερώτημα. Η κλάση Select, όπως και άλλες κλάσεις που αναπαριστούν δομές της γλώσσας sql(π.χ.Insert, Expression, FromItem κ.λπ.) ορίζονται στη βιβλιοθήκη JSqlParser. Μετά τη συντακτική ανάλυση του ερωτήματος, καλούμε την κατάλληλη μέθοδο της κλάσης QueryRewriter για επανεγγραφή συγκεκριμένου τύπου ερωτήματων, ανάλογα με το αντικείμενο που επιστράφηκε μετά την ανάλυση(τύπου Select, Insert, Update ή Delete). Ακολουθεί η περιγραφή των μεθόδων αυτών.

6.2.1.1 rewriteSelect

Η μέθοδος "rewriteSelect(Select select, Tenant tenant)" παίρνει ένα αντικείμενο της κλάσης Select το οποίο αφορά στο λογικό σχήμα του tenant. Αφού κάνει τις απαραίτητες τροποποιήσεις ώστε να αντιστοιχεί στο φυσικό σχήμα της βάσης, το επιστρέφει σε μορφή String.

Η διαδικασία που ακολουθεί η "rewriteSelect" για την επανεγγραφή είναι η ακόλουθη:

1. Ανάλυση του ερωτήματος για εύρεση των λογικών πινάκων που αναφέρονται, αλλά και των αντίστοιχων στηλών για κάθε έναν από αυτούς(για αποφυγή περιττών join όπως είδαμε στην 3.3.1). Η ανάλυση αυτή γίνεται μέσω της μεθόδου process(select) της κλάσης SelectTablesFinder. Ένα select ερώτημα μπορεί να αποτελείται από υποερωτήματα(subselects) οπότε αναζητούμε τους πίνακες σε όλες τις προτάσεις from(from clause) του ερωτήματος. Για την απλοποίηση της επανεγγραφής ενός select ερωτήματος, θέτουμε ως απαίτηση κάθε αναφορά σε στήλη στο ερώτημα να περιέχει και το όνομα του πίνακα(πιθανώς alias) στον οποίο ανήκει.

2. Έχοντας βρει τους λογικούς πίνακες που αναφέρονται στο ερώτημα, αντικαθιστούμε κάθε έναν από αυτούς με ένα υποερώτημα που τον "ανακασκευάζει" από τους φυσικούς πίνακες της βάσης. Η "ανακατασκευή" αυτή γίνεται με τη μέθοδο "getLogicalTableQuery" της κλάσης TenantSchema. Η μέθοδος αυτή παίρνει, όπως είδαμε, το όνομα ενός λογικού πίνακα και τα ονόματα των ζητούμενων στηλών του, και επιστρέφει ένα ερώτημα select που σχηματίζει το λογικό πίνακα(με τις ζητούμενες μόνο στήλες) από το φυσικό σχήμα της βάσης. Σε περίπτωση που στο αρχικό ερώτημα ζητούνται όλες οι στήλες ενός πίνακα(π.χ. με χρήση έκφρασης select *), τότε το δημιουργούμενο υποερώτημα φορτώνει όλες τις στήλες του πίνακα. Παραδείγματα της μορφής που θα πρέπει να έχει ένα από αυτά τα υποερωτήματα είδαμε στην 3.3.1. Αρκεί να επαναλάβουμε εδώ ότι κάθε ένα από αυτα τα ερωτήματα μπορεί να χρειαστεί να κάνει join ανάμεσα σε ένα βασικό πίνακα και στο Chunk Table(πιθανώς παραπάνω από μια φορά αν ο λογικός πίνακας έχει πολλαπλά chunks). Τα joins αυτά θα πρέπει να συνοδεύονται από κατάλληλες συνθήκες στα

πεδία μεταδεδομένων(π.χ. guid, chunk_num κ.λ.π.). Απαραίτητη σε κάθε περίπτωση είναι η συνθήκη που επιλέγει από τους αντίστοιχους φυσικούς πίνακες μόνο τις εγγραφές του συγκεκριμένου tenant. Φυσικά αν ο λογικός πίνακας είναι ιδιωτικός, θα τον ανακατασκευάζουμε μόνο από το Chunk Table. Επίσης, σε κάθε γενικό πεδίο που χρησιμοποιείται για το λογικό πίνακα θα πρέπει να δίνεται μέσα στο υποερώτημα το όνομα του αντίστοιχου ιδιωτικού πεδίου(π.χ. select prealloc_field_1 as custom_field_name).

3. Έχοντας τροποποιήσει το αρχικό ερώτημα ώστε κάθε αναφορά σε λογικό πίνακα να αντικατασταθεί από αντίστοιχο υποερώτημα ανάκτησης, το μετατρέπουμε σε μορφή String και το επιστρέφουμε για εκτέλεση στο κοινό σχήμα της multi-tenant βάσης δεδομένων.

6.2.1.2 rewriteInsert

Η μέθοδος "rewriteInsert(Insert insert, Tenant tenant)" παίρνει ένα αντικείμενο της κλάσης Insert το οποίο αναπαριστά ένα ερώτημα εισαγωγής σε ένα λογικό πίνακα του tenant και το μετατρέπει στα αντίστοιχα ερωτήματα εισαγωγής στο φυσικό σχήμα.

Στη μέθοδο αυτή αρχικά ελέγχουμε την εγκυρότητα του ερωτήματος. Δηλαδή, επιβεβαιώνουμε ότι υπάρχει πίνακας στο σχήμα του συγκεκριμένου tenant με το όνομα του πίνακα στο οποίο θα γίνει η εισαγωγή. Ένα ερώτημα εισαγωγής μπορεί να έχει μία από τις ακόλουθες μορφές:

a) INSERT INTO table_name (field_name_1, field_name_2, …) VALUES (value1_1, value1_2, …), (value2_1, value2_2, …), ...;

b) INSERT INTO table_name VALUES

(value1_1, value1_2, …), (value2_1, value2_2, …), ...;

Στην πρώτη μορφή θα πρέπει να ελέγξουμε ότι τα ονόματα στηλών που δίνονται αντιστοιχούν σε πεδία του πίνακα. Επίσης, κατά την εισαγωγή μιας εγγραφής θα πρέπει να δίνουμε τιμή σε όλα τα ιδιωτικά πεδία ενός πίνακα που έχουν οριστεί με not-null περιορισμό. Το αν είναι απαραίτητο να δώσουμε τιμή σε ένα βασικό πεδίο ενός βασικού πίνακα ελέγχεται από το ΣΔΒΔ αφού αυτά τα πεδία αντιστοιχούν σε φυσικά πεδία της βάσης. Στη δεύτερη μορφή δεν ορίζουμε ονόματα πεδίων αλλά οι εγγραφές προς εισαγωγή πρέπει να περιέχουν τιμές για όλα τα πεδία στη σειρά με την οποία υπάρχουν στο λογικό πίνακα. Σημειώνουμε ότι τιμές για τα πεδία μεταδεδομένων ορίζονται από το σύστημα και δε μπορούν να δοθούν από ένα tenant. Έχοντας ελέγξει την εγκυρότητα του ερωτήματος, χρησιμοποιούμε τη βιβλιοθήκη

Java UUID Generator για να παράξουμε ένα μοναδικό αναγνωριστικό GUID για κάθε μια από τις λογικές εγγραφές που εισάγονται με το insert ερώτημα.

Όπως προαναφέραμε, η εισαγωγή μιας εγγραφής σε ένα λογικό πίνακα μπορεί να αντιστοιχεί σε παραπάνω από μια εγγραφές στους φυσικούς πίνακες της βάσης. Αν ο λογικός πίνακας είναι βασικός, τότε κάθε νέα λογική εγγραφή θα συνεπάγεται και μια νέα εγγραφή στον αντίστοιχο βασικό φυσικό πίνακα. Σε περίπτωση που ο λογικός πίνακας περιέχει ένα ή περισσότερα chunks, τότε για κάθε λογική εγγραφή προς εισαγωγή σε αυτόν θα πρέπει να εισάγουμε και μια εγγραφή στο Chunk Table για κάθε chunk. Επιπλέον, αν κάποιο **ιδιωτικό** πεδίο έχει ορισθεί με περιορισμό μοναδικότητας, τότε για κάθε τιμή που δίνουμε στο πεδίο αυτό θα πρέπει να εισάγουμε μια εγγραφή στον κατάλληλο πίνακα ελέγχου μοναδικότητας. Όλες οι παραπάνω εισαγωγές ομαδοποιούνται από το σύστημα. Για παράδειγμα, ακόμα και αν έχουμε πολλαπλά chunks, όλες οι εισαγωγές που θα χρειαστεί να πραγματοποιήσουμε στο Chunk Table θα γίνουν στο ίδιο ερώτημα Insert. Φυσικά, κάθε μια θα έχει κατάλληλες τιμές στις στήλες μεταδεδομένων. Το ίδιο θα γίνεται και για τους πίνακες ελέγχου μοναδικότητας, σε περίπτωση που παραπάνω από ένα πεδία ίδιου τύπου δεδομένων έχουν ορισθεί με unique constraint. Η μέθοδος " rewriteInsert ", αφού συμβουλευθεί την πληροφορία για το λογικό σχήμα του tenant(κατάλληλο αντικείμενο της κλάσης TenantSchema), κάνει τις απαραίτητες αντιστοιχήσεις και δημιουργεί τα ερωτήματα insert που αναλογούν στο αρχικό ερώτημα εισαγωγής.

Για τα ιδιωτικά πεδία θα πρέπει στα τελικά ερωτήματα να "περιβάλλουμε" τις τιμές που τους δίνονται στο ερώτημα εισαγωγής με τη συνάρτηση "CAST", όπως είδαμε στην 3.4.1. Όμοια, θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση "check_null(expression)" για να ελέγξουμε ότι οι τιμές που δίνουμε σε ένα ιδιωτικό πεδίο με περιορισμό not-null δεν είναι null. Η συνάρτηση "check_null" είναι μια αποθηκευμένη συνάρτηση που υλοποιήσαμε στην PostgreSQL(με χρήση της γλώσσας PL/pgSQL). Αν ένα λογικό ερώτημα εισαγωγής δίνει σε ένα πεδίο με περιορισμό not-null ως τιμή την έκφραση "exp", στο φυσικό ερώτημα που θα παράξει το σύστημα θα δίνεται στην αντίστοιχη φυσική γενική στήλη η τιμή "check_null(exp)". Έτσι, όταν το φυσικό αυτό ερώτημα εισαγωγής εκτελεσθεί στη βάση, θα κληθεί η check_null και αν η έκφραση exp υπολογισθεί ως null, τότε θα επιστραφεί λάθος και θα απορριφθεί η δοσοληψία.

Αντίστοιχα, υλοποιήσαμε στην PostgreSQL τις αποθηκευμένες συνάρτησεις "check_rel_st(guidExp, table_name, tenant_id)" και "check_rel_cust(guidExp, table_id)" και για το έλεγχο αναφορικής ακεραιότητας. Αν ένα πεδίο περιέχει ένα ξένο κλειδί, τότε η τιμή που ανατίθεται σε αυτό(που θα είναι τύπου guid) σε ένα ερώτημα εισαγωγής θα δίνεται στο τελικό ερώτημα(δηλαδή σε αυτό που παράγεται μετά την επανεγγραφή) με κλήση μιας από τις παραπάνω συναρτήσεις. Αν ο πίνακας που δείχνει το πεδίο αυτό είναι βασικός(και άρα

αντιστοιχεί σε φυσικό πίνακα της βάσης), τότε θα χρησιμοποιούμε τη συνάρτηση "check_rel_st(guidExp, table_name, tenant_id)". Όταν εκτελεσθεί η συνάρτηση αυτή στη βάση δεδομένων, θα αναζητηθεί στο βασικό πίνακα με όνομα "table_name" εγγραφή του αντίστοιχου tenant με αναγνωριστικό "guidExp". Αν δε βρεθεί, θα απορριφθεί η δοσοληψία. Στην περίπτωση που ο πίνακας που δείχνει το πεδίο είναι ιδιωτικός τότε θα χρησιμοποιήσουμε τη συνάρτηση "check_rel_cust(guidExp, table_id)". Η συνάρτηση αυτή αναζητά στο Chunk Table εγγραφή με αναγνωριστικό guidExp που να ανήκει στον ιδιωτικό πίνακα με αναγνωριστικό table_id. Όμοια, αν δε βρεθεί τέτοια εγγραφή απορρίπτεται η δοσοληψία.

Η μέθοδος "rewriteInsert" επιστρέφει τελικά ένα String που περιέχει το σύνολο των ερωτημάτων insert που αντιστοιχούν στο αρχικό ερώτημα εισαγωγής του tenant. Επαναλαμβάνουμε ότι τα ερωτήματα θα στέλνονται τελικά στη βάση στα πλαίσια μιας δοσοληψίας, έτσι ώστε οι αλλαγές να αποθηκεύονται στη βάση μόνο όταν όλα εκτελούνται επιτυχώς.

Αν και παρουσιάσαμε παράδειγμα επανεγγραφής εισαγωγής στην Παρ. 3.3.2, θα δώσουμε και εδώ ένα παράδειγμα το οποίο θα παρουσιάζει όλες τις περιπτώσεις ελέγχου περιορισμών που παρουσιάσαμε παραπάνω. Έστω, λοιπόν, πάλι ο βασικός πίνακας "surveys" τον οποίο μοιράζονται όλοι οι tenants. Στον πίνακα αυτόν, που υπάρχει στο λογικό σχήμα όλων των tenants, τα πεδία "survey_title", "description" και "end_date", όπως και πριν, είναι κοινά για όλους τους tenants(βασικά πεδία). Έστω, όμως, ότι κάθε έρευνα που αποθηκεύει ο tenant 2 έχει και ένα μοναδικό αναγνωριστικό "survey_id" τύπου varchar και διαφορετικό από το αναγνωριστικό guid που χρησιμοποιείται από το σύστημα για το μοναδικό προσδιορισμό κάθε εγγραφής. Το πεδίο αυτό προστίθεται ως ιδιωτικό πεδίο στο λογικό σχήμα του tenant, με περιορισμό μοναδικότητας, καθώς και περιορισμό τύπου not-null. Επιπλέον, ο ίδιος tenant προσθέσει και τρία επιπλέον ιδιωτικά πεδία με όνομα "is_open", "company" και "survey_manager" αντίστοιχα. Το πεδίο "survey_manager" αποτελεί ξένο κλειδί προς ένα πίνακα που περιέχει εγγραφές που αναπαριστούν διαχειριστές ερευνών. Τα πεδία "survey_id" και "is_open" αντιστοιχίζονται στα δυο γενικά πεδία του βασικού πίνακα "surveys", ενώ τα πεδία "company" και "survey_manager" αντιστοιχίζονται σε δυο γενικά πεδία ενός chunk στο chunk_table.

Έστω, λοιπόν, ότι ο tenant 2 στέλνει στο σύστημα το παρακάτω ερώτημα εισαγωγής. Το ερώτημα αυτό εισάγει δύο εγγραφές στον πίνακα "surveys".

INSERT INTO surveys

(survey_title, description, end_date, survey_id, is_open, company, survey_manager)

VALUES

('Customer Satisfaction', 'Yearly customer satisfaction survey', 'Mar 5 2014', 's13n57', true, 'Dot Telecom Co.', 'd2b91b6f-e0f6-11e3-b1f6-c5a5b9dd04f5'),

('Product #432 Launch', 'market research for new product', 'Feb 19 2014', 's13n58', true, 'FrozenYo Co.', 'a57cf6b8-e0fb-11e3-9743-6175c6ba29e9')

Αφού το ερώτημα αυτό αναλυθεί συντακτικά, δίνεται στη "rewriteInsert" η οποία το επανεγγράφει με βάση το λογικό σχήμα του tenant. Με βάση την αντιστοίχιση των ιδιωτικών πεδίων του tenant που αναφέραμε παραπάνω, η "rewriteInsert" επιστρέφει το παρακάτω σύνολο ερωτημάτων, τα οποία εκτελεσμένα στα πλαίσια μιας δοσοληψίας, αντιστοιχούν στο αρχικό ερώτημα που έδωσε ο tenant.

INSERT INTO unique_varchar

(guid, custom_field_id, table_id, tenant_id, value)

VALUES

('cf4cb2b6-e0fb-11e3-bf7f-533a7603bbd6', 33, 2, 2, CAST('s13n57' AS VARCHAR)),

('cf4cb2b6-e0fb-11e3-bf7f-8bc8f30a8010', 33, 2, 2, CAST('s13n58' AS VARCHAR));

INSERT INTO surveys

(guid, tenant_id, end_date, description, survey_title, prealloc_field_1, prealloc_field_2)

VALUES

('cf4cb2b6-e0fb-11e3-bf7f-533a7603bbd6', 2, 'Mar 5 2014', 'Yearly customer satisfaction survey', 'Customer Satisfaction', check_null(CAST('s13n57' AS VARCHAR)), check_null(CAST(true AS BOOLEAN))),

('cf4cb2b6-e0fb-11e3-bf7f-8bc8f30a8010', 2, 'Feb 19 2014', 'market research for new product', 'Product #432 Launch', check_null(CAST('s13n58' AS VARCHAR)), check_null(CAST(true AS BOOLEAN)));

INSERT INTO chunk_table

(guid, chunk_num, table_id, tenant_id, custom_field_1, custom_field_2, custom_field_3, custom_field_4, custom_field_5)

VALUES

('cf4cb2b6-e0fb-11e3-bf7f-533a7603bbd6', 1, 2, 2, CAST('Dot Telecom Co.' AS VARCHAR), check_rel_cust(CAST('d2b91b6f-e0f6-11e3-b1f6-c5a5b9dd04f5' AS UUID), 6), NULL, NULL, NULL),

('cf4cb2b6-e0fb-11e3-bf7f-8bc8f30a8010', 1, 2, 2, CAST('FrozenYo Co.' AS VARCHAR), check_rel_cust(CAST('a57cf6b8-e0fb-11e3-9743-6175c6ba29e9' AS UUID), 6), NULL, NULL, NULL);

Παρατηρούμε, αρχικά, ότι στα ερωτήματα περιλαμβάνεται και ένα ερώτημα εισαγωγής στον πίνακα μοναδικότητας “unique_varchar”, μιας και το ιδιωτικό πεδίο “survey_id”, που είναι τύπου varchar, έχει ορισθεί με περιορισμό μοναδικότητας. Επίσης, βλέπουμε το πως εξασφαλίζεται ο περιορισμός ξένου κλειδιού που υπάρχει για το πεδίο “survey_manager” με τη χρήση της αποθηκευμένης συνάρτησης “ check_rel_cust ”.

6.2.1.3 rewriteUpdate

Όπως και οι προηγούμενες μέθοδοι, η “rewriteUpdate(Update update, Tenant tenant)” λαμβάνει ως παράμετρο το αντικείμενο που προκύπτει μετά από την ανάλυση του sql ερωτήματος(εδώ ερώτημα update), καθώς και τον tenant που έστειλε το ερώτημα. Όπως περιγράψαμε στην 3.3.2, κατά την επανεγγραφή ενός ερωτήματος update, αρχικά συλλέγουμε το σύνολο των αναγνωριστικών GUID των λογικών εγγραφών που θα επηρεαστούν. Στην ίδια παράγραφο είδαμε ότι το γεγονός ότι ένα λογικό update μπορεί να αντιστοιχεί σε πολλαπλά updates στους φυσικούς πίνακες της βάσης, μπορεί να οδηγήσει σε υπολογισμό του συνόλου αυτού παραπάνω από μια φορές. Για την αποφυγή των περιττών αυτών υπολογισμών επιλέξαμε να υπολογίζουμε το σύνολο των GUIDs μία φορά και να το

αποθηκεύουμε σε ένα προσωρινό πίνακα. Ο πίνακας αυτός θα υπάρχει μόνο στα πλαίσια της δοσοληψίας που θα εκτελούνται τα ερωτήματα που αντιστοιχούν στο αρχικό ερώτημα update.

Η διαδικασία που ακολουθείται από την "rewriteUpdate" για την επανεγγραφή είναι η ακόλουθη:

1. Αρχικά ελέγχουμε ότι το ερώτημα είναι έγκυρο, δηλαδή ότι αναφέρεται σε κάποιο πίνακα που υπάρχει στο λογικό σχήμα του tenant και ότι τροποποιεί τις τιμές υπαρκτών λογικών πεδίων του πίνακα. Επίσης, ελέγχουμε ότι δεν επιχειρείται μέσα στο ερώτημα να τροποποιηθεί η τιμή κάποιου από τα πεδία μεταδεδομένων, καθώς και του πεδίου GUID. Το πεδίο GUID, αν και ορατό στους tenants, μπορεί να πάρει τιμή μόνο από το σύστημα και μόνο κατά την εισαγωγή μιας νέας εγγραφής.

2. Απομονώνουμε το "where" κομμάτι του ερωτήματος και από αυτό σχηματίζουμε ένα ερώτημα select με ίδιο "where" κομμάτι, στο οποίο ζητούμε να επιστραφεί το πεδίο guid. Στο "from" κομμάτι του ερωτήματος βάζουμε το όνομα του λογικού πίνακα που τροποποιεί το αρχικό ερώτημα τροποποίησης. Το select αυτό ερώτημα αντιστοιχεί προφανώς στο λογικό σχήμα του tenant και πρέπει να επανεγγραφεί ως προς το φυσικό σχήμα. Για την επανεγγραφή αυτή χρησιμοποιούμε τη μέθοδο rewriteSelect που εξετάσαμε παραπάνω. Τελικά, προκύπτει ένα ερώτημα που υπολογίζει το σύνολο των αναγνωριστικών GUIDs που θα τροποποιήσει το ερώτημα update που έστειλε ο tenant.

3. Για να αποφύγουμε, όπως προαναφέραμε, το πρόβλημα της πολλαπλή εκτέλεσης του παραπάνω select ερωτήματος, θα συμπεριλάβουμε μαζί με τα τελικά ερωτήματα update(που αντιστοιχούν στο αρχικό ερώτημα), και ένα ερώτημα δημιουργίας ενός προσωρινού πίνακα. Ο πίνακας αυτός, που θα υπάρχει μόνο στα πλαίσια της δοσοληψίας που θα εκτελεστούν τα τελικά ερωτήματα updates, θα περιέχει τα αποτελέσματα που προκύπτουν μετά την εκτέλεση του παραπάνω ερωτήματος select.

4. Στη συνέχεια, ελέγχουμε τα λογικά πεδία που τροποποιεί το αρχικό ερώτημα, και δημιουργούμε τα κατάλληλα ερωτήματα updates. Αν ο πίνακας είναι βασικός και αν τροποποιούμε κάποιο βασικό πεδίο ή κάποιο πεδίο επέκτασης που αποθηκεύεται σε γενικό πεδίο του βασικού πίνακα, τότε θα δημιουργήσουμε ένα ερώτημα update για το βασικό αυτό πίνακα. Αν το αρχικό ερώτημα τροποποιεί λογικά πεδία που αποθηκεύονται σε κάποιο chunk, θα δημιουργήσουμε ένα ερώτημα update για κάθε chunk του λογικού πίνακα που τροποποιείται(με κατάλληλη τιμή του πεδίου μεταδεδομένων chunk_num). Αν τροποποιείται κάποιο πεδίο με περιορισμό μοναδικότητας, τότε θα πρέπει να τροποιήσουμε και τις αντίστοιχες εγγραφές στον κατάλληλο πίνακα ελέγχου μοναδικότητας. Στα ερωτήματα αυτά, ο έλεγχος των

περιορισμών(π.χ. not-null constraint, περιορισμός αναφορικής ακεραιότητας) που μπορούν να έχουν ορισθεί για κάθε ένα από τα πεδία που τροποποιεί το ερώτημα, εξασφαλίζεται με τους ίδιους τρόπους που εξετάσαμε στη μέθοδο “rewriteInsert” για την επανεγγραφή ερωτημάτων εισαγωγής.

Έστω ότι αποστέλλεται στο σύστημα το παρακάτω ερώτημα τροποποίησης που αφορά το λογικό πίνακα “surveys” που εξετάσαμε παραπάνω:

```
UPDATE surveys s
SET s.survey_manager = 'd2b91b6f-e0f6-11e3-b1f6-c5a5b9dd04f5' ,
    s.survey_id = 's13d2'
WHERE s.survey_id = 's13n62'
```

Η “rewriteUpdate” θα επιστρέψει το παρακάτω σύνολο ερωτημάτων:

```
CREATE TEMP TABLE tmp230589632 ON COMMIT DROP AS
    SELECT s.guid FROM
        (SELECT surveys.guid,
                CAST(surveys.prealloc_field_1 AS VARCHAR) AS survey_id
         FROM surveys WHERE surveys.tenant_id = 2) AS s
    WHERE s.survey_id = 's13n62'

UPDATE surveys SET
        prealloc_field_1 = check_null(CAST('s13d2' AS VARCHAR))
WHERE guid IN (SELECT guid FROM tmp230589632);

UPDATE chunk_table  SET
        custom_field_2 =
        check_rel_cust(CAST('d2b91b6f-e0f6-11e3-b1f6-c5a5b9dd04f5' AS  UUID), 6)
```

```
WHERE chunk_num = 1 AND guid IN (SELECT guid FROM tmp230589632);


UPDATE unique_varchar AS u SET
        value = CAST(t.prealloc_field_1 AS VARCHAR)
FROM surveys AS t
WHERE  t.guid = u.guid AND
         u.custom_field_id = 33 AND
         u.guid IN (SELECT guid FROM tmp230589632);
```

Παρατηρούμε ότι αρχικά δημιουργείται ένας προσωρινός πίνακας στον οποίο διατηρείται το σύνολο των guids των λογικών εγγραφών που θα τροποποιηθούν στον πίνακα "surveys". Βλέπουμε, επίσης, ότι μιας και το αρχικό ερώτημα τροποποιεί και βασικά πεδία του λογικού πίνακα αλλά και πεδία που αντιστοιχίζονται σε κάποιο chunk, θα πρέπει να τροποποιήσουμε τις αντίστοιχες εγγραφές και στο φυσικό πίνακα "surveys" αλλά και στο chunk_table. Η ύπαρξη, επιπλέον, περιορισμού μοναδικότητας στο ιδιωτικό πεδίο "survey_id" σημαίνει ότι θα πρέπει να διαγράψουμε και τις αντίστοιχες εγγραφές από τον πίνακα ελέγχου μοναδικότητας "unique_varchar".

6.2.1.4 rewriteDelete

Η "rewriteDelete(Delete delete, Tenant tenant)" λαμβάνει το αντικείμενο τύπου Delete που προκύπτει μετά από την ανάλυση ενός λογικού ερωτήματος διαγραφής, και επιστρέφει σε μορφή String τα ερωτήματα που αντιστοιχούν σε αυτό ως προς το φυσικό σχήμα. Η διαδικασία που ακολουθεί η μέθοδος αυτή είναι η ακόλουθη:

1. Αρχικά ελέγχουμε ότι το ερώτημα είναι έγκυρο και ότι αναφέρεται σε κάποιο πίνακα που υπάρχει στο λογικό σχήμα του tenant.
2. Όπως και στο αντίστοιχο βήμα της μεθόδου rewriteUpdate, σχηματίζουμε ένα ερώτημα select το οποίο, αφού επανεγγραφεί από τη μέθοδο rewriteSelect, χρησιμοποιείται για τον υπολογισμό του σύνολου των αναγνωριστικών GUIDs που θα τροποποιήσει το αρχικό ερώτημα delete που έστειλε ο tenant. Το ερώτημα αυτό, όπως και στην περίπτωση της rewriteUpdate, θα εκτελείται μια φορά και τα αποτελέσματα θα αποθηκεύονται σε ένα προσωρινό πίνακα.

3. Για κάθε μία από τις λογικές εγγραφές που θέλει να διαγράψει ο tenant με το αρχικό delete ερώτημα του, μπορεί να υπάρχουν πολλαπλές εγγραφές στο chunk table και σε κάποιους από τους πίνακες ελέγχου μοναδικότητας(σε περίπτωση που κάποια ιδιωτικά πεδία του πίνακα έχουν ορισθεί με περιορισμό μοναδικότητας). Επίσης, σε περίπτωση που ο πίνακας είναι βασικός, μπορεί να υπάρχει και μια εγγραφή στον αντίστοιχο φυσικό βασικό πίνακα. Τις εγγραφές αυτές μπορούμε να τις σβήσουμε εκτελώντας ένα delete ερώτημα για κάθε ένα από αυτούς τους πίνακες στο οποίο θα επιλέγουμε τις εγγραφές με αναγνωριστικό που ανήκει στο σύνολο που υπολογίζει το ερώτημα που σχηματίσαμε στο βήμα 2.

Έστω το παρακάτω ερώτημα διαγραφής που αφορά το λογικό πίνακα “surveys” που εξετάσαμε παραπάνω:

```
DELETE FROM surveys
WHERE surveys.is_open = 'true'
```

Η “rewriteDelete” θα επιστρέψει το παρακάτω σύνολο ερωτημάτων:

```
CREATE TEMP TABLE tmp170579611 ON COMMIT DROP AS
        SELECT surveys.guid FROM
                (SELECT surveys.guid,
                        CAST(surveys.prealloc_field_2 AS BOOLEAN) AS is_open
                FROM surveys WHERE surveys.tenant_id = 2) AS surveys
        WHERE surveys.is_open = 'true';

DELETE FROM surveys
WHERE guid IN (SELECT guid FROM tmp170579611);

DELETE FROM chunk_table
WHERE guid IN (SELECT guid FROM tmp170579611);
```

DELETE FROM unique_varchar

WHERE guid IN (SELECT guid FROM tmp170579611);

Παρατηρούμε, όπως και στην περίπτωση της επανεγγραφής ερωτημάτων update, ότι αρχικά δημιουργείται ένας προσωρινός πίνακας στον οποίο διατηρείται το σύνολο των guids των λογικών εγγραφών που θα διαγραφούν από τον πίνακα "surveys". Μιας και ο πίνακας είναι βασικός, με κάποια ιδιωτικά του πεδία να διατηρούνται σε κάποιο chunk, θα πρέπει να διαγράψουμε τις αντίστοιχες εγγραφές και από το φυσικό πίνακα "surveys" αλλά και από το chunk_table. Η ύπαρξη, επιπλέον, περιορισμού μοναδικότητας στο ιδιωτικό πεδίο "survey_id" σημαίνει ότι θα πρέπει να διαγράψουμε και τις αντίστοιχες εγγραφές από τον πίνακα ελέγχου μοναδικότητας "unique_varchar".

6.2.2 Λειτουργίες εξατομίκευσης σχήματος

Στην παράγραφο αυτή θα δούμε τον τρόπο που διαχειρίζεται το σύστημα την εξατομίκευση του σχήματος ενός tenant. Δε θα εξετάσουμε τις λειτουργίες μετονομασίας ενός ιδιωτικού πίνακα ή ενός ιδιωτικού πεδίου, καθώς αυτές απλά αλλάζουν τις τιμές που αφορούν το όνομα στο επιλεγμένο αντικείμενο Table ή CustomField , και προωθούν τις αλλαγές στον αντίστοιχο πίνακα μεταδεδομένων στη βάση.

6.2.2.1 Προσθήκη νέου πεδίου

Η προσθήκη ενός νέου ιδιωτικού πεδίου γίνεται μέσω της μεθόδου

addCustomField(String *fieldName*, int *datatype*, boolean *unique*, boolean *notNull*,

String *foreignTableName*)

της κλάσης Table. Η κλάση Table, όπως προείπαμε, αναπαριστά ένα λογικό πίνακα της multi-tenant βάσης και περιέχει λίστα με αναφορές σε όλα τα πεδία του πίνακα αυτού. Ένα Table αφορά έναν tenant, δηλαδή, ακόμα και αν αναπαριστά ένα βασικό πίνακα θα περιέχει αναφορές σε CustomFields του συγκεκριμένου tenant και μόνο.

Η διαδικασία που ακολουθούμε για την προσθήκη ενός πεδίου είναι η εξής:

1. Έλεγχος ότι δεν υπάρχει ήδη πεδίο με το ίδιο όνομα και ότι η τιμή datatype αντιστοιχεί σε υποστηριζόμενο από το σύστημα τύπο δεδομένων.
2. Σε περίπτωση που το νέο πεδίο θα περιέχει ένα ξένο κλειδί προς κάποιο πίνακα, έλεγχος ότι το *foreignTableName* αντιστοιχεί σε κάποιο πίνακα στο λογικό σχήμα του tenant.
3. Αν το νέο πεδίο ορίζεται με περιορισμό not-null, τότε πρέπει να ελέγξουμε ότι ο αντίστοιχος πίνακας δεν περιέχει ήδη κάποια εγγραφή στη βάση(προφανώς για το συγκεκριμένο tenant, αν ο πίνακας είναι βασικός) και αν υπάρχει εγγραφή δε μπορούμε να προσθέσουμε το νέο πεδίο.
4. Δημιουργία ενός νέου στιγμιοτύπου της κλάσης CustomField και κατάλληλη αρχικοποίηση της με βάση τις παραμέτρους που δόθηκαν στην addCustomField.
5. Εύρεση της φυσικής στήλης στη βάση που θα αποθηκεύεται το πεδίο.

   Αν το πεδίο προστίθεται σε ένα βασικό πίνακα, τότε αναζητούμε πρώτα μια γενική στήλη, που δε χρησιμοποιείται ήδη για κάποιο πεδίο-επέκταση, σε αυτόν. Αν δε βρούμε τέτοια στήλη ή αν ο πίνακας είναι ιδιωτικός, τότε ψάχνουμε κάποια ελεύθερη θέση σε κάποιο υπάρχον chunk και αν δε βρούμε δημιουργούμε νέο chunk.
6. Σε περίπτωση που δε χρειάζεται να δημιουργήσουμε κάποιο νέο chunk για το πεδίο, τότε αρκεί, καλώντας την κατάλληλη μέθοδο της κλάσης CustomFieldDAO, να αποθηκεύσουμε και στη βάση την πληροφορία για το νέο πεδίο. Αν όμως δεν υπάρχει ελεύθερη γενική στήλη σε κάποιο υπάρχον chunk και αναγκαστούμε να δημιουργήσουμε νέο, θα πρέπει να εισάγουμε μια νέα εγγραφή(με chunk_num τον αριθμό του νέου chunk) στο chunk_table για κάθε εγγραφή του λογικού πίνακα. Τελικά, η αποθήκευση της πληροφορίας για το νέο πεδίο στον πίνακα μεταδεδομένων “fields” θα πρέπει να ακολουθήσει την παραπάνω εισαγωγή στο chunk_table, εφόσον αυτή είναι επιτυχής.

Να σημειώσουμε ότι η ίδια μέθοδος καλείται και για την προσθήκη ενός πεδίου σε ένα Table που αντιστοιχεί σε νέο ιδιωτικό πίνακα, πριν αυτός αποθηκευτεί στο σχήμα μεταδεδομένων.

6.2.2.2 Προσθήκη νέου πίνακα

Η προσθήκη ενός ιδιωτικού πίνακα στο σχήμα ενός tenant γίνεται με τη μέθοδο addCustomTable(Table table) της κλάσης TenantSchema. Η μέθοδος αυτή παίρνει ένα στιγμιότυπο της κλάσης Table το οποίο δε διατηρείται στη βάση(δηλαδή δεν έχει γίνει ακόμη

persistent), και απλά το αποθηκεύει σε αυτή. Το στιγμιότυπο αυτό θα πρέπει να έχει αρχικοποιηθεί κατάλληλα(π.χ. να έχουμε θέσει με την setTableName(name) το όνομα του νέου πίνακα κ.λ.π.) και θα πρέπει να έχουμε προσθέσει τα πεδία που θα έχει ο νέος πίνακας μέσω της μεθόδου addCustomField που είδαμε παραπάνω . Φυσικά, η addCustomTable ελέγχει ότι δεν υπάρχει πίνακας(είτε βασικός είτε ιδιωτικός) με το ίδο όνομα στο σχήμα του tenant. Πέραν από τα στοιχεία του ίδιου του νέου πίνακα(που θα μπουν στον πίνακα μεταδεδομένων tables), η μέθοδος αυτή θα αποθηκεύσει τελικά στη βάση και την πληροφορία για τα πεδία που θα έχει αυτός. Αυτό γίνεται αυτόματα από το Hibernate με κατάλληλη ρύθμιση.

Παρατηρούμε, λοιπόν, ότι η δημιουργία ενός πίνακα από έναν tenant αντιστοιχεί απλά στην εισαγωγή κατάλληλων εγγραφών στους πίνακες μεταδεδομένων και όχι σε δημιουργία(μέσω π.χ. ddl) ενός φυσικού πίνακα στη βάση.

6.2.2.3 Διαγραφή πίνακα

Ένας tenant επιτρέπεται να διαγράψει μόνο ιδιωτικούς του πίνακες. Η διαγραφή ενός τέτοιου πίνακα γίνεται στη μέθοδο dropCustomTable(String tableName) της κλάσης TenantSchema. Η μέθοδος αυτή ψάχνει στο σχήμα του tenant για ιδιωτικό πίνακα με το όνομα tableName και εφόσον το βρει, διαγράφει την αναφορά και από τη λίστα με τους πίνακες του(στην κλάση TenantSchema) αλλά και από το σχήμα μεταδεδομένων στη βάση. Όπως και κατά την προσθήκη ενός πίνακα, με κατάλληλη ρύθμιση του Hibernate διαγράφουμε και τις εγγραφές στον πίνακα μεταδεδομένων “fields” που αντιστοιχούν στο συγκεκριμένο πίνακα. Σε περίπτωση που υπάρχει ένα ξένο κλειδί προς τον πίνακα που διαγράφουμε(δηλαδή σε κάποια εγγραφή του πίνακα “fields” το πεδίο “target_table_id” έχει τιμή το αναγνωριστικό του πίνακα που διαγράφουμε), τότε η διαγραφή της αντίστοιχης εγγραφής στον πίνακα μεταδεδομένων “tables” θα απορριφθεί από το ΣΔΒΔ.

Ο πίνακας που επιθυμεί να διαγράψει ο tenant ενδεχομένως περιέχει δεδομένα. Αυτό σημαίνει ότι μπορεί να υπάρχουν εγγραφές στο chunk_table για τον πίνακα αυτόν. Ακόμη, σε περίπτωση που έχει και πεδία με περιορισμούς μοναδικότητας, θα υπάρχουν εγγραφές και στους αντίστοιχους πίνακες μοναδικότητας(unique_datatype). Αυτές οι εγγραφές θα πρέπει τελικά να σβηστούν. Μπορούμε, λοιπόν, στην ίδια δοσοληψία με τα ερωτήματα διαγραφής μεταδεδομένων(που θα παραχθούν από το Hibernate), να συμπεριλάβουμε και το παρακάτω ερώτημα:

```
DELETE FROM chunk_table WHERE table_id = #id
```

Ως #id εννοούμε την τιμή του αναγνωριστικού του πίνακα που διαγράφουμε. Επίσης, αν ο πίνακας έχει και unique πεδία, θα πρέπει να συγκεντρώσουμε τους τύπους δεδομένων των πεδίων αυτών και να εκτελέσουμε για κάθε έναν από αυτούς ένα ερώτημα διαγραφής της ακόλουθης μορφής:

DELETE FROM unique_<datatype> WHERE table_id = #id

Σε κάθε τέτοιο ερώτημα θα πρέπει να βάλουμε το σωστό όνομα πίνακα μοναδικότητας, π.χ. unique_numeric ή unique_datetime.

Η παραπάνω προσέγγιση, αν και λύνει το πρόβλημα, και που στα πλαίσια της παρούσας διπλωματικής εργασίας κρίθηκε αρκετή, δεν είναι η ιδανική. Ερωτήματα μαζικής τροποποίησης πινάκων συνήθως καθυστερούν πολύ και μειώνουν την απόδοση της αντίστοιχης βάσης. Στη δική μας περίπτωση, το γεγονός ότι η βάση χρησιμοποιείται για τα δεδομένα πολλαπλών tenants κάνει το πρόβλημα εντονότερο αφού θα μειώνεται η απόδοση για όλους τους tenants. Μια καλύτερη προσέγγιση θα ήταν η διαγραφή απλά των μεταδεδομένων που αφορούν τον πίνακα, και η αναβολή της διαγραφής των εγγραφών του σε χρόνο που η χρήση της βάσης θα είναι χαμηλή(π.χ. από κάποιο daemon process).

Πράγματι, από τη στιγμή που διαγράφονται τα αντίστοιχα μεταδεδομένα, ο πίνακας δε θα φαίνεται πλέον στο σχήμα του tenant και τα περιεχόμενα του δε θα είναι προσβάσιμα. Έτσι, η διαγραφή των εγγραφών του δεν είναι άμεσα απαραίτητη.

6.2.2.4 Διαγραφή πεδίου

Ένας tenant μπορεί να διαγράψει μόνο τα ιδιωτικά του πεδία, δηλαδή είτε τα πεδία-επεκτάσεις σε ένα βασικό πίνακα, είτε τα πεδία ενός ιδιωτικού του πίνακα. Η μέθοδος που πραγματοποιεί τη διαγραφή αυτή είναι η μέθοδος dropCustomField(String fieldname) της κλάσης Table. Με τη μέθοδο αυτή σβήνουμε και το αντίστοιχο CustomField από τη λίστα που έχει το Table με τα πεδία του, αλλά και την αντίστοιχη εγγραφή στον πίνακα μεταδεδομένων "fields"(με χρήση μεθόδου delete(customField) της κλάσης CustomFieldDAO).

Αν το πεδίο που σβήνουμε είναι unique, θα πρέπει να σβήσουμε και τις εγγραφές που το αφορούν στον αντίστοιχο πίνακα μοναδικότητας με ένα ερώτημα της παρακάτω μορφής:

DELETE FROM unique_<datatype> WHERE custom_field_id = #id;

όπου #id είναι το αναγνωριστικό του συγκεκριμένου ιδιωτικού πεδίου. Όπως και στην περίπτωση της διαγραφής ενός πίνακα, τα δεδομένα αυτά δεν είναι απαραίτητο να σβηστούν άμεσα, οπότε θα μπορούσαν να διαγραφούν με αντίστοιχη διαδικασία όταν η χρήση του συστήματος είναι χαμηλή(μέσω του ίδιου daemon process).

7

Έλεγχος

Στο κεφάλαιο αυτό, έχοντας αναπτύξει το σύστημα, πραγματοποιούμε τον έλεγχο του και παρουσιάζουμε και εποπτικά τις λειτουργίες του.

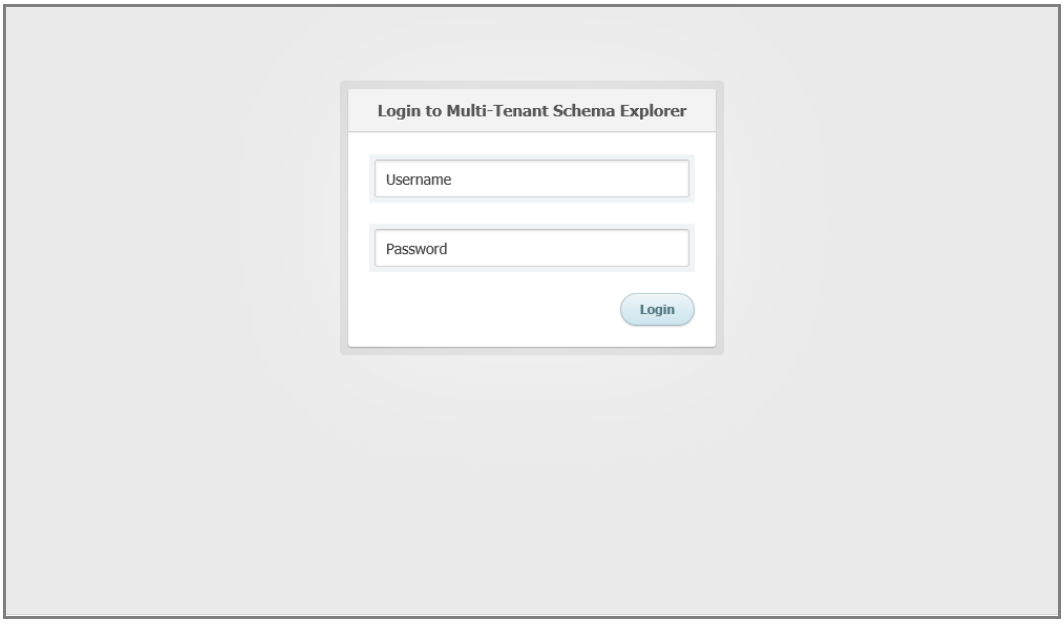
7.1 Μεθοδολογία ελέγχου

Για τον έλεγχο του συστήματος αναπτύξαμε μια απλή διεπαφή ιστού με την οποία ένας tenant μπορεί να συνδεθεί, να δει το σχήμα του, να το τροποποιήσει αλλά και να πραγματοποιήσει ερωτήματα σε αυτό. Για την ανάπτυξη της διεπαφής χρησιμοποιήσαμε τη γλώσσα προγραμματισμού Java, καθώς και την τεχνολογία Java Server Pages(JSP). Ως web server χρησιμοποιήσαμε τον Apache Tomcat v7.0.

Ο έλεγχος θα γίνει με την εκτέλεση ενός σεναρίου το οποίο θα περιλαμβάνει όλες τις ενέργειες τροποποίησης και όλα τα ερωτήματα που μπορεί να εκτελέσει ένας tenant. Έμφαση θα δοθεί στον έλεγχο της εξασφάλισης περιορισμών (π.χ. αναφορικής ακεραιότητας, μοναδικότητας).

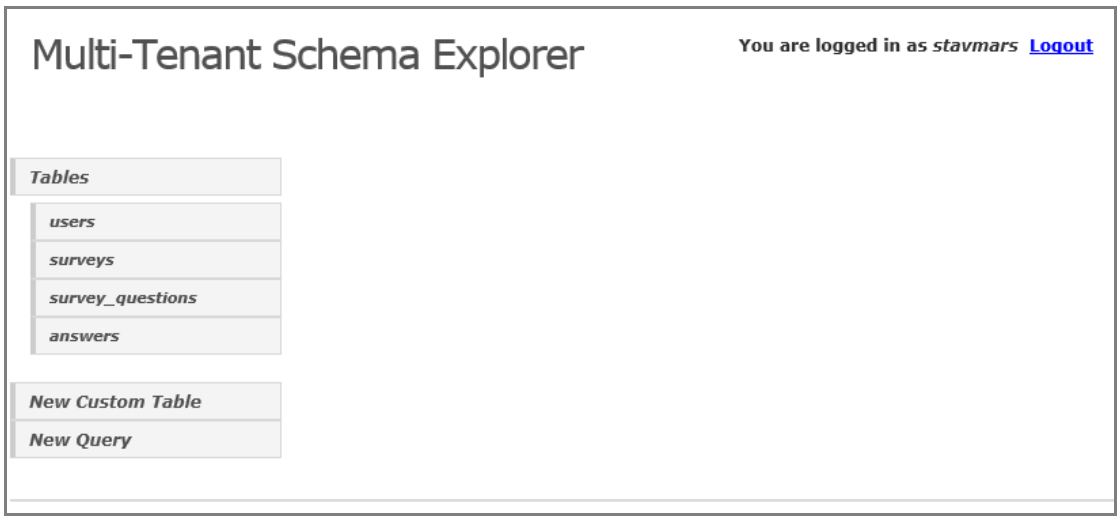
7.2 Αναλυτική παρουσίαση ελέγχου

Στην ενότητα αυτή παρουσιάζουμε αναλυτικά τον έλεγχο του συστήματος. Στην Εικόνα 7-1 βλέπουμε την οθόνη σύνδεσης. Εδώ, ένας tenant εισάγει τα στοιχεία σύδεσης του στο σύστημα.



Εικόνα 7-1

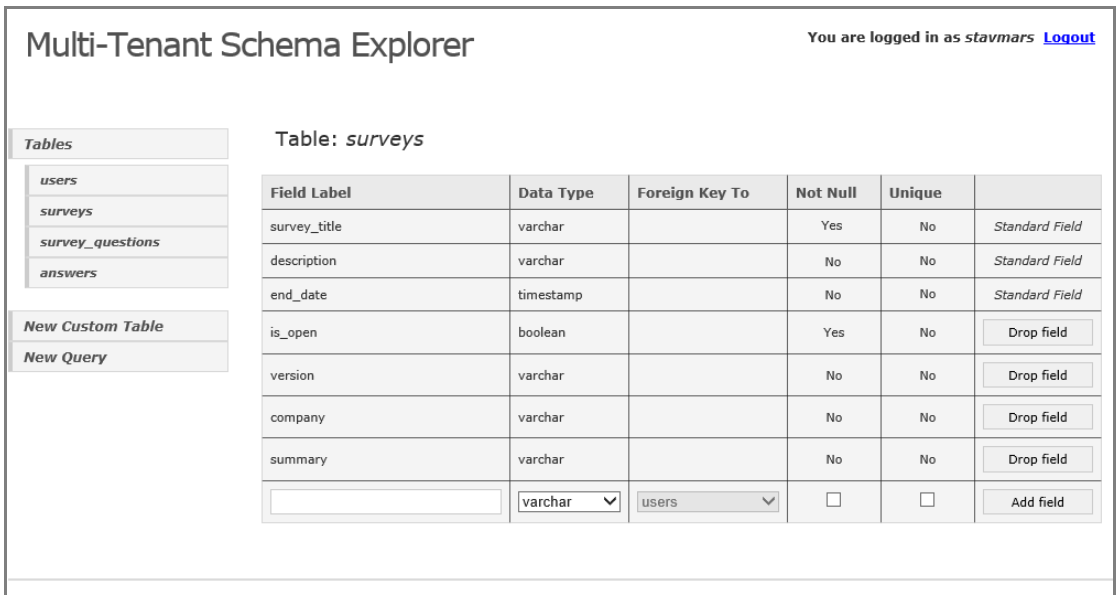
Αν τα στοιχεία που εισήγαγε είναι ορθά, τότε μεταφέρεται στην κεντρική σελίδα, που φαίνεται στην Εικόνα 7-2.



Εικόνα 7-2

Στη σελίδα αυτή βλέπουμε στα αριστερά τους πίνακες που έχει στο σχήμα του ο συγκεκριμένος tenant. Επίσης, βλέπουμε επιλογές για δημιουργία νέου πίνακα, αλλά και για εκτέλεση ενός νέου ερωτήματος.

Πατώντας στον πίνακα "surveys", οδηγούμαστε στην οθόνη που φαίνεται στην εικόνα 7-3.

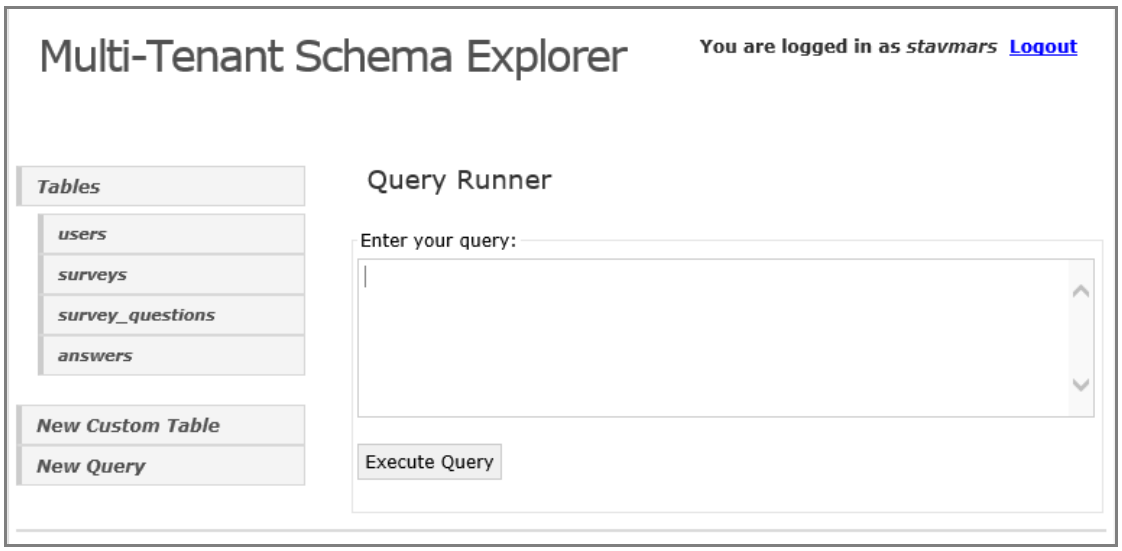


Εικόνα 7-3

Στην οθόνη αυτή βλέπουμε ένα πίνακα όπου παρουσιάζονται οι στήλες της σχέσης "surveys". Η σχέση αυτή είναι βασική, οπότε περιέχει και βασικά πεδία, κοινά για όλους τους tenants. Τα πεδία αυτά φαίνονται από την ένδειξη "Standard Field" στην τελευταία στήλη του πίνακα, και σε αντίθεση με τα πεδία-επεκτάσεις ενός tenant, δε μπορούν να διαγραφούν. Πράγματι, μόνο στα πεδία-επεκτάσεις υπάρχει η επιλογή "Drop Field".

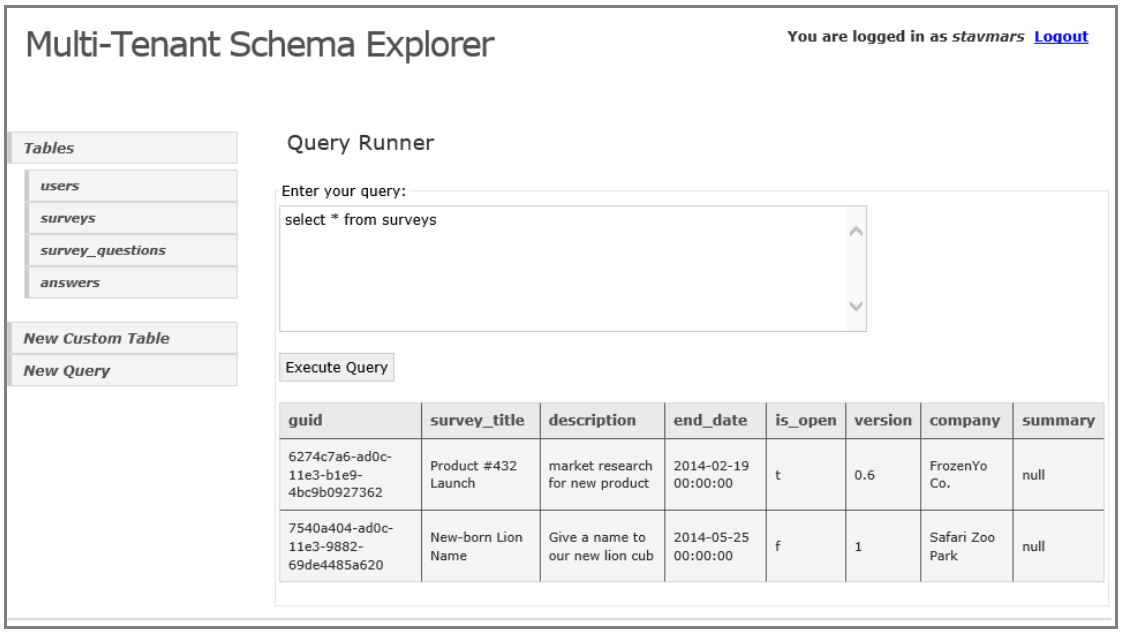
Για κάθε πεδίο βλέπουμε τον τύπο δεδομένων του, περιορισμούς τύπου not-null ή μοναδικότητας, καθώς και σε ποιον πίνακα δείχνουν σε περίπτωση που περιέχουν ένα ξένο κλειδί(δηλαδή είναι τύπου relationship).

Πατώντας στην επιλογή "New Query", μεταφερόμαστε στην οθόνη που φαίνεται στην επόμενη εικόνα.



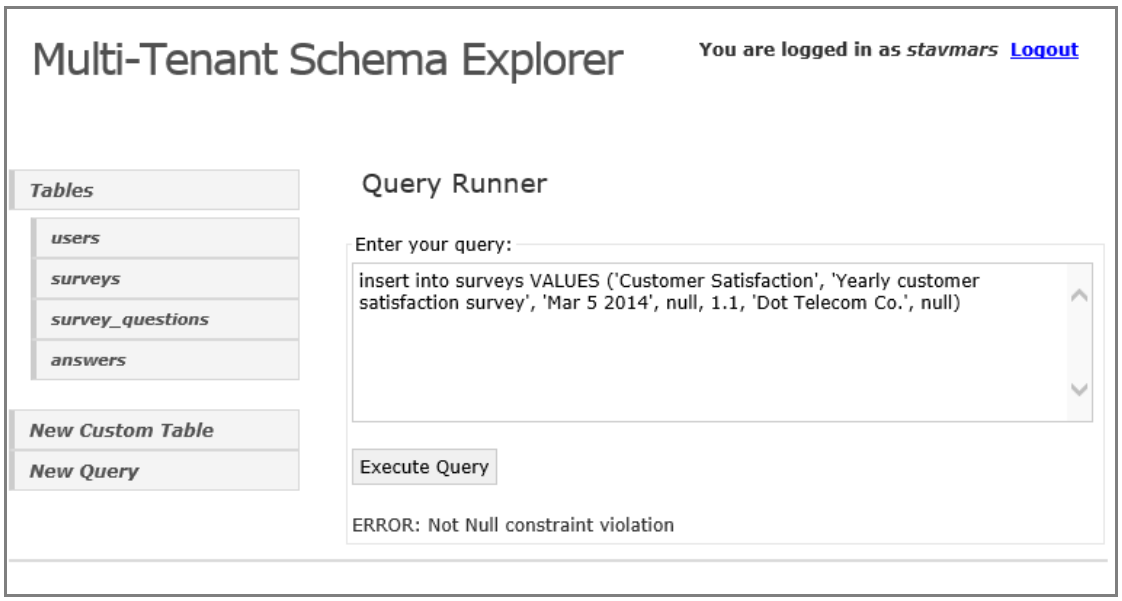
Εικόνα 7-4

Εδώ, μπορούμε να εισάγουμε ένα ερώτημα(select, insert, delete ή update) στο λογικό σχήμα του tenant και να δούμε τα αποτελέσματα. Εισάγοντας ένα ερώτημα που μας επιστρέφει όλες τις εγγραφές του πίνακα "surveys", μεταφερόμαστε στην παρακάτω οθόνη:



Εικόνα 7-5

Παρατηρούμε ότι ο πίνακας “surveys” έχει δύο εγγραφές. Ας εκτελέσουμε τώρα ένα ερώτημα εισαγωγής στον πίνακα αυτό.



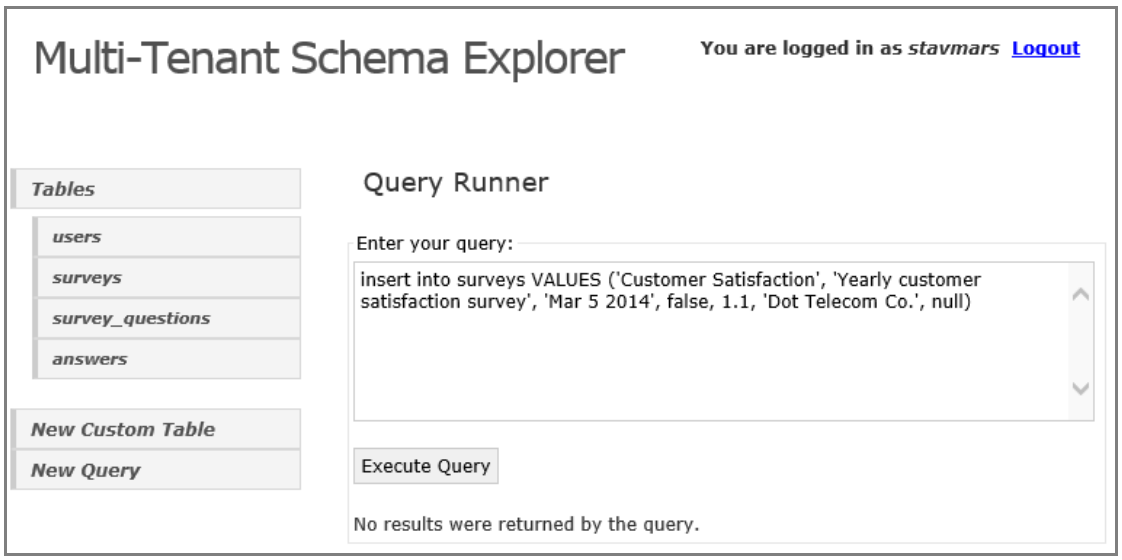
Εικόνα 7-6

Παρατηρούμε ότι η εισαγωγή δεν ολοκληρώθηκε με επιτυχία καθώς παραβίαζε περιορισμό τύπου not-null. Αν δούμε στην Εικόνα 7-4, το πεδίο-επέκταση “is_open” έχει οριστεί ως not-null. Ο έλεγχος αυτός έγινε με χρήση της stored procedure “check_null”, όπως είδαμε σε προηγούμενο κεφάλαιο.

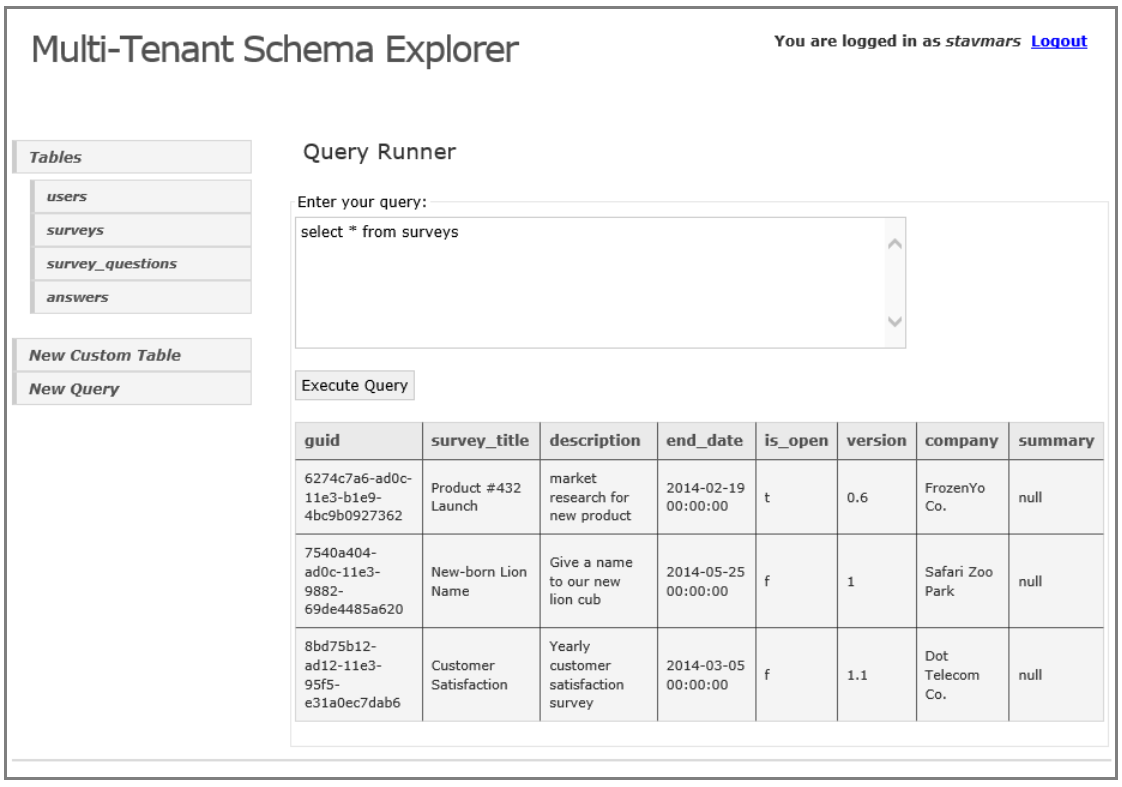
Αν επαναλάβουμε την ίδια εισαγωγή δίνοντας τιμή στο πεδίο “is_open”, μεταφερόμαστε στην οθόνη που φαίνεται στην Εικόνα 7-7.

Βλέπουμε ότι μετά την εκτέλεση εμφανίζεται μήνυμα που μας ενημερώνει ότι δεν επιστράφηκαν αποτελέσματα από το συγκεκριμένο ερώτημα μιας και δεν είναι ερώτημα ανάκτησης δεδομένων.

Αν τώρα εκτελέσουμε εκ νέου το ερώτημα ανάκτησης των εγγραφών του πίνακα “surveys”, μας εμφανίζεται η οθόνη που φαίνεται στην Εικόνα 7-8.

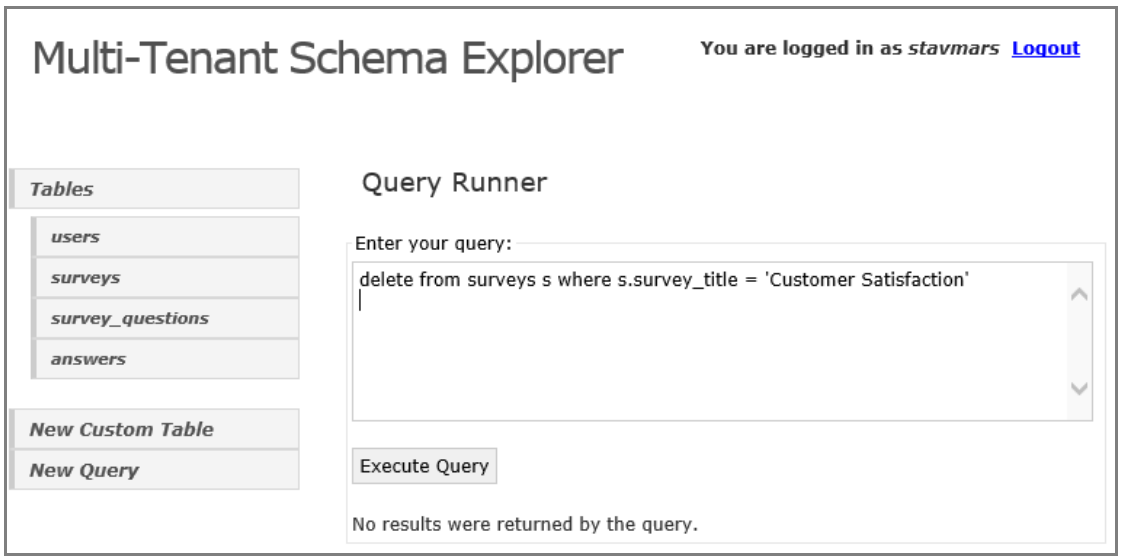


Εικόνα 7-7



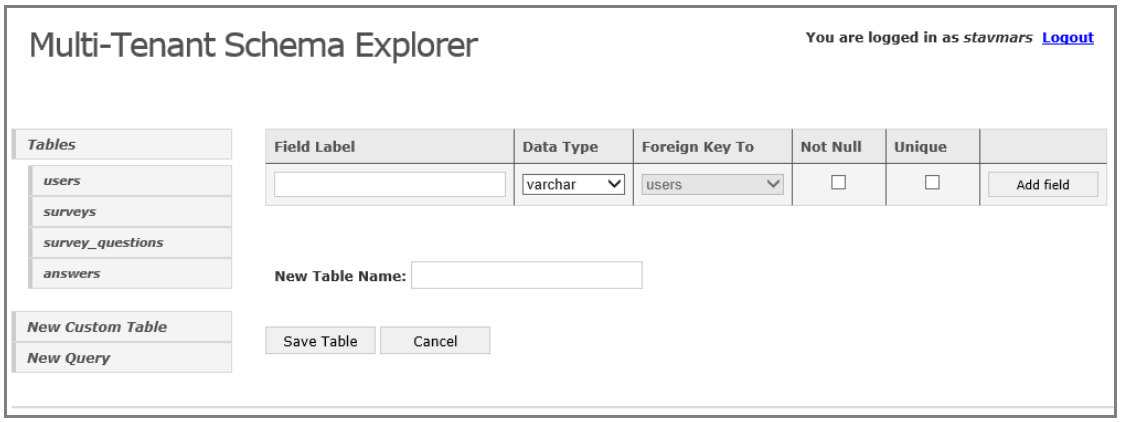
Εικόνα 7-8

Αν θελήσουμε να σβήσουμε την εγγραφή που μόλις προσθέσαμε, μπορούμε να εκτελέσουμε το ερώτημα που φαίνεται στην Εικόνα 7-9.



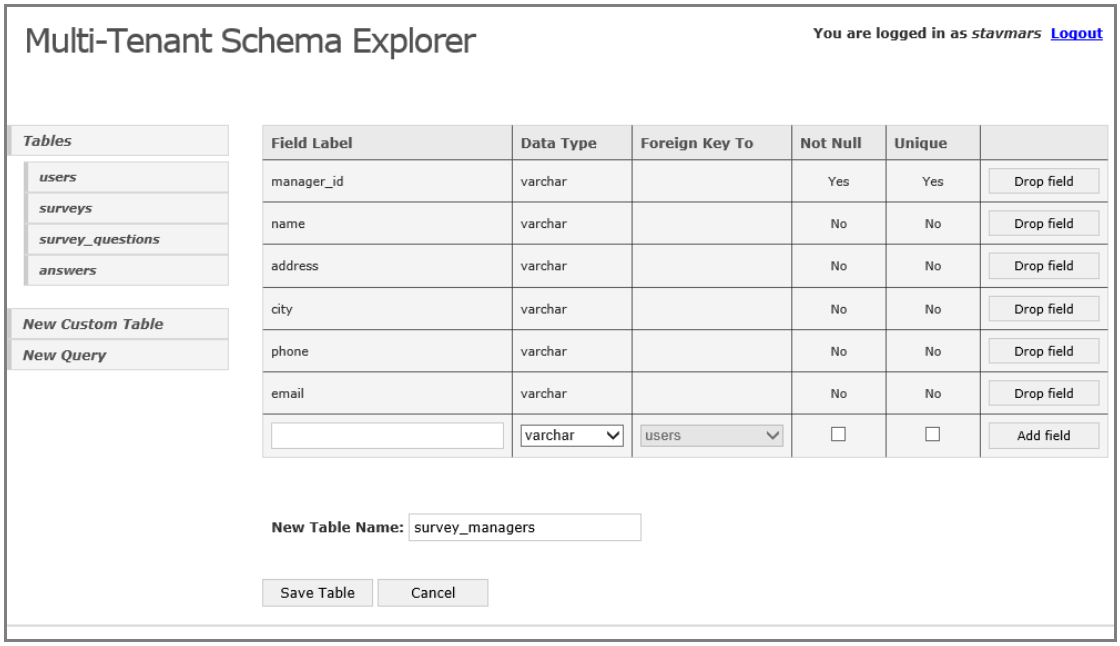
Εικόνα 7-9

Έστω ότι ο συγκεκριμένος tenant θέλει να μπορεί να αποθηκεύσει για κάθε έρευνα και τον υπεύθυνο της. Για να το κάνουμε αυτό, πρέπει κατ'αρχάς να δημιουργήσουμε έναν πίνακα που θα περιέχει τα στοιχεία των διαχειριστών ερευνών. Πατώντας στην επιλογή "New Custom Table" μεταβαίνουμε στην παρακάτω οθόνη:



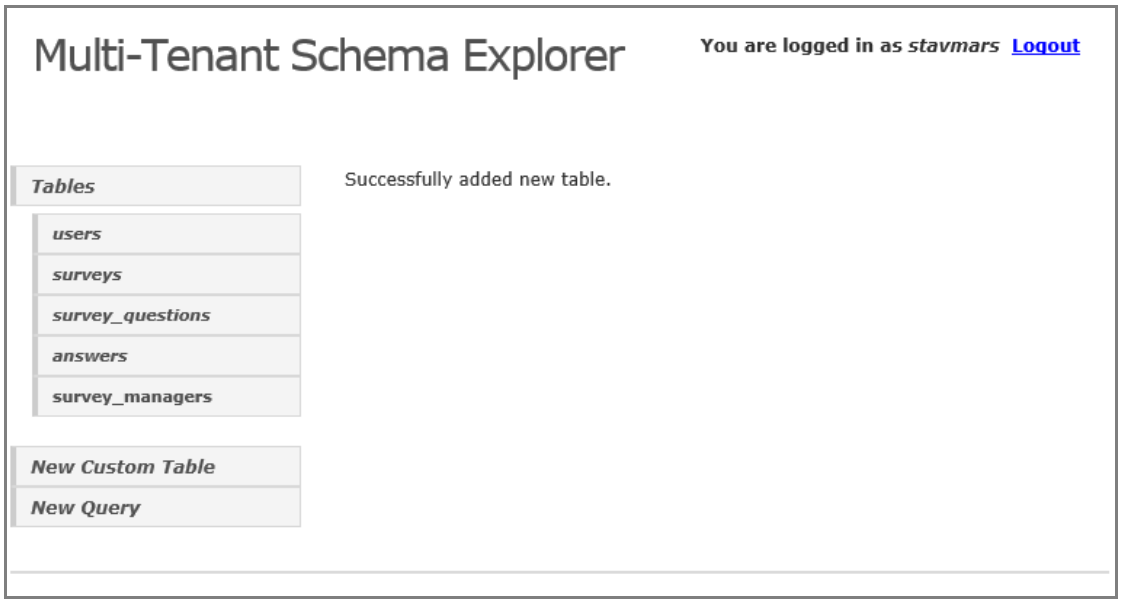
Εικόνα 7-10

Στην οθόνη αυτή μπορούμε να προσθέσουμε τα πεδία του νέου πίνακα, καθώς και το όνομα του. Αφού, λοιπόν, ολοκληρώσουμε τα παραπάνω καταλήγουμε στην επόμενη οθόνη:



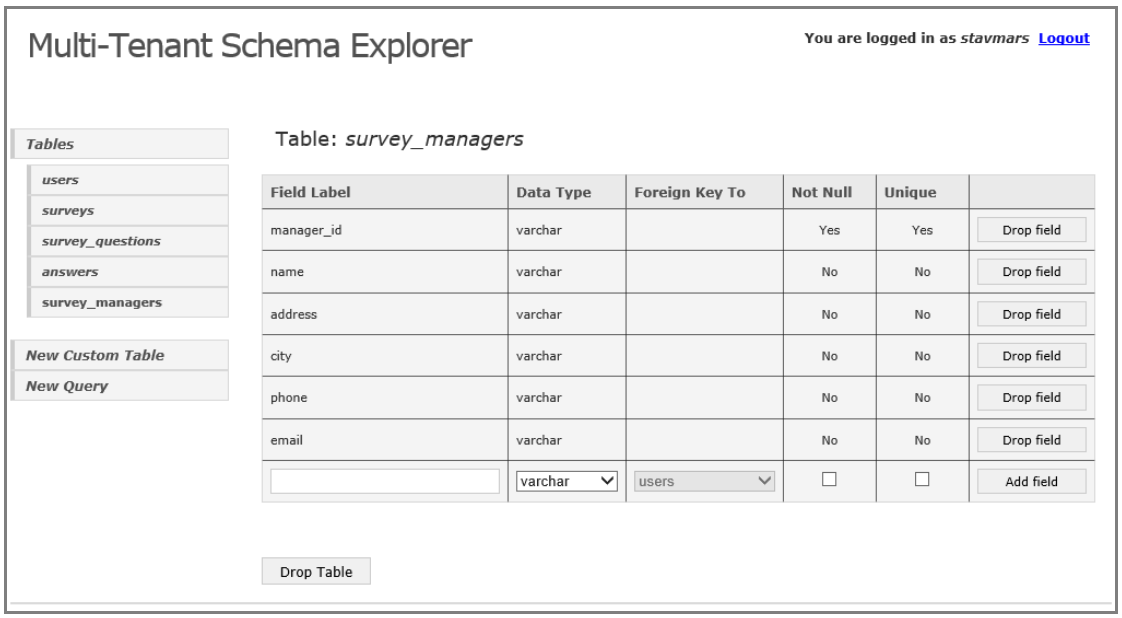
Εικόνα 7-11

Πατώντας τώρα την επιλογή “Save Table”, δημιουργείται ο νέος ιδιωτικός πίνακας και εμφανίζεται στη λίστα με τους πίνακες στα αριστερά της σελίδας. Ο νέος πίνακας δημιουργείται όπως δείξαμε με τροποποίηση μόνο σε πίνακες μεταδεδομένων και οι εγγραφές του θα αποθηκεύονται αποκλειστικά στο Chunk Table.



Εικόνα 7-12

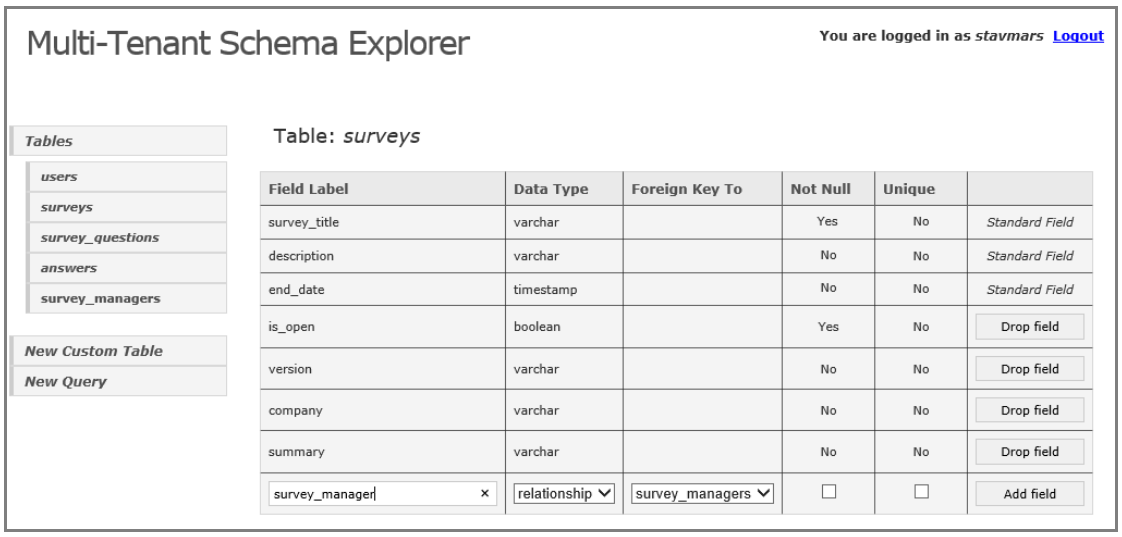
Σε αντίθεση με τους βασικούς πίνακες, ο νέος πίνακας δεν εμφανίζεται με πλάγια γραφή στη λίστα. Αν πατήσουμε πάνω στο νέο πίνακα μεταβαίνουμε στην παρακάτω οθόνη:



Εικόνα 7-13

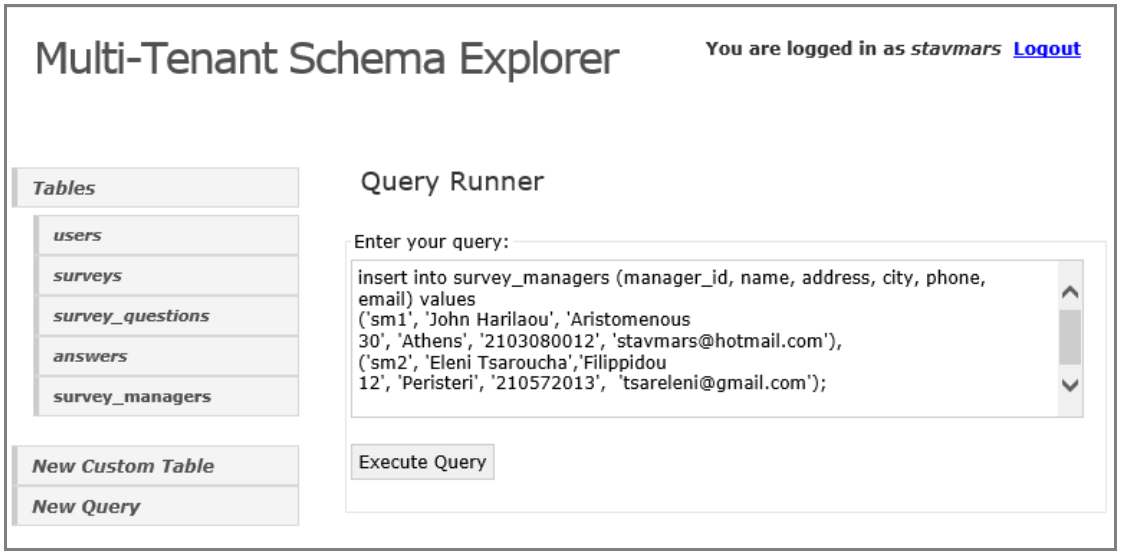
Παρατηρούμε, ότι σε αντιδιαστολή με την αντίστοιχη οθόνη για τον πίνακα "surveys", εδώ εμφανίζεται η επιλογή "Drop Table" για διαγραφή ενός ιδιωτικού πίνακα.

Για να μπορούμε να συσχετίσουμε ένα survey manager με μια έρευνα πρέπει να προσθέσουμε στον πίνακα "surveys" ένα πεδίο τύπου relationship που θα δείχνει στον πίνακα "survey_managers", όπως φαίνεται στην Εικόνα 7-14. Πατώντας την επιλογή "Add Field", αποθηκεύουμε το νέο πεδίο. Ο συνδεδεμένος tenant δεν αντιλαμβάνεται το ότι για το νέο πεδίο δε δημιουργείται κάποιο νέο φυσικό πεδίο αλλά απλά μια αντιστοίχιση με ένα ελεύθερο γενικό πεδίο(preallocated field) ή με μια στήλη ενός chunk.

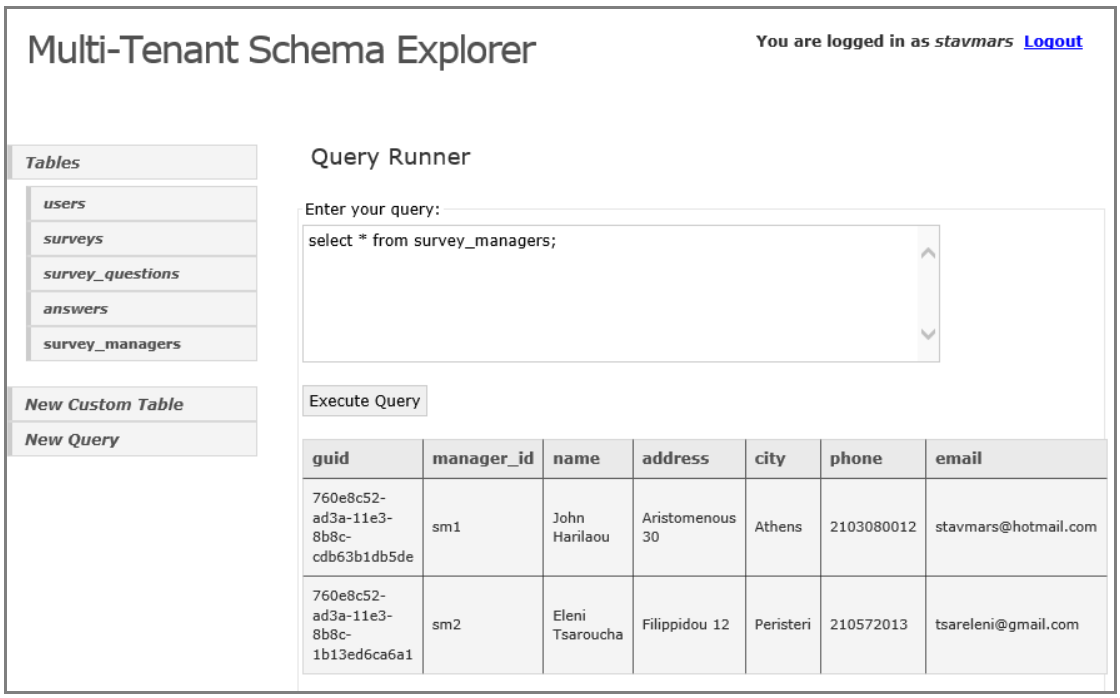


Εικόνα 7-14

Αν προσθέσουμε δυο διαχειριστές ερευνών, όπως φαίνεται στην Εικόνα 7-15, βλέπουμε ότι αποθηκεύονται κανονικά και εμφανίζονται με το ερώτημα ανάκτησης που εκτελούμε στην Εικόνα 7-16. Τα δεδομένα αυτά επαναλαμβάνουμε ότι αποθηκεύονται αποκλειστικά στο Chunk Table.



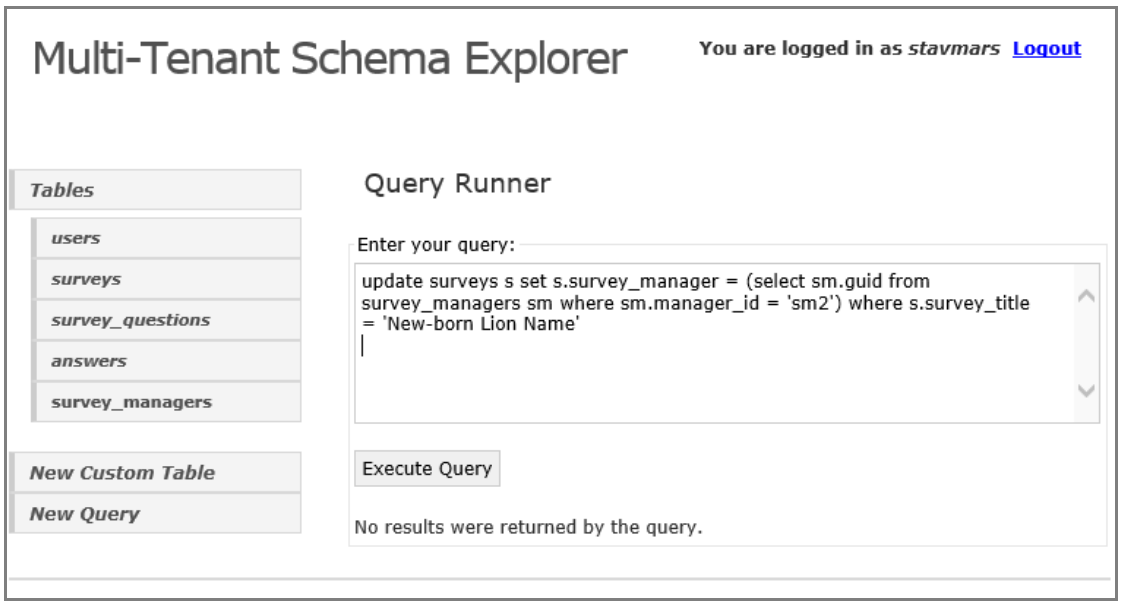
Εικόνα 7-15



Εικόνα 7-16

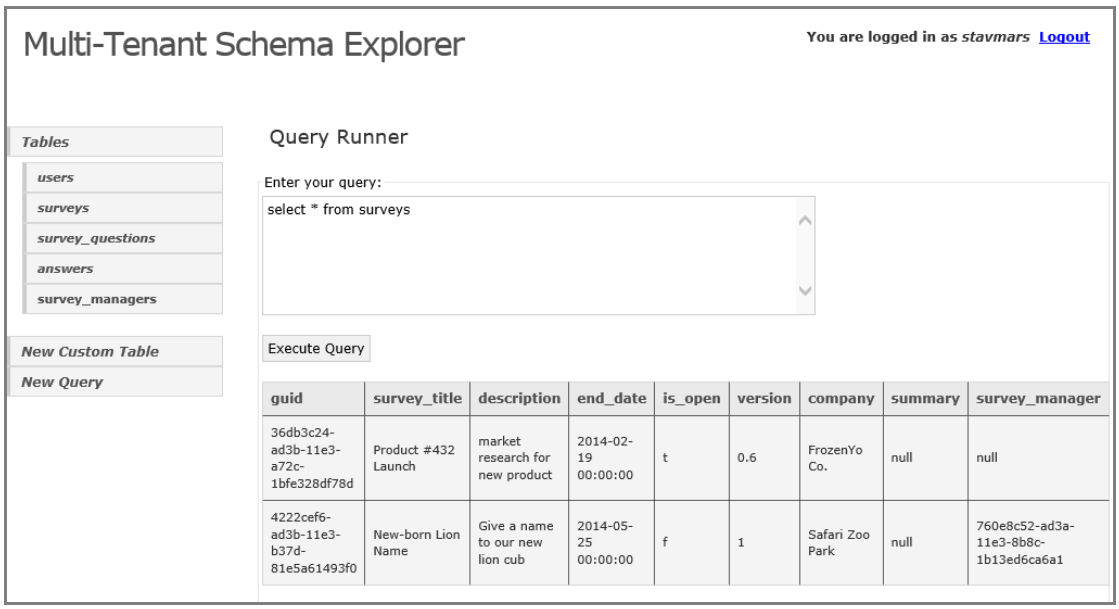
Για να ορίσουμε έναν υπεύθυνο σε μία έρευνα πρέπει να θέσουμε στο πεδίο "survey_manager" της εγγραφής της έρευνας αυτής το αναγνωριστικό GUID του αντίστοιχου υπεύθυνου. Η χρήση των GUIDs ως πρωτεύοντα κλειδιά, και συνεπώς και ως ξένα κλειδιά, δυσχεραίνει τη δημιουργία ερωτημάτων προς τη βάση. Αυτό συμβαίνει γιατί τα αναγνωριστικά αυτά δεν είναι ευανάγνωστα για έναν άνθρωπο. Παρ'όλα αυτά, οι multi-tenant βάσεις χρησιμοποιούνται, ως επί το πλείστον, στα πλαίσια multi-tenant εφαρμογών. Οι εφαρμογές αυτές χρησιμοποιούνται μέσω κάποιας γραφικής διεπαφής που αποκρύπτει τα πραγματικά ερωτήματα που εκτελούνται στη βάση. Τα ερωτήματα αυτά απασχολούν συνήθως μόνο τους σχεδιαστές της εφαρμογής.

Ένα ερώτημα το οποίο ορίζει έναν υπεύθυνο σε μια έρευνα χωρίς να χρειάζεται εκ των προτέρων την τιμή του αναγνωριστικού GUID, φαίνεται στην Εικόνα 7-17.



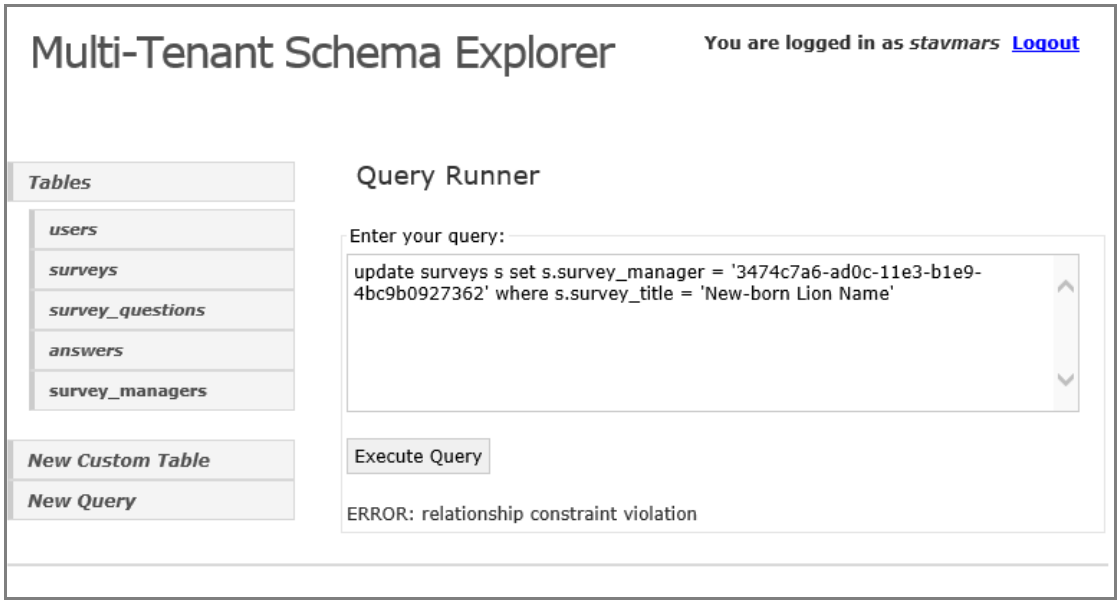
Εικόνα 7-17

Αν δούμε τώρα τα περιεχόμενα του πίνακα "surveys" βλέπουμε ότι για την έρευνα με τίτλο "New-born Lion Name" στη στήλη "survey_manager" υπάρχει πλέον τιμή.



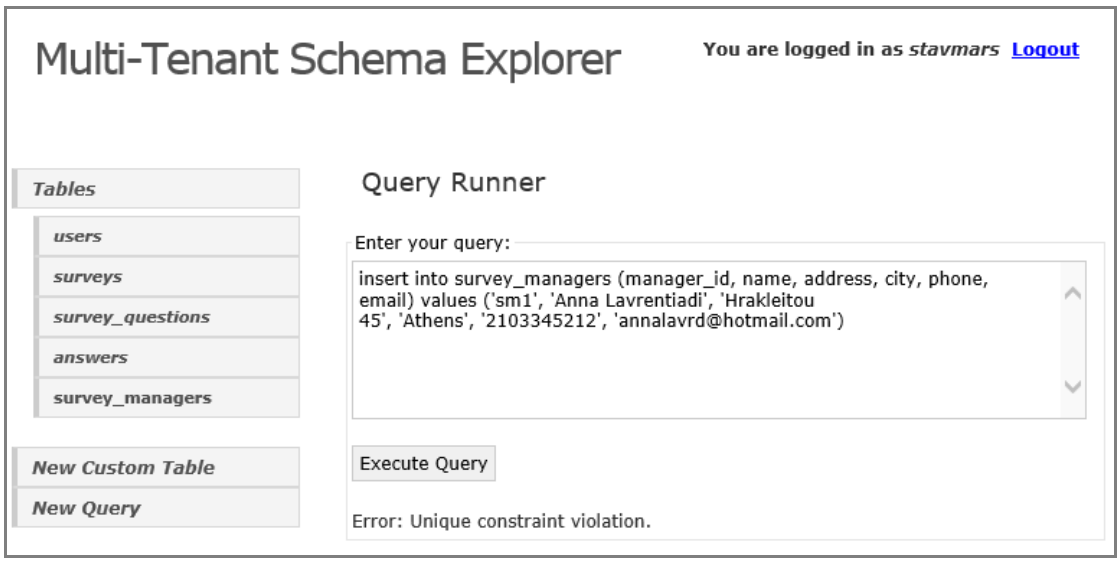
Εικόνα 7-18

Αν προσπαθήσουμε να δώσουμε στη στήλη “survey manager” την τιμή ενός αναγνωριστικού που δεν αντιστοιχεί σε εγγραφή του λογικού πίνακα “survey_managers” του συγκεκριμένου tenant, τότε θα λάβουμε μήνυμα σφάλματος, όπως φαίνεται στην Εικόνα 7-19.



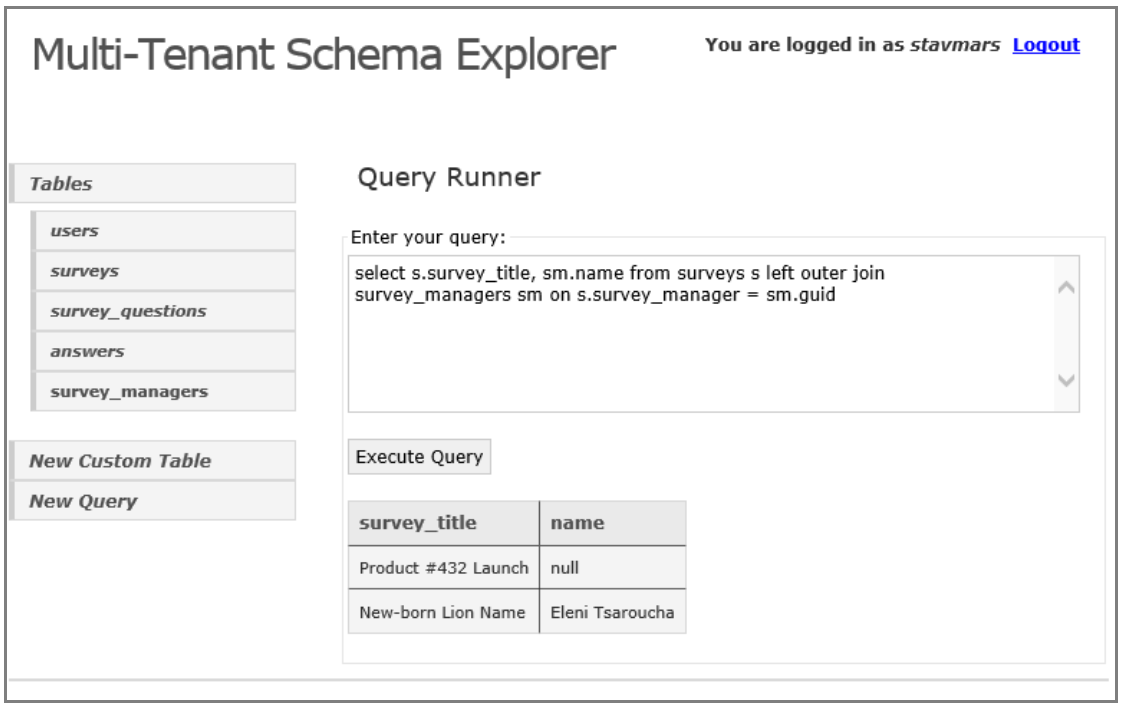
Εικόνα 7-19

Προσπαθώντας να εισάγουμε μια νέα εγγραφή στον πίνακα “survey_managers” με τιμή για τη στήλη “manager_id” ίδια με αυτή ήδη υπάρχουσας εγγραφής, οδηγούμαστε σε σφάλμα και η εισαγωγή απορρίπτεται(Εικόνα 7-20). Αυτό γίνεται γιατί το πεδίο “manager_id” έχει οριστεί με περιορισμό μοναδικότητας(unique constraint). Υπενθυμίζουμε ότι για το έλεγχο τέτοιων περιορισμών σε ιδιωτικά πεδία ενός tenant χρησιμοποιείται ένας από τους πίνακες τύπου “unique_datatype”. Στη συγκεκριμένη περίπτωση, χρησιμοποιείται ο πίνακας “unique_varchar”.



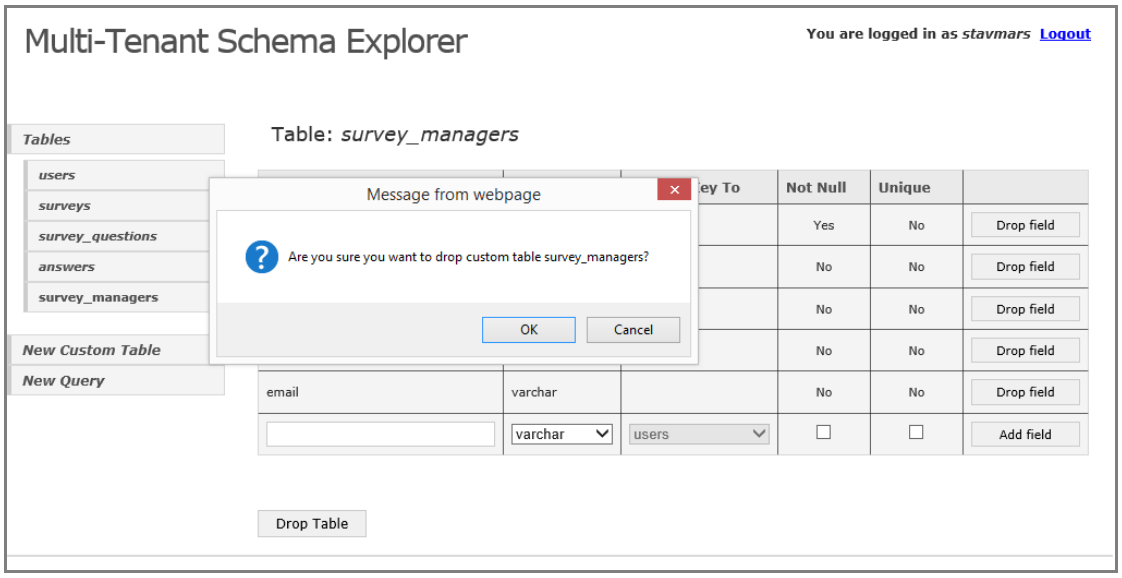
Εικόνα 7-20

Αν τώρα εκτελέσουμε έναν εξωτερικό σύνδεσμο έτσι ώστε να δούμε τον τίτλο κάθε έρευνας και το όνομα του υπεύθυνου της(αν υπάρχει), παίρνουμε τα αποτελέσματα που φαίνονται στην Εικόνα 7-21.



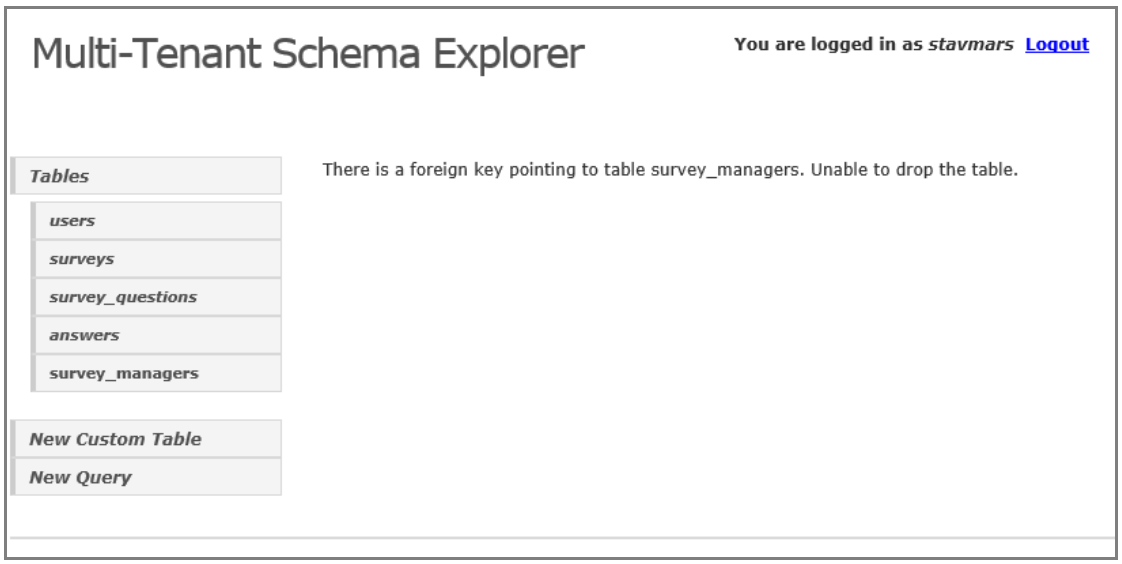
Εικόνα 7-21

Αν προσπαθήσουμε να σβήσουμε τον πίνακα “survey_managers”, πατώντας την επιλογή “Drop Table”, μας έρχεται μήνυμα επιβεβαίωσης της διαγραφής(Εικ. 7-22).



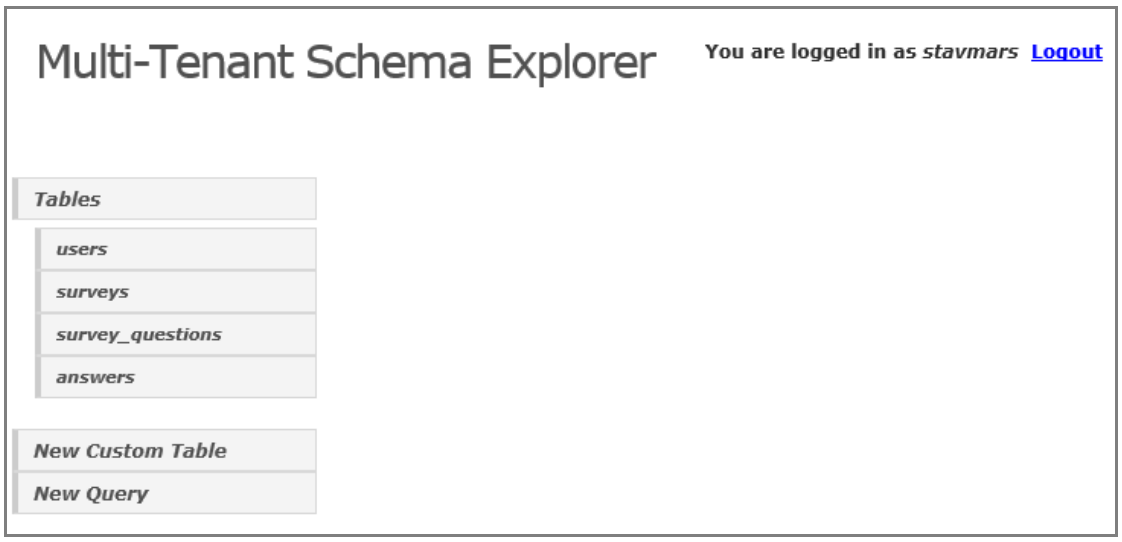
Εικόνα 7-22

Πατώντας “OK” για να προχωρήσουμε με τη διαγραφή βλέπουμε στην Εικόνα 7-23 ότι αυτή δε μπορεί να πραγματοποιηθεί, καθώς υπάρχει ένα ξένο κλειδί προς τον πίνακα αυτό.



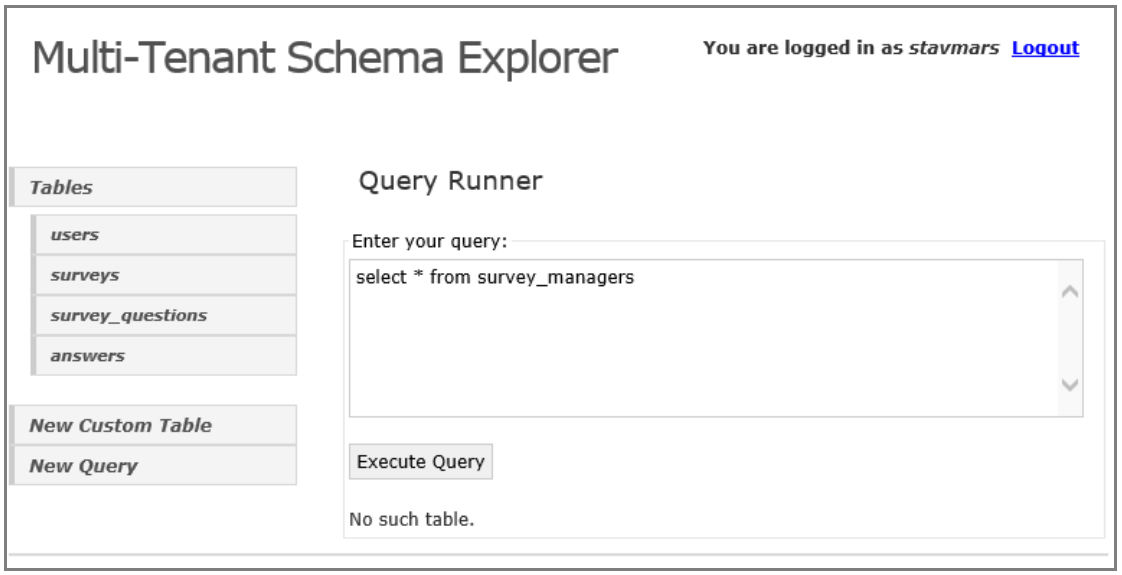
Εικόνα 7-23

Για να μπορέσουμε να διαγράψουμε τον πίνακα πρέπει να διαγράψουμε πρώτα το πεδίο "survey_manager" του πίνακα "surveys". Ύστερα, μπορούμε να επανεπιλέξουμε την επιλογή "Drop Table" στην οθόνη που αφορά τον πίνακα "survey_managers", και αφού επιβεβαιώσουμε τη διαγραφή, οδηγούμαστε στην οθόνη της Εικόνας 7-24.



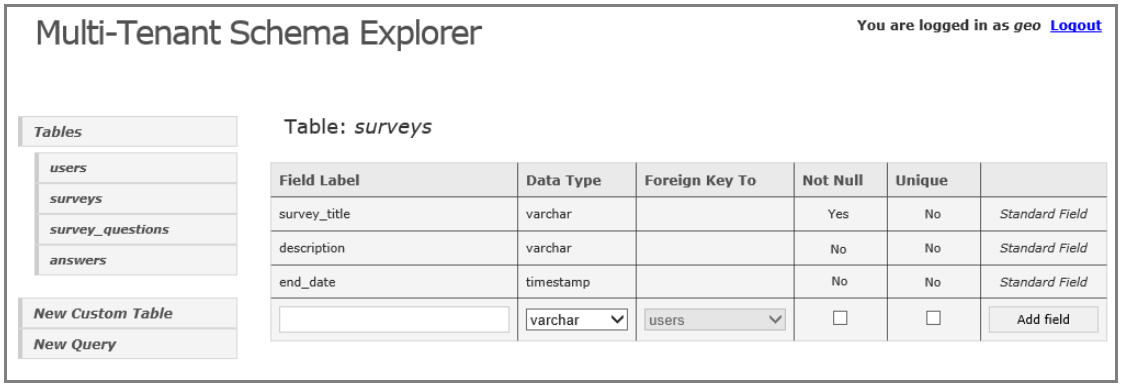
Εικόνα 7-24

Παρατηρούμε ότι ο πίνακας δεν υπάρχει πλέον στη λίστα στα αριστερά της οθόνης. Επίσης, αν εκτελέσουμε το ερώτημα που φαίνεται στην Εικόνα 7-25, λαμβάνουμε μήνυμα ότι δεν υπάρχει τέτοιος πίνακας.



Εικόνα 7-25

Αν τώρα αποσυνδεθούμε και συνδεθούμε ως κάποιος άλλος tenant, θα δούμε ότι τα αν και οι δυο tenants μοιράζονται τους ίδιους βασικούς πίνακες, δε μοιράζονται τις ίδιες επεκτάσεις και τα ίδια δεδομένα. Συγκεκριμένα, ο tenant που συνδέθηκε όπως φαίνεται στην παρακάτω εικόνα, δεν έχει τροποποιήσει το σχήμα και έχει μόνο τους βασικούς πίνακες και τα βασικά πεδία αυτών.



Εικόνα 7-26

8

Επίλογος

Στο τελευταίο κεφάλαιο, συνοψίζεται η παρούσα διπλωματική εργασία και τα αποτελέσματα της. Τέλος, αναφέρονται πιθανές μελλοντικές επεκτάσεις για το σύστημα που υλοποιήσαμε που θα μπορούσαν να διευρύνουν τις λειτουργίες του και να βελτιώσουν την απόδοση του.

8.1 Σύνοψη και συμπεράσματα

Στην παρούσα διπλωματική εργασία ξεκινήσαμε εξετάζοντας τις multi-tenant αρχιτεκτονικές που χρησιμοποιούνται από τις SaaS εφαρμογές. Έπειτα, εστιάζοντας στο επίπεδο των δεδομένων των εφαρμογών αυτών, παρουσιάσαμε τα είδη των multi-tenant βάσεων δεδομένων και είδαμε ότι από τις βασικές προσεγγίσεις που χρησιμοποιούνται η προσέγγιση διαμοιραζόμενου σχήματος μειώνει περισσότερο το κόστος και αυξάνει την αξιοποίηση των υπολογιστικών πόρων. Εντούτοις, η προσέγγιση αυτή απαιτεί σημαντικότερη προγραμματιστική προσπάθεια κατά την ανάπτυξη της εφαρμογής, ώστε να μπορεί το λογικό σχήμα κάθε tenant(που ενδέχεται να διαφοροποιείται από των υπολοίπων) να αντιστοιχίζεται στους κοινούς πίνακες της βάσης δεδομένων.

Στη συνέχεια, αφού επικεντρώσαμε το ενδιαφέρον μας στην προσέγγιση διαμοιραζόμενου σχήματος, εξετάσαμε τις τεχνικές που χρησιμοποιούνται για την ανάπτυξη multi-tenant βάσεων με αυτή. Συνδυάζοντας την τεχνική των preallocated fields(γενικά πεδία) με μια

παραλλαγή της τεχνικής των Chunk Tables, διατυπώσαμε μια τεχνική που προσπαθεί να αντιμετωπίσει το πρόβλημα της αντιστοίχισης των δεδομένων πολλαπλών tenants σε κοινές φυσικές δομές μιας σχεσιακής βάσης δεδομένων. Η τεχνική αυτή υποστηρίζει την εξατομίκευση του σχήματος ενός tenant, προσπαθώντας παράλληλα να μεγιστοποιήσει τον αριθμό των tenants που μοιράζονται την ίδια βάση και τελικά τον ίδιο εξυπηρετητή. Επίσης, προσπαθεί να αξιοποιήσει σημαντικά (τουλάχιστον στους κοινούς πίνακες και στα κοινά πεδία) τις υπηρεσίες που προσφέρει ένα ΣΔΒΔ για δημιουργία ευρετηρίων, έλεγχο περιορισμών κ.λ.π.

Τέλος, αναπτύξαμε και αναλύσαμε ένα σύστημα που χρησιμοποιώντας την τεχνική που διατυπώσαμε, προσφέρει τη δυνατότητα υποστήριξης multi-tenant βάσεων από ένα σχεσιακό ΣΔΒΔ.

8.2 Μελλοντικές επεκτάσεις

8.2.1 Υποστήριξη ευρετηρίων από ιδιωτικά πεδία

Είδαμε ότι στα κοινά πεδία ενός βασικού πίνακα μπορούμε να ορίσουμε ευρετήρια και να βελτιώσουμε την εκτέλεση των ερωτημάτων. Εντούτοις, στα πεδία-επεκτάσεις ενός βασικού πίνακα, ή στα πεδία ενός ιδιωτικού πίνακα, κάτι τέτοιο είναι αδύνατο. Πράγματι, τα πεδία αυτά αποθηκεύονται σε γενικές στήλες τις οποίες μπορούν να μοιράζονται με άλλα πεδία, πιθανώς διαφορετικού πίνακα, ή και διαφορετικού tenant. Έτσι, για δυο λογικά πεδία που μοιράζονται την ίδια γενική στήλη, μπορεί να επιθυμούμε στο ένα ευρετήριο, ενώ στο άλλο όχι. Επιπλέον, οι τιμές στις γενικές αυτές στήλες αποθηκεύονται ως varchar, ό,τι τύπου δεδομένων και να είναι τα πεδία τους.

Η δυνατότητα δημιουργίας ευρετηρίων μπορεί όμως να κρίνεται σημαντική για κάποια ιδιωτικά πεδία. Έτσι, μια πολύ ενδιαφέρουσα μελλοντική επέκταση του συστήματος θα ήταν η προσθήκη τέτοιας δυνατότητας. Μια ιδέα για το πως θα μπορούσαμε να υλοποιήσουμε ένα τέτοιο μηχανισμό, μπορούμε να πάρουμε από την πλατφόρμα της salesforce.com. Όπως αναφέραμε και στην παράγραφο 2.3, και η πλατφόρμα αυτή δε μπορεί να χρησιμοποιήσει το μηχανισμό δημιουργίας ευρετηρίων του ΣΔΒΔ, μιας και τελικά οι τιμές για όλα τα πεδία αποθηκεύονται ως αλφαριθμητικά. Γι'αυτό, χρησιμοποιεί μια δομή πίνακα(index pivot table) στην οποία υπάρχει ένα πεδίο για κάθε υποστηριζόμενο τύπο δεδομένων, καθώς και αντίστοιχο ευρετήριο του ΣΔΒΔ για κάθε ένα από αυτά. Επίσης, υπάρχουν και κατάλληλα πεδία μεταδεδομένων. Όταν κάποιος tenant επιθυμεί να προσθέσει ένα ευρετήριο σε κάποιο πεδίο, για κάθε τιμή αυτού του πεδίου δημιουργείται μια εγγραφή στο index pivot table για

κατάλληλες τιμές στα πεδία μεταδεδομένων. Οι τιμές του πεδίου αυτού αποθηκεύονται στο πεδίο του κατάλληλου τύπου στο index pivot table. Όταν ένα ερώτημα περιέχει μια παράμετρο αναζήτησης που αναφέρει ένα λογικό πεδίο με ευρετήριο, ο βελτιστοποιητής ερωτημάτων της πλατφόρμας χρησιμοποιεί το index pivot table για να εκτελεστεί γρηγορότερα το ερώτημα. Περισσότερες λεπτομέρειες για αντίστοιχο μηχανισμό ευρετηρίων δίνονται και στο [18]. Τέτοια ευρετήρια προϋποθέτουν βέβαια και την ανάπτυξη ενός βελτιστοποιητή ερωτημάτων. Ο βελτιστοποιητής αυτός θα πρέπει να κρατάει στατιστικά για τις λογικές σχέσεις και να τα συμβουλεύεται για να κρίνει αν η χρήση ενός ευρετηρίου οδηγεί σε αποδοτικότερη εκτέλεση ενός ερωτήματος.

8.2.2 Αλλαγή παραμέτρων συστήματος και αναδιάταξη δεδομένων

Η τεχνική που επιλέξαμε για την αντιστοίχιση του λογικού σχήματος στο φυσικό, προϋποθέτει αρχικά την επιλογή του μεγέθους του Chunk Table, καθώς και του αριθμού των γενικών πεδίων κάθε βασικού πίνακα. Μετά τον αρχικό σχεδιασμό της βάσης και ενώ το σύστημα είναι ήδη σε λειτουργία, μπορεί να βρεθεί ότι αλλαγή των παραπάνω παραμέτρων βελτιώνουν ενδεχομένως τη απόδοση του. Αν και στην παρούσα υλοποίηση του συστήματος επιτρέπεται μόνο αύξηση των παραμέτρων αυτών, εντούτοις, αυτή η αύξηση δε συνοδεύεται από αντίστοιχη αναδιάταξη των δεδομένων στη βάση. Για παράδειγμα, αν και μπορούμε να αλλάξουμε τον αριθμό των γενικών πεδίων σε ένα βασικό πίνακα, δε θα προκύψει ανακατανομή των πεδίων-επεκτάσεων των tenants έτσι ώστε να εκμεταλλευούμε τα νέα γενικά πεδία και να γλιτώσουμε πιθανώς περιττά joins με το Chunk Table. Τα νέα γενικά πεδία θα χρησιμοποιηθούν μόνο από πεδία-επεκτάσεις που ορίζονται μετά τη δημιουργία τους. Επομένως, μια ακόμη ενδιαφέρουσα επέκταση θα ήταν η ανάπτυξη τέτοιου μηχανισμού για αναδιάταξη των δεδομένων μετά από αύξηση αλλά και από μείωση των παραμέτρων αυτών.

9

Βιβλιογραφία

[1] Final Version of NIST Cloud Computing Definition Published, [Online]. Available at: http://www.nist.gov/itl/csd/cloud-102511.cfm

[2] Software as a Service, [Online].
Available at: http://en.wikipedia.org/wiki/Software_as_a_service

[3] Multitenancy, [Online].
Available at: http://en.wikipedia.org/wiki/Multitenancy

[4] Cor-Paul Bezemer, Andy Zaidman. 2010. Multi-tenant SaaS applications: maintenance dream or nightmare?. In Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE) (IWPSE-EVOL '10). ACM

[5] M. Grund, M. Schapranow, J. Krueger, J. Schaffner and A. Bog: Shared Table Access Pattern Analysis for Multi-Tenant Applications , IEEE Symposium on Advanced Management of Information for Globalized
Enterprises, 2008, AMIGE, (2008).

[6] Frederick Chong, Gianpaolo Carraro, and Roger Wolter: Multi-Tenant Data Architecture, June 2006, [Online].
Available at http://msdn.microsoft.com/en-us/library/aa479086.aspx

[7] Dennis Howlett, "Multi-tenancy vs. single tenancy: Looking beyond TCO", The Software Engineering Research Group Technical Reports, Aug. 2010, [Online].
Available at:
http://www.zdnet.com/blog/howlett/multi-tenancy-vs-single-tenancy-looking-beyond-tco/2404

[8] Nati Shalom, "Multi-tenancy: does it have to be that hard?", The Software Engineering Research Group Technical Reports, Mar. 2010, [Online].
Available at:
http://natishalom.typepad.com/nati_shaloms_blog/2010/03/multitenancy-does-it-have-to-be-that-hard.html

[9] Mary Taylor and Chang Jie Guo. Data Integration and Composite Business Services, Part 3: Build a Multi-Tenant Data Tier with Access Control and Security, 2007, [Online].
Available at: http://www.ibm.com/developerworks/db2/library/techarticle/dm-0712taylor

[10] D. Jacobs and S. Aulbach. Ruminations on multi-tenant databases. In A. Kemper, H. Schoning, T. Rose, M. Jarke, T. Seidl, C. Quix, and C. Brochhaus, editors, BTW, volume 103 of LNI, pages 514–521. GI, 2007.

[11] Stefan Aulbach, Dean Jacobs, Alfons Kemper, Michael Seibold: A comparison of flexible schemas for software as a service. SIGMOD Conference 2009: 881-888

[12] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, Jan Rittinger: Multi-tenant databases for software as a service: schema-mapping techniques. SIGMOD Conference 2008: 1195-1206

[13] Foping F.S., Dokas I.M., Feehan J., Imran S., "A new hybrid schema-sharing technique for multitenant applications" Digital Information Management, 2009. ICDIM 2009. Fourth International Conference on , vol., no., pp.1,6, 1-4 Nov. 2009

[14] Craig D. Weissman and Steve Bobrowski: The design of the force.com multitenant internet application development platform. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (SIGMOD '09)

[15] Globally Unique Identifier, [Online].
Available at: http://en.wikipedia.org/wiki/Globally_unique_identifier

[16] SQL, [Online].
Available at: http://en.wikipedia.org/wiki/SQL

[17] L. Fegaras and D. Maier: “Optimizing object queries using an effective calculus”. ACM Transactions on Database Systems (TODS), 25(4), 2000.

[18] Chun-Feng Liao; Kung Chen; Jiu-Jye Chen, "Toward a tenant-aware query rewriting engine for Universal Table schema-mapping," Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference, 3-6 Dec. 2012

[19] PostgeSQL Documentation, [Online].
Available at: http://www.postgresql.org/docs/

[20] Hibernate ORM, [Online].
Available at: http://hibernate.org/orm/

[21] Ehcache, [Online].
Available at: http://ehcache.org/

[22] JSqlParser, [Online].
Available at: https://github.com/JSQLParser/JSqlParser/wiki

[23] Java Uuid Generator (JUG), [Online].
Available at: http://wiki.fasterxml.com/JugHome