Τμήμα Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

ΔΙΠΛΩΜΑΤΙΚΉ ΕΡΓΑΣΊΑ

«Ανάπτυξη Οδηγού Εγκατάστασης, Χρήσης και Δημιουργίας Εφαρμογής μέσω της Neo4J – noSQL Βάσης Δεδομένων»

ΦΟΙΤΗΤΈΣ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΟΥΤΣΟΓΙΌΥΛΑΣ (ΑΜ:15602)
ΧΡΙΣΤΙΝΑ – ΚΩΝ/ΝΑ ΑΣΣΙΟΎΡΑ (ΑΜ: 15647)

ΕΠΙΒΛΈΠΩΝ ΚΑΘΗΓΗΤΉΣ

ΙΩΆΝΝΗΣ ΤΣΑΚΝΆΚΗΣ

The #1 Database for Connected Data

ΠΕΡΙΛΗΨΗ ΠΤΥΧΙΑΚΗΣ

Η πλειονότητα των συμβατικών εφαρμογών αποθήκευσης και ανάκτησης υψηλής έντασης δεδομένων χρησιμοποιεί σχεσιακές βάσεις δεδομένων, οι οποίες υπάρχουν εδώ και πολλά χρόνια. Συνήθως, οι ανακτήσεις πραγματοποιούνται χρησιμοποιώντας SQL, μια δηλωτική γλώσσα ερωτημάτων. Γενικά, τα συστήματα σχεσιακών βάσεων δεδομένων είναι αποτελεσματικά έως ότου τα δεδομένα έχουν πολυάριθμους συσχετισμούς που χρειάζονται σύνδεση μεγάλων πινάκων.

Στην παρούσα πτυχιακή εργασία παρουσιάζεται η ανάπτυξη ενός οδηγού εγκατάστασης καθώς και η χρήση και δημιουργία εφαρμογής μέσω του προγράμματος Neo4j βασισμένο πάνω στην noSQL βάση δεδομένων. Αρχικά γίνεται μια εισαγωγή στις Β.Γ.Δ και στο μοντέλο γράφων με τις ετικέτες και τις ιδιότητες τους, στα χαρακτηριστικά τους, στις βασικές έννοιες και στη τεχνική μοντελοποίησης τους. Συγχρόνως, αναφέρονται οι αιτίες της μετατροπής του Σχεσιακού σχήματος σε μοντέλο γράφων καθώς και ο τρόπος υλοποίησης του με τα αντίστοιχα παραδείγματα. Παράλληλα, παρουσιάζονται οι λόγοι επιλογής τους, τα πλεονεκτήματα και τα μειονεκτήματα τους καθώς και ο τρόπος με τον οποίο αποθηκεύονται και επεξεργάζονται τα γραφήματα.

Μετέπειτα, γίνεται παρουσίαση του προγράμματος που θα δουλέψουμε, του Neo4J, αναλύοντας τα χαρακτηριστικά του, τα πλεονεκτήματα χρήσης και τις εφαρμογές του. Γίνεται αναφορά στη γλώσσα CYPHER, στην οποία στηρίζεται η Neo4J και από την οποία δανείζεται βασικές εντολές και τελεστές για την υλοποίηση κώδικα. Στη συνέχεια, παρουσιάζεται αναλυτικά κάθε εντολή με τα αντίστοιχα παραδείγματα της και τις εικόνες υλοποίησης τους στο πρόγραμμα. Ακολουθούν βοηθητικές ασκήσεις για την κατανόηση της θεωρίας και τέλος γίνεται υλοποίηση μίας εφαρμογής για καλύτερη αφομοίωση των εντολών που προαναφέρθηκαν.

ABSTRACT

The majority of conventional data-intensive storage and retrieval applications use relational databases, which have been around for many years. Typically, retrievals are carried out using SQL, a declarative query language. In general, relational database systems are effective until the data has numerous associations that need joining big tables.

This thesis presents the development of an installation guide in addition to the use and creation of an application through the Neo4j program based on the noSQL database. First and foremost, an introduction to the Graph DataBase is made, as well as in the graph model with its labels and properties, its characteristics, its basic concepts and its modeling technique. Simultaneously, the reasons for converting the Relational scheme into a graph model are mentioned, along with the way of its implementation with the corresponding examples. At the same time, the reasons for their selection, their advantages and disadvantages, plus the way in which the graphs are stored and processed are presented.

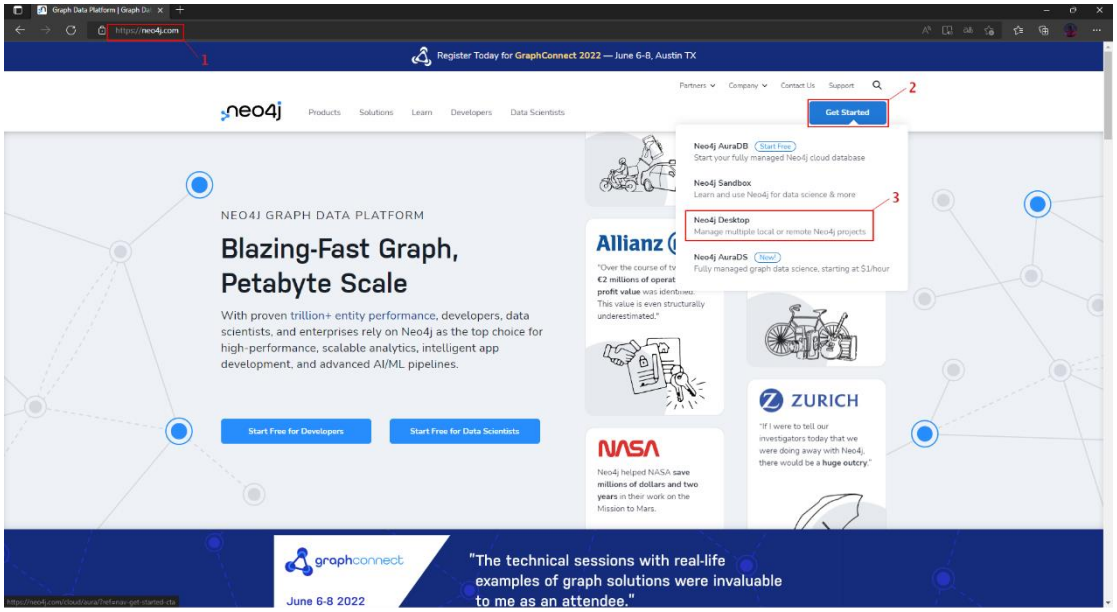
Afterwards, the program we will work on, Neo4J, is presented, analyzing its features, advantages of use and its applications. Reference is made to the CYPHER language, on which Neo4J is based and from which it borrows basic commands and operators to implement code. Then, each command is presented in detail with its corresponding examples and their implementation images in the program. Helpful exercises for understanding the theory follow and finally an application is implemented for a better assimilation of the aforementioned commands.

ΠΕΡΙΕΧΟΜΕΝΑ

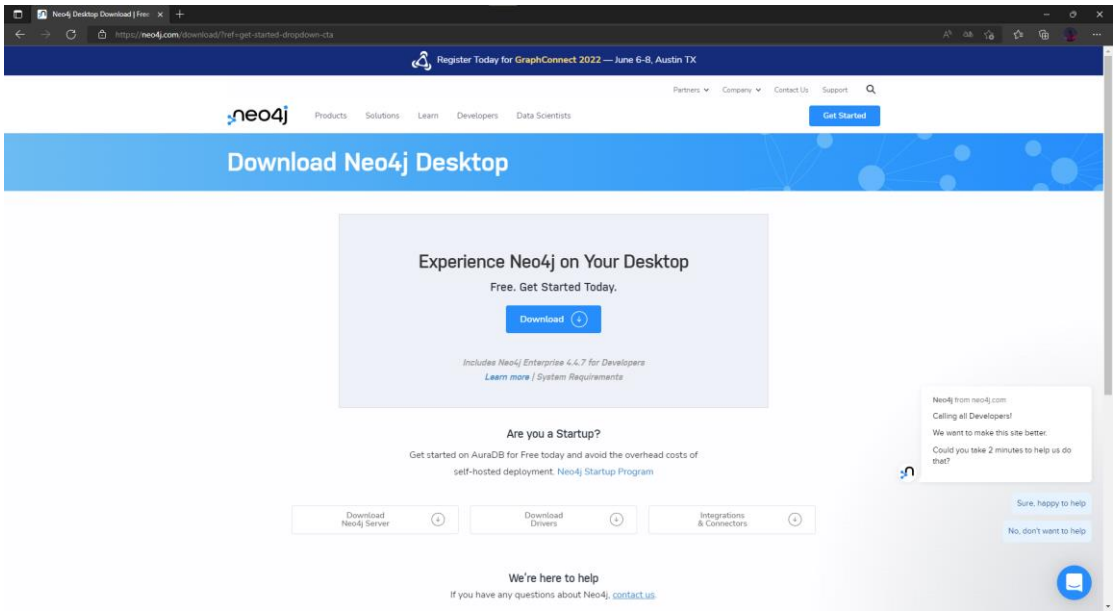
ΚΕΦΑΛΑΙΟ 1

1.1 ΟΔΗΓΟΣ ΕΓΚΑΤΑΣΤΑΣΗΣ NEO4J

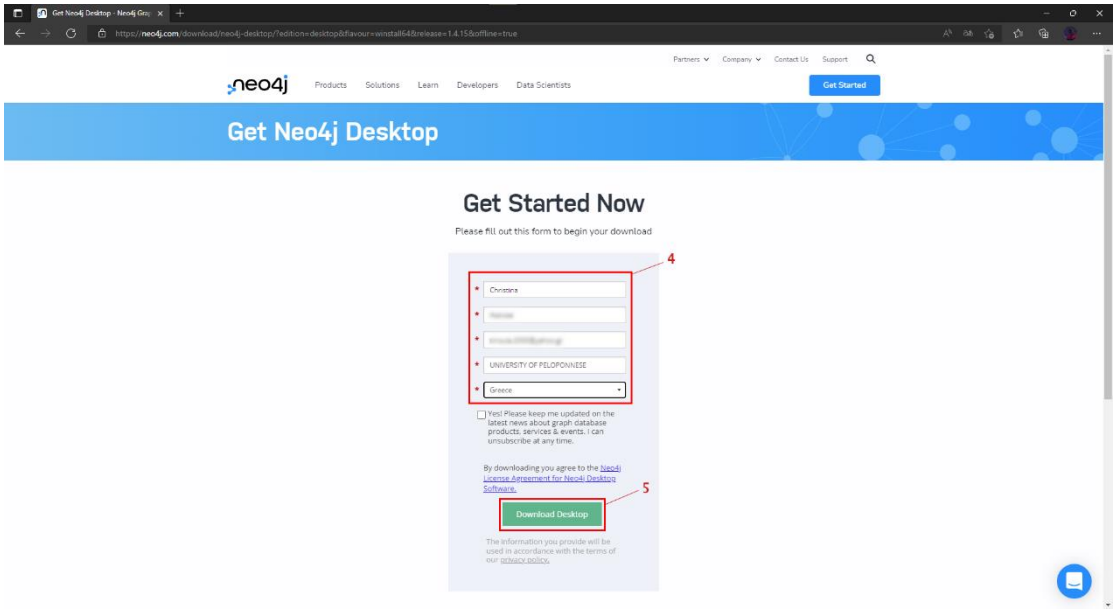
1. Αρχικά μεταβαίνουμε στην ιστοσελίδα του NEO4J (**https://neo4j.com/**)
2. Επιλέγουμε πάνω δεξιά το κουμπί **Get Started**
3. Επιλέγουμε **την τρίτη επιλογή Neo4j Desktop**



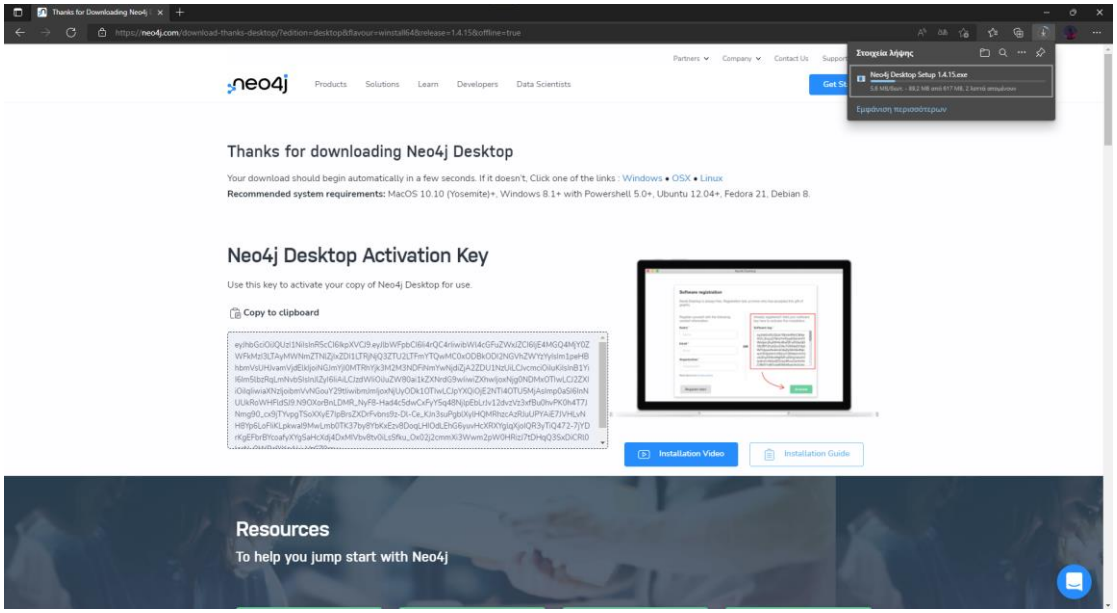
- Στην επόμενη σελίδα πατάμε **Download.**



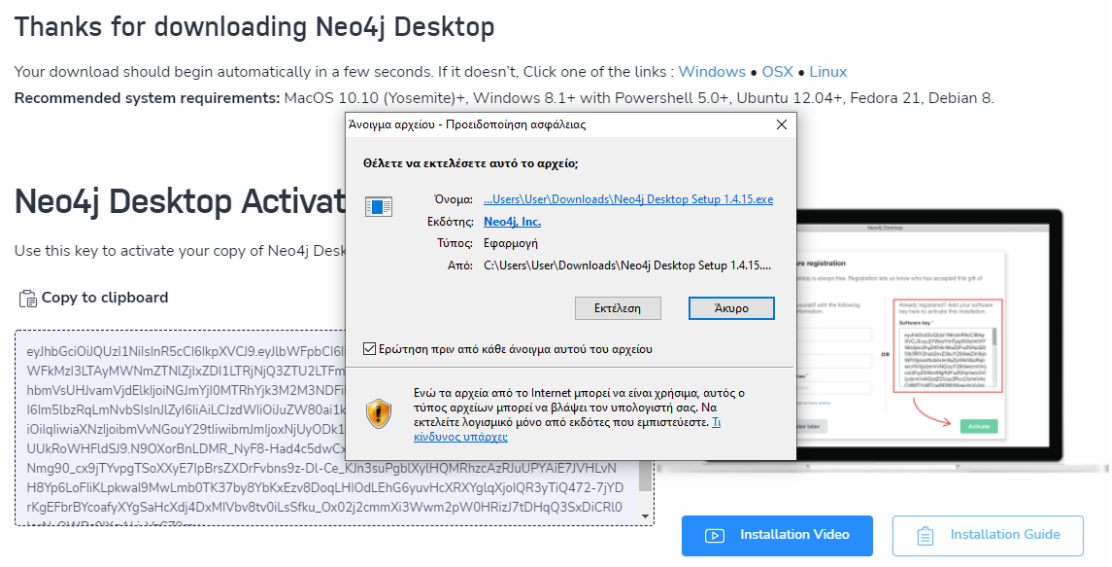
4. **Συμπληρώνουμε τα στοιχεία μας** (Όνομα, Επώνυμο, Email, Σχολή, Χώρα)
5. Πατάμε **Download Desktop**



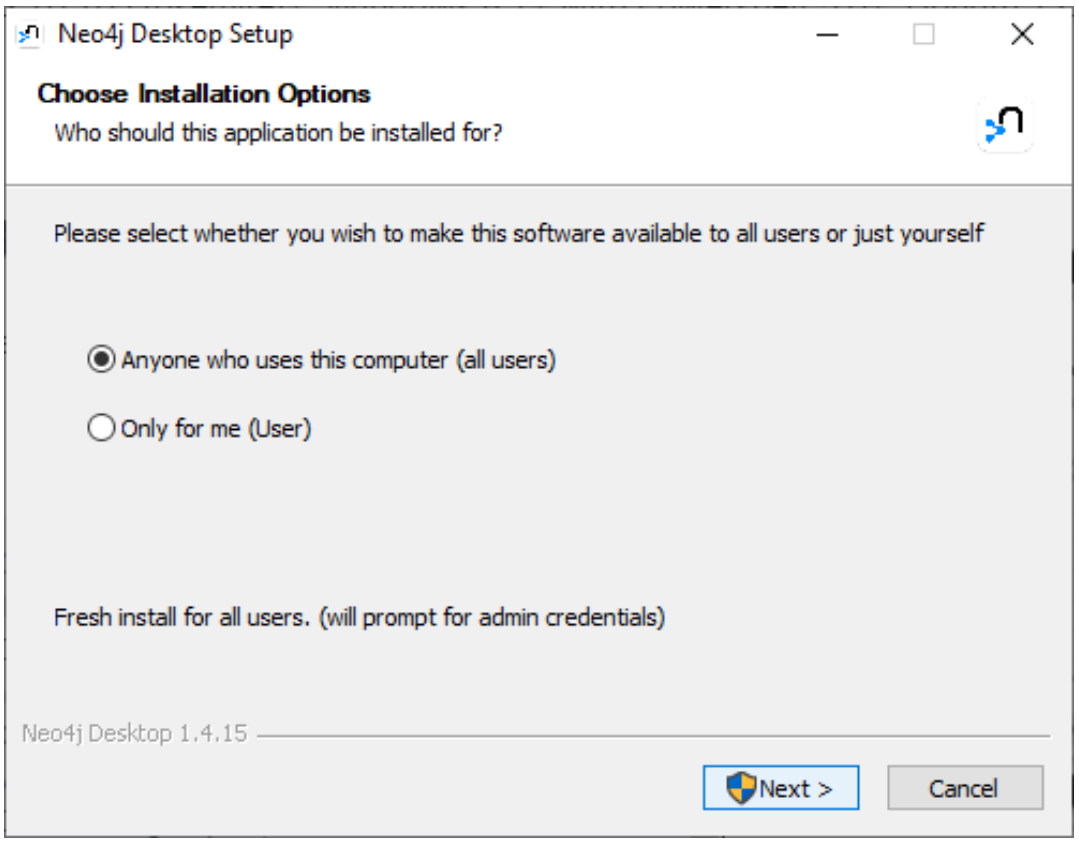
- **Αποθηκεύουμε το κλειδί** μας **(Activation Key)** για παρακάτω χρήση.
- Πατάμε **Copy to clipboard**



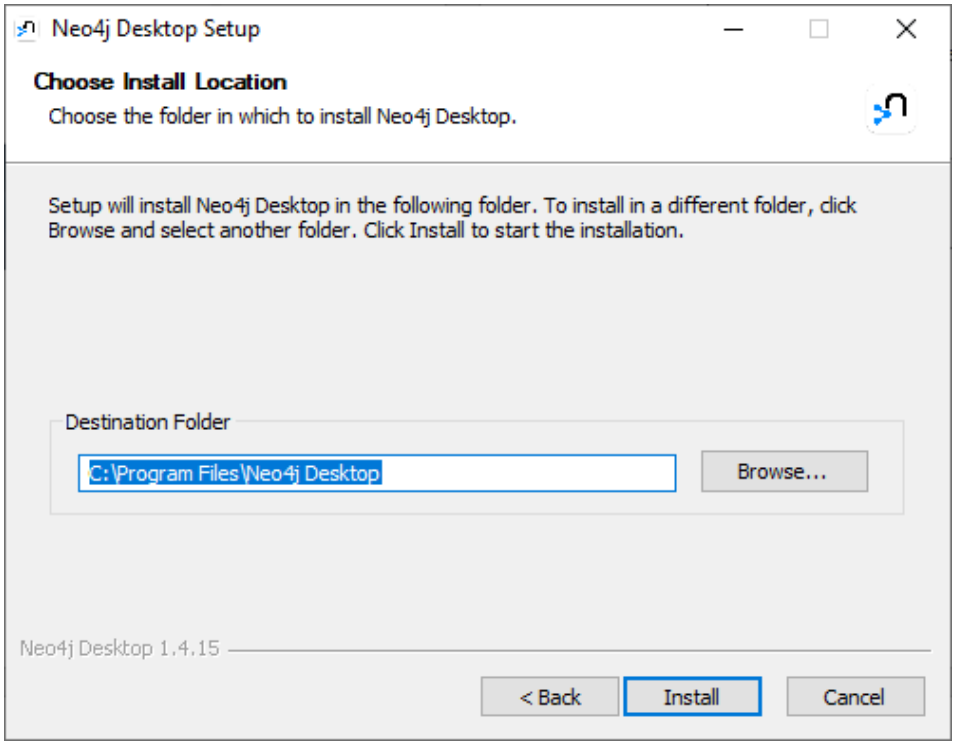
- Όταν ολοκληρωθεί η λήψη του αρχείου, ανοίγουμε το .exe αρχείο
- Στο παράθυρο που θα εμφανιστεί πατάμε **Εκτέλεση**



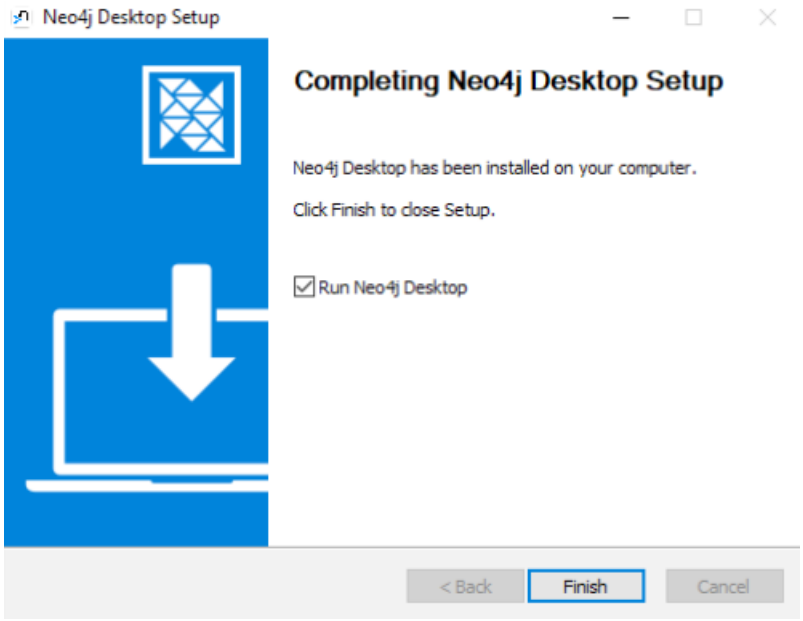
- Επιλέγουμε **Anyone who uses this computer (all users)** και στη συνέχεια πατάμε **Next**



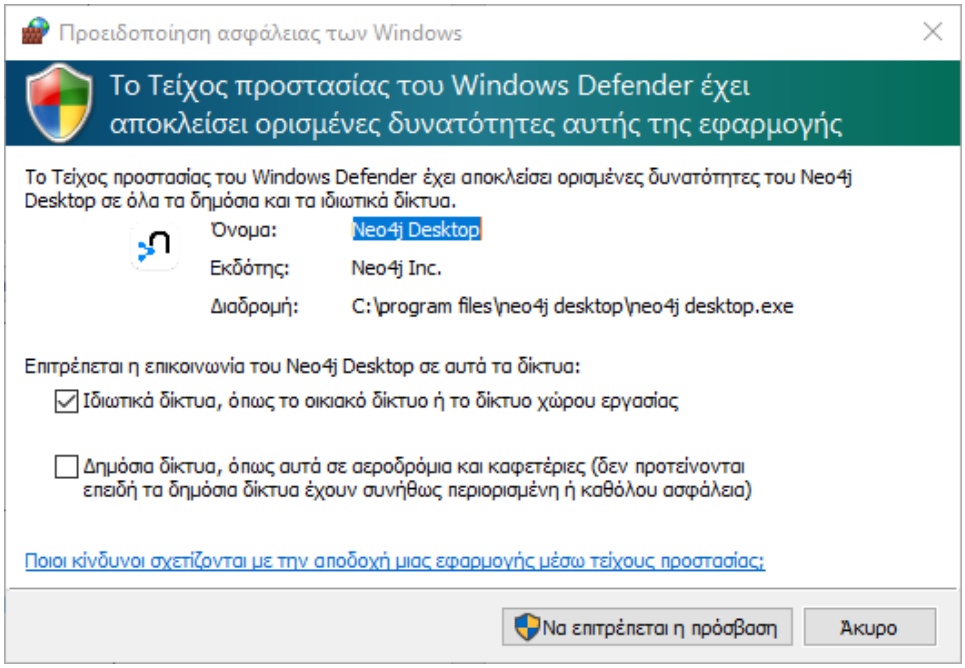
- Επιλέγουμε που θέλουμε να εγκατασταθεί το πρόγραμμά μας μέσω της επιλογής **Browse** και Επιλέγουμε **Install**



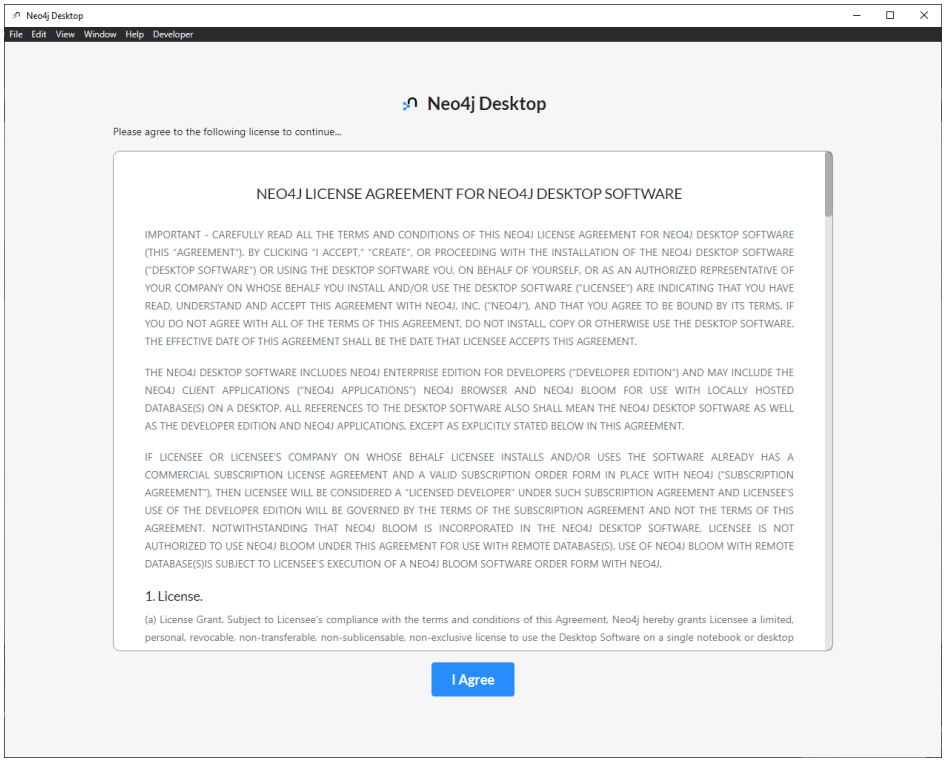
- **Τέλος πατάμε Finish**



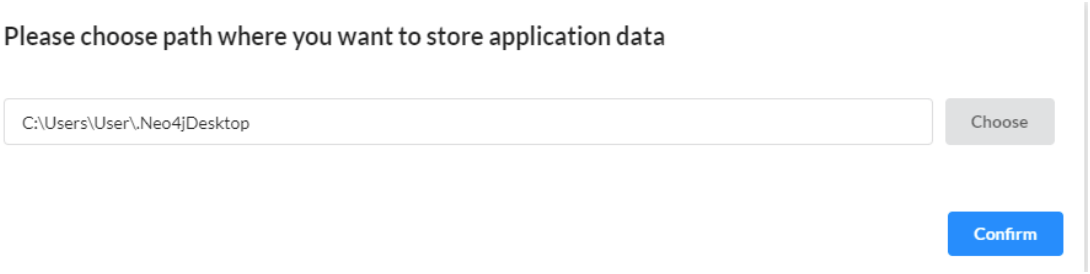
- Σε περίπτωση που εμφανιστεί το μήνυμα περί Τείχους Προστασίας, επιλέγουμε **<u>Ιδιωτικά δίκτυα</u>** και στη συνέχεια **<u>Να επιτρέπεται η πρόσβαση</u>**



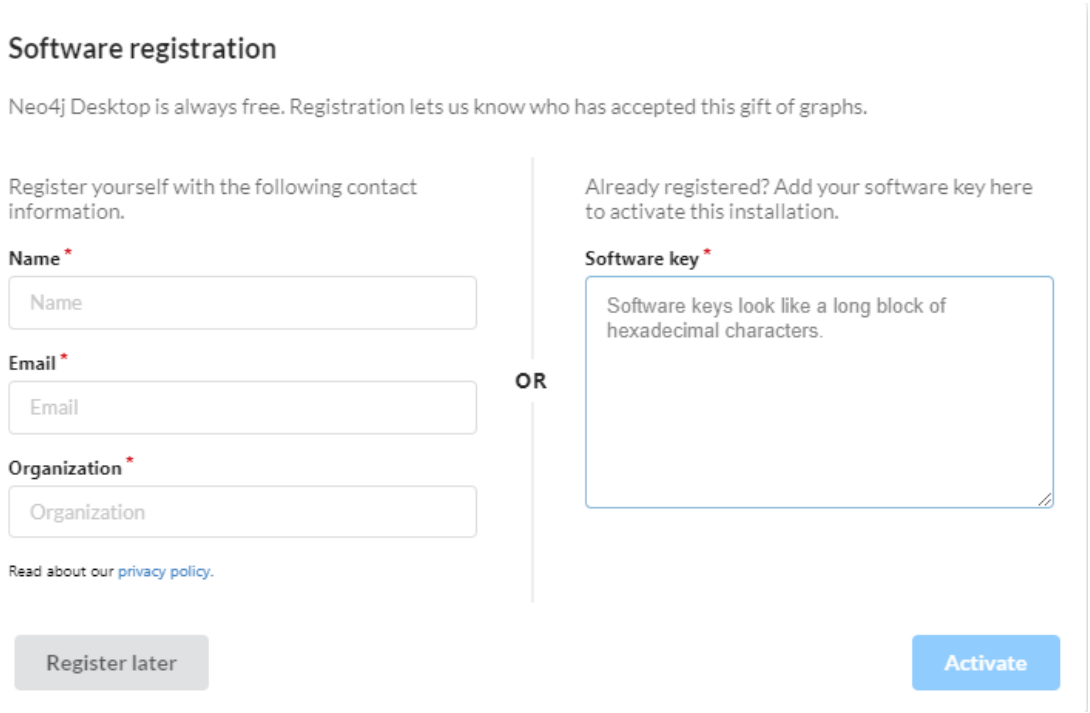
- Στο επόμενο παράθυρο αποδεχόμαστε τους Όρους Χρήσης του προγράμματος **(I Agree)**



- Επιλέγουμε το μονοπάτι (path) που θέλουμε να αποθηκευτούν τα δεδομένα του προγράμματος.

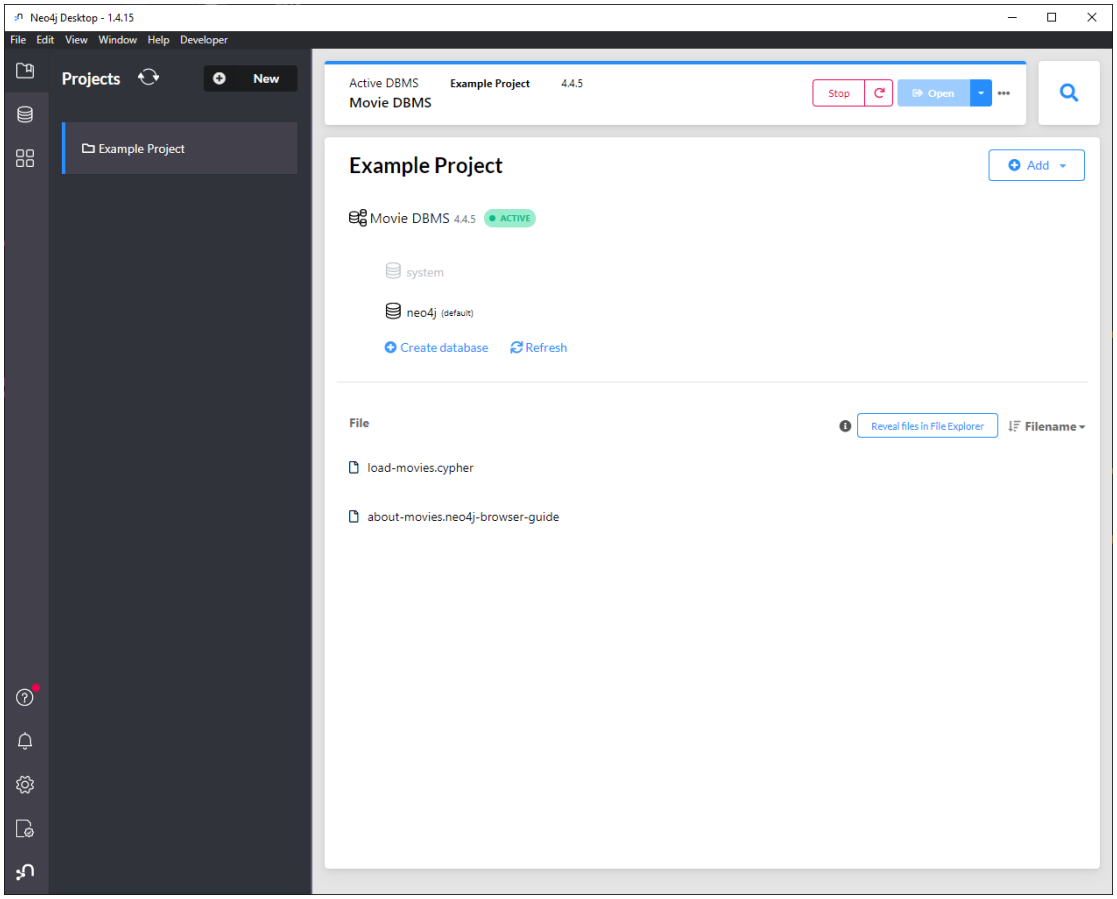


- Στο παράθυρο που θα αναδυθεί, μπορούμε να εγγραφούμε είτε βάζοντας τα στοιχεία (Όνομα, Email, Οργανισμός) είτε με το κλειδί (Activation Key) που έχουμε αποθηκεύσει προηγουμένως.

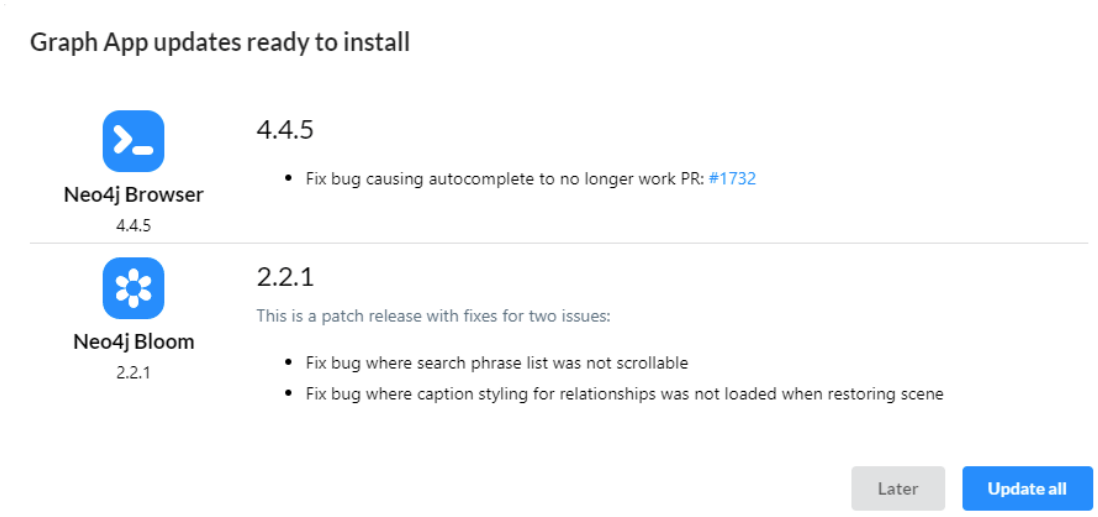


- **Τέλος πατάμε Activate.**

- **Εφόσον έχουν εκτελεστεί σωστά οι παραπάνω οδηγίες, θα εμφανιστεί αυτό το κεντρικό παράθυρο.**



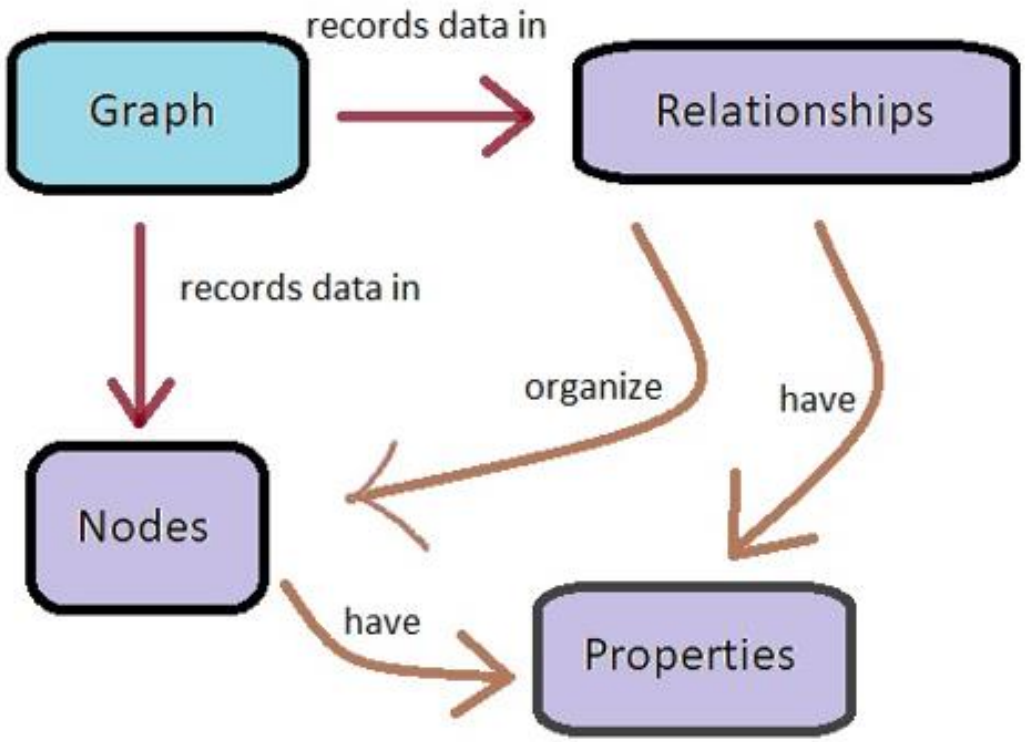
- Κάνουμε την Ενημέρωση των Γράφων πατώντας **Update All.**



1.2 ΕΙΣΑΓΩΓΗ

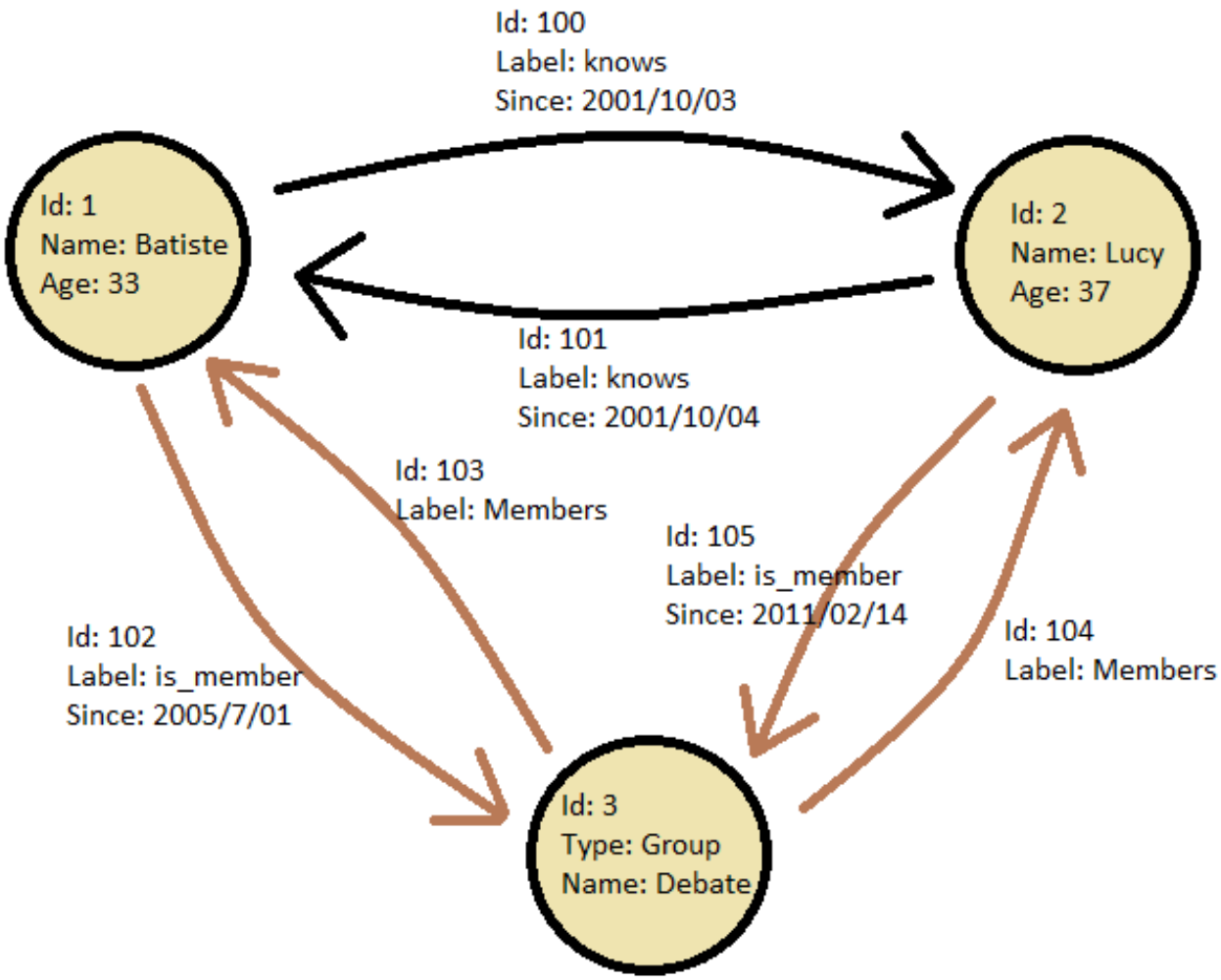
1.2.1 ΜΟΝΤΕΛΟ ΓΡΑΦΩΝ ΜΕ ΕΤΙΚΕΤΕΣ ΚΑΙ ΙΔΙΟΤΗΤΕΣ

- Οι δομές γράφων χρησιμοποιούνται για τα σημασιολογικά ερωτήματα και την αποθήκευση πληροφοριών μέσω μιας Βάσης Δεδομένων Γράφων.
- Ένας γράφος αποτελείται από κόμβους και ακμές. Οι κόμβοι αντιπροσωπεύουν τις entities. Οι ακμές αντιπροσωπεύουν τις σχέσεις (relationships) που τους συνδέουν.
- Και οι κόμβοι αλλά και οι ακμές είναι σε θέση να αντιπροσωπεύουν και να αποθηκεύουν δεδομένα.
- Ο γράφος Ιδιοτήτων είναι ο πιο κοινός τύπος γράφων.
  Επιτρέπει την ανάθεση ετικετών και ιδιοτήτων (όπως έχουμε ήδη εφαρμόσει στα παραδείγματα) στις κορυφές και τις άκρες του γράφου.
- Όλες οι κορυφές και άκρες έχουν ένα μοναδικό αναγνωριστικό η κάθε μία, που μπορεί να χρησιμοποιηθεί σαν σημείο αναφοράς έτσι ώστε να μπορέσουμε να δια-συνδέσουμε με τα υπόλοιπα δεδομένα με τη μορφή ζευγών κλειδιού-τιμής.
- Στο παρακάτω σχήμα απεικονίζεται η λογική της μοντελοποίησης δεδομένων μέσω του γράφου ιδιοτήτων (property graph):

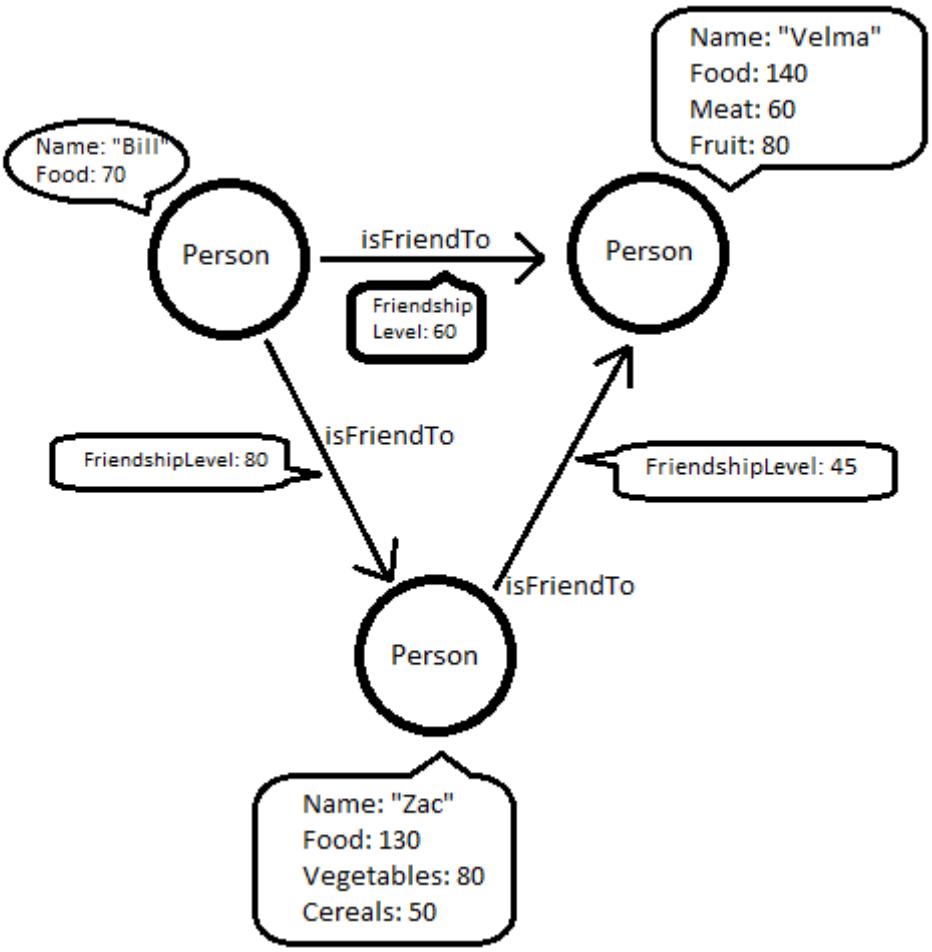


- Η δομή που χρησιμοποιούν οι περισσότερες Β.Δ.Γ. για την αποθήκευση πληροφοριών είναι το μοντέλο γράφων με ετικέτες και ιδιότητες.

  Ο γράφος ιδιοτήτων περιέχει τα βασικά χαρακτηριστικά για την μοντελοποίηση γράφων, αλλά δημιουργούνται και διαφορετικοί τύποι γράφων χάρη στις απλές μορφοποιήσεις του. Εν ολίγοις, τα συστήματα που υποστηρίζουν το μοντέλο του γράφου ιδιοτήτων, υποστηρίζουν έμμεσα και άλλους τύπους γράφων, όπως παρακάτω:

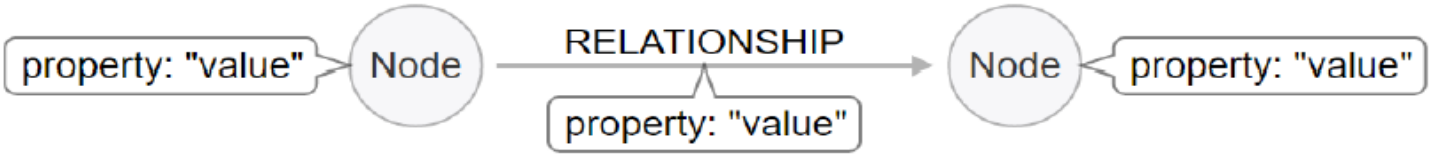


Βασικά στοιχεία ενός γράφου ιδιοτήτων και οι συνδέσεις μεταξύ τους:



1.2.2 ΧΑΡΑΚΤΗΡΙΣΤΙΚΆ

- Η γενική βασική παράσταση του labeled property graph είναι:



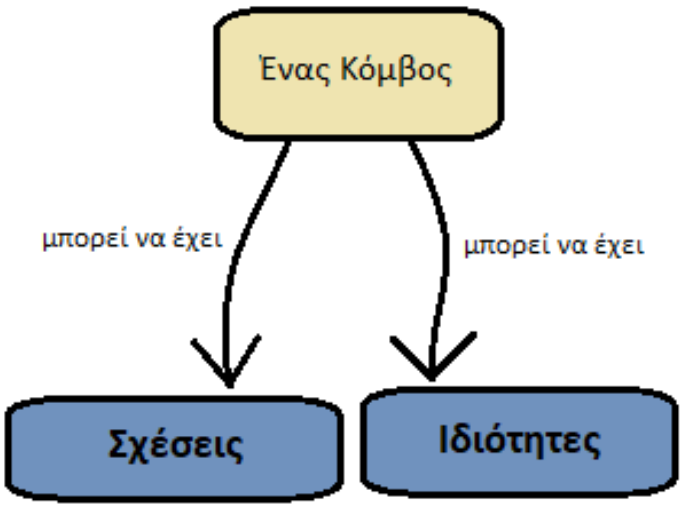
- Τα χαρακτηριστικά του μοντέλου είναι τα ακόλουθα:

i. Ένας γράφος με ετικέτες και ιδιότητες αποτελείται από κόμβους, σχέσεις, ιδιότητες και ετικέτες.

ii. Πιο Αναλυτικά:

- Οι **Κόμβοι** μπορούν να αποκτήσουν ετικέτες, οι οποίες δίνουν κάποιον ρόλο σε αυτούς, ενώ παράλληλα δημιουργούνται διακριτές ομάδες και υποομάδες αυτών των κόμβων. Τα στοιχεία αναγράφονται στον πίνακα RDBMS και κάθε κόμβος περιέχεται ως εγγραφή του πίνακα με την αντίστοιχη ετικέτα.

  Οι κόμβοι περιέχουν και αποθηκεύουν ιδιότητες σε μορφή ζευγαριών (key-value). Συνδέονται μεταξύ τους με σχέσεις.

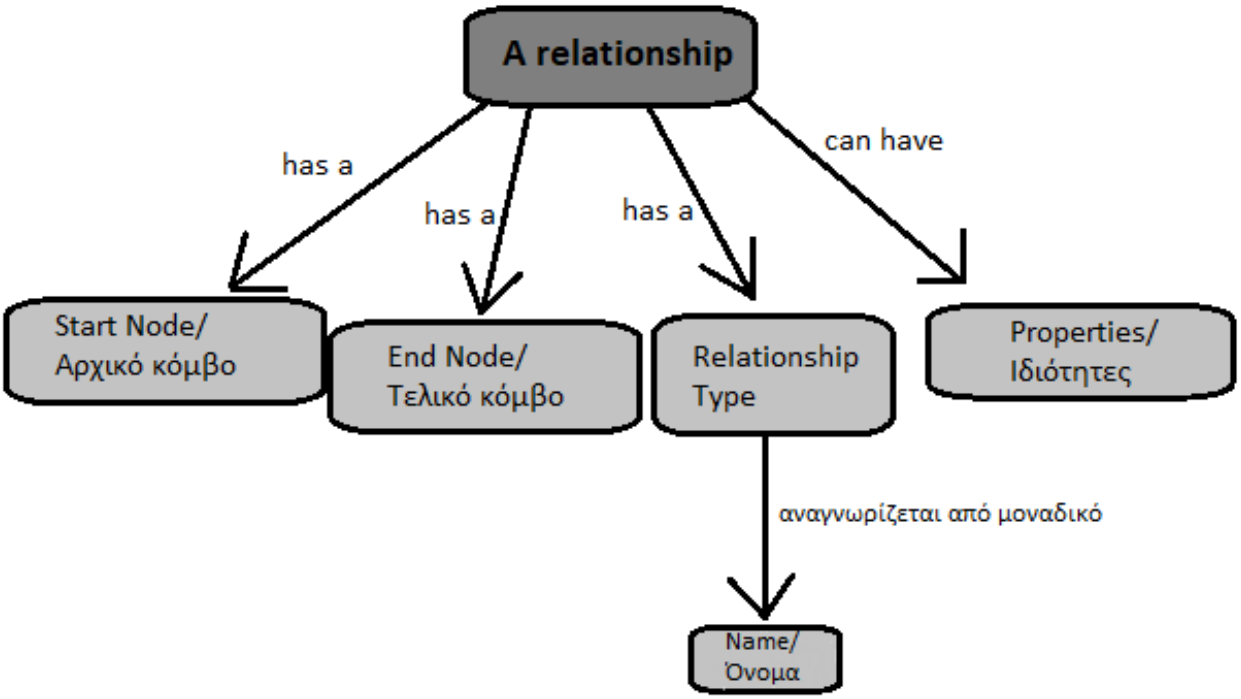


- Οι **Σχέσεις** συνδέουν τους κόμβους και καθορίζουν μια δομή και τη σημασιολογία του γράφου. Όλες οι σχέσεις έχουν μοναδικό όνομα, αρχικό και τελικό κόμβο αλλά και κατεύθυνση. Επίσης, οι σχέσεις αυτές πάντα έχουν αρχικό και τελικό κόμβο.

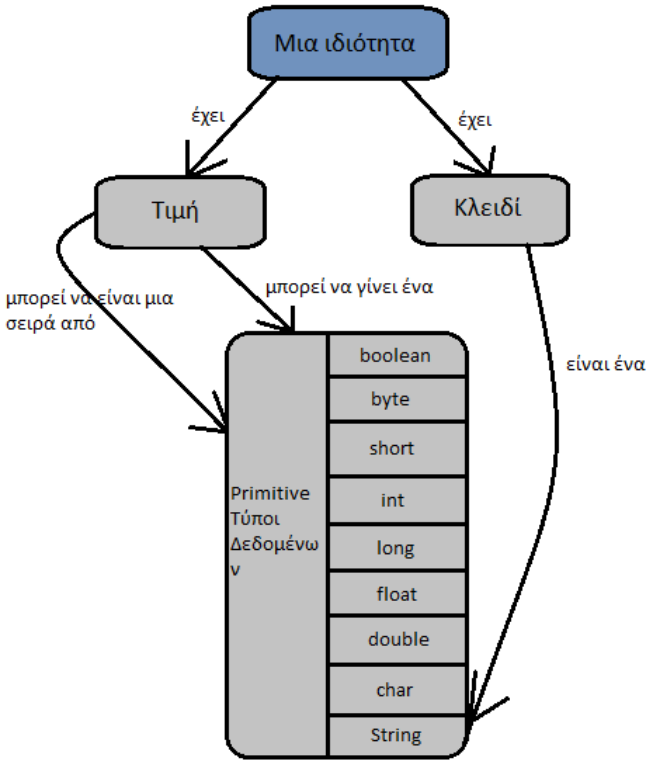
  Ακόμα και οι σχέσεις (relationships) περιέχουν ιδιότητες (properties).

Η σχέση από τη στιγμή που έχει μια κατεύθυνση, μπορεί να χαρακτηριστεί είτε εισερχόμενη είτε εξερχόμενη σε έναν κόμβο, εξίσου σημαντική πληροφορία για το σχήμα μας.

Οι σχέσεις μπορούν να διαβαστούν με διάφορους τρόπους (σειρά και κατεύθυνση), ενώ για την αποτελεσματική διέλευση του γράφου, όλες οι σχέσεις έχουν ένα μοναδικό όνομα σαν ετικέτα, δηλαδή Relationship Type.



- Οι **Ιδιότητες** έχουν τη μορφή ζευγαριών (key-value).
  Τα ονόματα είναι συνήθως συμβολοσειρές και οι τιμές είναι είτε συμβολοσειρές Java, είτε πρωτόγονοι τύποι δεδομένων είτε και πίνακες αυτών.
  Εντός των ιδιοτήτων που αποθηκεύονται στις σχέσεις, παρέχονται επιπλέον πληροφορίες και δεδομένα κατά τη διάσχιση των γράφων όπως ημερομηνίες, περιορισμοί διάσχισης και λοιπά.



1.2.3 ΒΑΣΙΚΕΣ ΈΝΝΟΙΕΣ

- Μονοπάτια – Paths

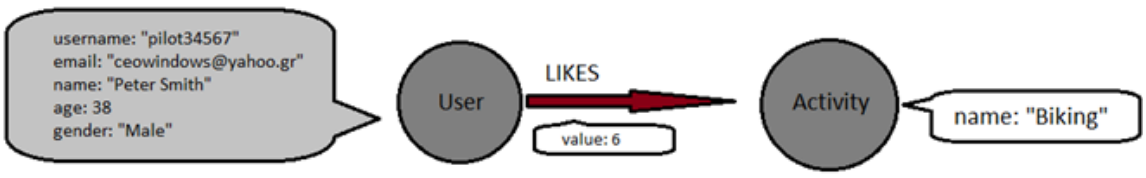
  Το μονοπάτι είναι ο τρόπος διασύνδεσης ενός ή περισσότερων κόμβων μεταξύ τους, δηλαδή οι σχέσεις τους. Αποτελεί συνήθως αποτέλεσμα ενός ερωτήματος ή διάσχισης και ανακτάται κατά αυτόν τον τρόπο.

- Διάβαση – Traversal

  Διάβαση σημαίνει η επίσκεψη ενός γράφου στους κόμβους του, ακολουθώντας τις σχέσεις που του συνδέουν, με βάση κάποιους ορισμένους κανόνες. Τις περισσότερες φορές επισκέπτεται συγκεκριμένο τμήμα του γράφου, εκεί όπου βρίσκονται οι πιο ενδιαφέρουσες σχέσεις και κόμβοι.

Παράδειγμα

- Παρακάτω έχουμε ένα παράδειγμα μοντελοποίησης μεταξύ μιας σχέσης “LIKES” , ενός χρήστη (user) και μιας δραστηριότητας (activity), με χρήση μοντέλου γράφων με ετικέτες και ιδιότητες:



Στον γράφο συμπεριλαμβάνεται ο χρήστης με το όνομα “Peter Smith”, του οποίου του αρέσει η δραστηριότητα “Biking” σε βαθμό ίσο με 6.

Με βάση αυτό το παράδειγμα μπορούμε να διακρίνουμε πολλά χαρακτηριστικά του μοντέλου:

- Πρωτίστως, έχουμε έναν γράφο αποτελούμενο από δύο κόμβους και μία σχέση που τους συνδέει
- Στον κόμβο του χρήστη έχει προστεθεί η ετικέτα “User”, ενώ στον κόμβο της δραστηριότητας έχει προστεθεί η ετικέτα “Activity”.
- Οι δύο κόμβοι έχουν αποθηκευμένες και ιδιότητες με τη μορφή ονόματος-τιμής. Ο κόμβος χρήστη έχει τις ιδιότητες username, email, name, age και gender, ενώ ο κόμβος της δραστηριότητας έχει μια ιδιότητα name.
- Η σχέση “LIKES” δίνει δομή και σημασιολογία στο γράφο, ενώνοντας τους δύο κόμβους και δείχνοντας πως ένας χρήστης έχει δηλώσει ότι του αρέσει μια δραστηριότητα. Παρατηρούμε ότι η σχέση έχει ένα μοναδικό όνομα (LIKES), έχει μια κατεύθυνση (από το χρήστη προς τη δραστηριότητα), έναν αρχικό κόμβο (user) και έναν τελικό κόμβο (activity).
- Τέλος, βλέπουμε ότι η σχέση “LIKES” έχει αποθηκευμένη μια ιδιότητα με τη μορφή ονόματος-τιμής. Έχει δηλαδή την ιδιότητα με το όνομα “value” και την τιμή 6, δίνοντάς μας μια επιπλέον πληροφορία για τα δεδομένα μας. Αυτή είναι, πως ο χρήστης όχι μόνο έχει δηλώσει πως του αρέσει η συγκεκριμένη δραστηριότητα, αλλά έχει προσδιορίσει και το βαθμό με τον οποίο του αρέσει.

1.2.4 ΒΑΣΙΚΗ ΤΕΧΝΙΚΗ ΜΟΝΤΕΛΟΠΟΙΗΣΗΣ

- Πλεονέκτημα του μοντέλου γράφων είναι, πως εκτός από την απεικόνιση των γράφων και σχέσεων με τον ακριβές τρόπο σκέψης μας, περιέχει και μια εικόνα ερωτήσεων που θέλουμε να απαντηθούν. Εν ολίγοις, αυτή η διαδικασία μοντελοποίησης στην ουσία αποτελεί μια καλή προσπάθεια δημιουργίας δομών γράφων από μερικά ερωτήματα που θέλουμε να απαντήσουμε.

- Τα βήματα είναι τα εξής:
    1. Αναφέρουμε τους στόχους που θέλουμε να έχει το μοντέλο μας.
    2. Μετατρέπουμε αυτούς τους στόχους σε αντίστοιχα ερωτήματα που θα απαντηθούν.
    3. Προσδιορίζουμε τις οντότητες και τις σχέσεις των παραπάνω ερωτημάτων.
    4. Δημιουργείται ο γράφος με βάση τα παραπάνω στοιχεία.

- Αυτό που πρέπει να προσέξουμε είναι ο προσδιορισμός των οντοτήτων και των σχέσεων του πεδίου μας. Ένας τρόπος για να ελέγξουμε μερικά στοιχεία όπως (Τι είναι ιδιότητα, η ετικέτα, η οντότητα, η σχέση κλπ.), ελέγχουμε τη φυσική γλώσσα που χρησιμοποιούμε για να περιγράψουμε ερωτήματα.
- Είναι καλό να αναφερθεί πως δεν υπάρχει κάποια στάνταρ τεχνική που να ταιριάζει σε κάθε περίπτωση, αλλά θα πρέπει να διαμορφώνουμε τα μοντέλα σύμφωνα με την κριτική μας σκέψη, λαμβάνοντας υπόψιν κάθε φορά τις ιδιαιτερότητες κάθε προβλήματος.

1.2.5 ΜΕΤΑΤΡΟΠΗ ΣΧΕΣΙΑΚΟΥ ΣΧΗΜΑΤΟΣ ΣΕ ΜΟΝΤΕΛΟ ΓΡΑΦΩΝ ΜΕ ΕΤΙΚΕΤΕΣ ΚΑΙ ΙΔΙΟΤΗΤΕΣ

ΒΑΣΙΚΉ ΙΔΈΑ

- Οι σχεσιακές βάσεις δεδομένων στηρίζονται σε αναζητήσεις ευρετηρίου και ενώσεις πινάκων για την ένωση διαφορετικών οντοτήτων. Ως αποτέλεσμα μπορεί να έχουμε πρόβλημα με την απόδοση, λόγω των πολυπληθών ενωμένων πινάκων, των εκατομμύρια σειρών σε πίνακες ή σύνθετα ερωτήματα που διαπερνούν διάφορα επίπεδα μέσω υποερωτημάτων.

- Σε ένα γράφημα όμως, δεν ανησυχούμε για ενώσεις πινάκων και αναζητήσεις ευρετηρίου, γιατί τα δεδομένα γραφήματος δομούνται από κάθε ξεχωριστή οντότητα και τις σχέσεις της με άλλες ξεχωριστές οντότητες.

1.2.6 ΠΩΣ ΠΕΡΝΑΜΕ ΑΠΟ ΤΗ ΔΗΜΙΟΥΡΓΙΑ ΣΧΕΣΙΑΚΩΝ ΜΟΝΤΕΛΩΝ ΔΕΔΟΜΕΝΩΝ ΣΕ ΕΝΑ ΜΟΝΤΕΛΟ ΔΕΔΟΜΕΝΩΝ ΓΡΑΦΗΜΑΤΟΣ;

- Πίνακας <==> Ετικέτα σε κόμβο
  Κάθε πίνακας οντοτήτων του σχεσιακού μοντέλου, γίνεται ετικέτα σε κόμβους στο μοντέλο γραφήματος.
- Γραμμή <==> Κόμβο
  Κάθε σειρά/εγγραφή (record) σε έναν πίνακα σχεσιακών οντοτήτων μετατρέπεται ως κόμβος στο γράφημα.
- Στήλη <==> Ιδιότητα κόμβου
  Οι στήλες (πεδία) στους σχεσιακούς πίνακες γίνονται ιδιότητες κόμβου στο γράφημα.
  Διατηρούνται μόνο κύρια κλειδιά εφαρμογής – καταργούμε τα κύρια τεχνικά κλειδιά, διατηρούμε τα κύρια κλειδιά της εφαρμογής.

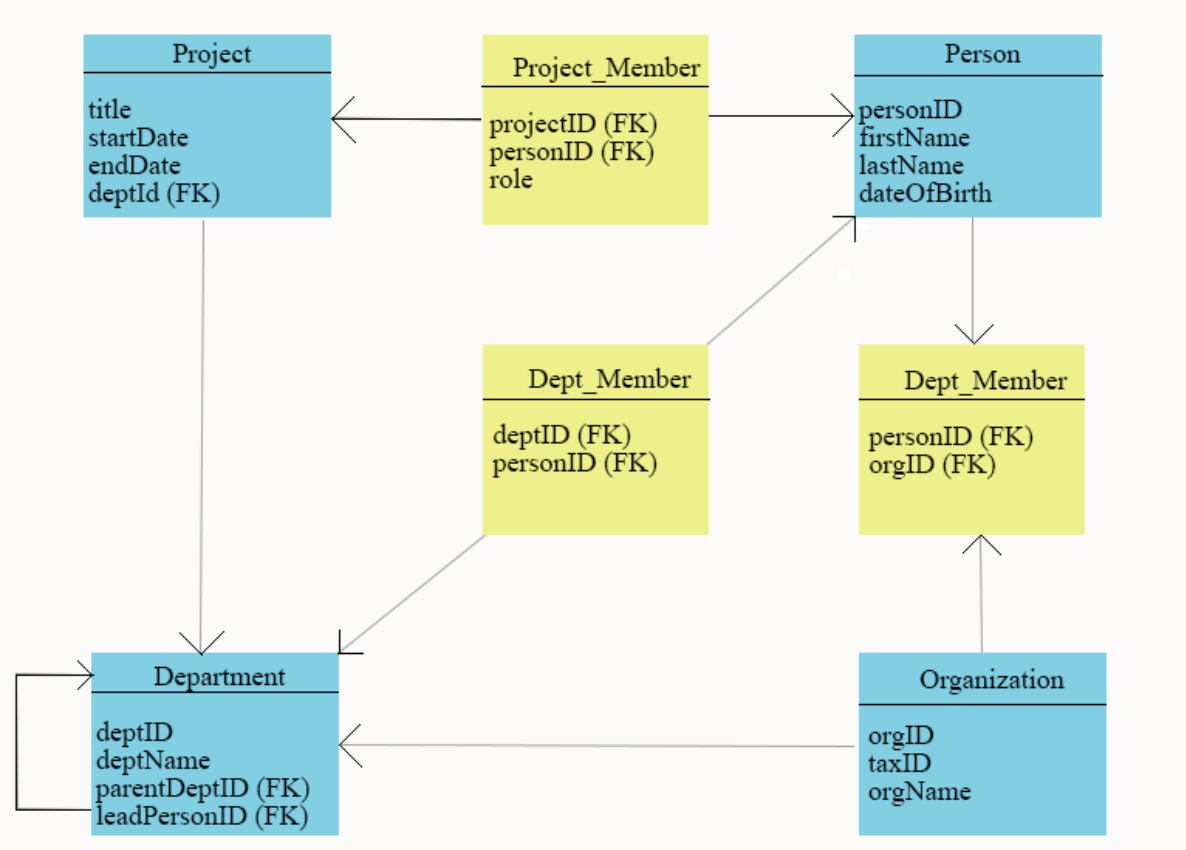
- Ξένα κλειδιά <==> Σχέσεις
  Αντικαθιστούμε τα ξένα κλειδιά (foreign keys) στον άλλον πίνακα με σχέσεις και τα αφαιρούμε στη συνέχεια.

Sr. No	RDBMS	Graph Database
1	Tables	Graphs
2	Rows	Nodes
3	Columns, Data	Properties and Its values
4	Constraints	Relationships
5	Joins	Traversal

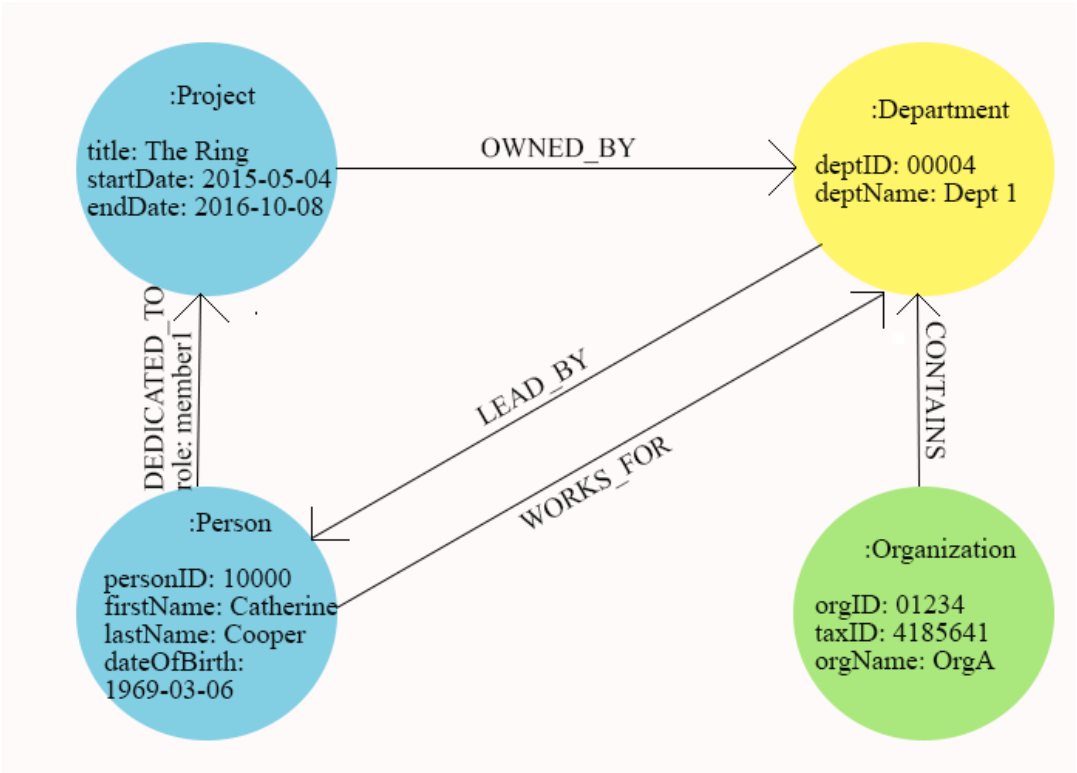
- Join Arrays = Σχέσεις
  Οι πίνακες σύνδεσης μετατρέπονται σε σχέσεις και οι στήλες αυτών γίνονται ιδιότητες της σχέσης.
- Προσθήκη περιορισμών / ευρετηρίων
  Συνήθως προστίθενται μοναδικοί περιορισμοί για τα κύρια κλειδιά και επίσης ευρετήρια για χαρακτηριστικά συχνής αναζήτησης.
- Δίχως προεπιλογές
  Δεν είναι απαραίτητο να αποθηκεύουμε προεπιλεγμένες τιμές, οπότε καταργούμε κάποια δεδομένα.
- Εκκαθάριση δεδομένων
  Διπλά δεδομένα ενσωματωμένα σε μη κανονικοποιημένους πίνακες μπορούν να αντληθούν από ξεχωριστούς κόμβους έτσι ώστε να αποκτήσουμε ένα πιο καθαρό μοντέλο.
- Columns ευρετηρίου σε πίνακα
  Τα ονόματα στηλών με ευρετήριο συνήθως υποδεικνύουν μια ιδιότητα πίνακα.

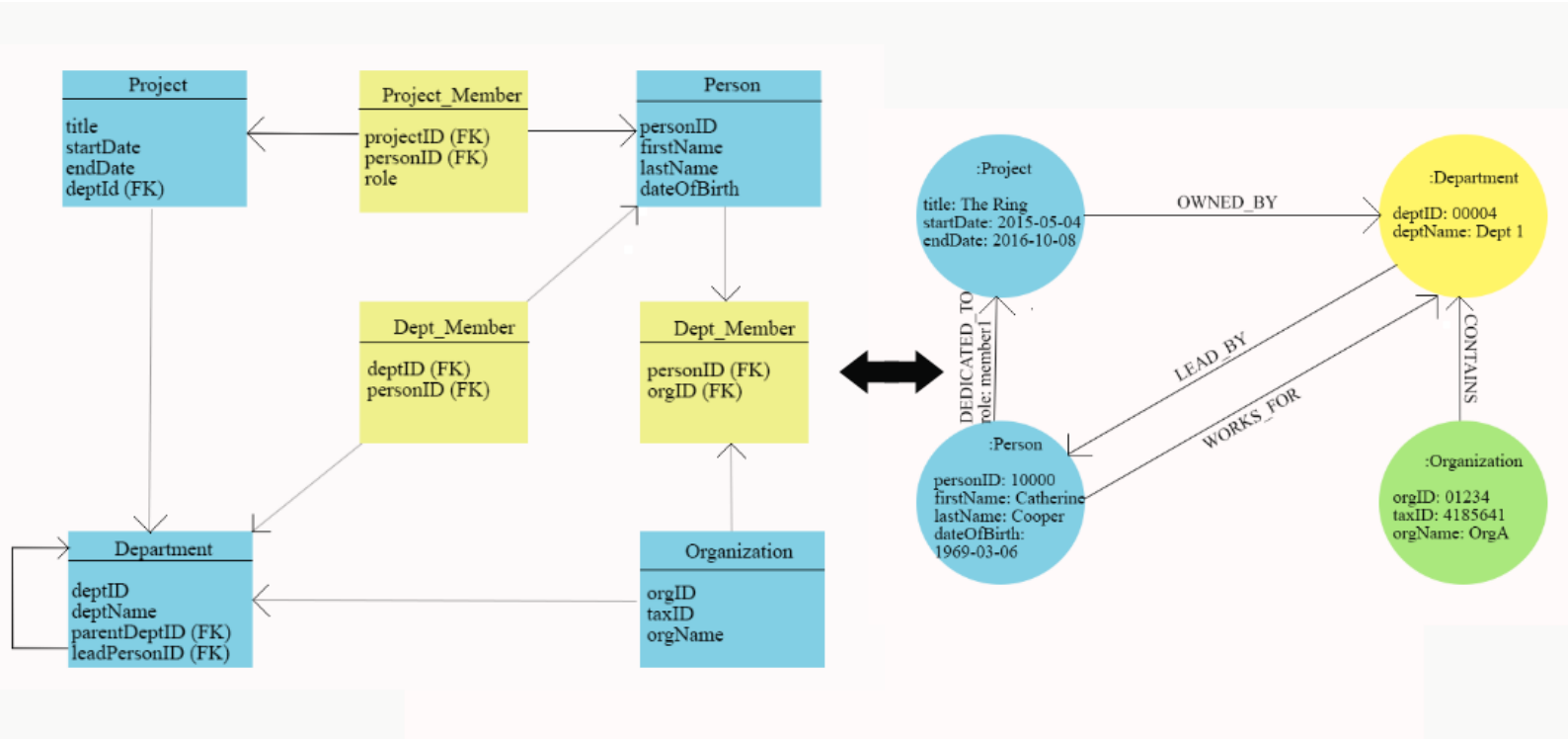
1.2.7 ΜΕΡΙΚΑ ΠΑΡΑΔΕΙΓΜΑΤΑ

- Σχεσιακό Σχήμα



- Μοντέλο Γράφου



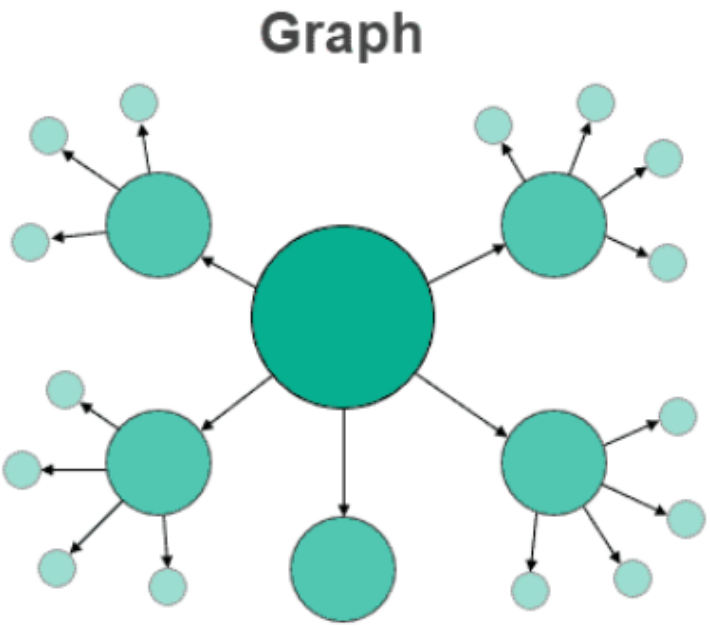


ΚΕΦΑΛΑΙΟ 2

2.1 ΟΡΙΣΜΟΣ – ΤΙ ΕΣΤΙ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ΓΡΑΦΩΝ ;

Στην ουσία μία Βάση Δεδομένων Γραφήματος αποτελεί ένα σύστημα (βάσης δεδομένων) τύπου SQL και βασίζεται σε μία τοπογραφική δομή δικτύου. Η ιδέα αυτή συσχετίζεται και προέρχεται ταυτόχρονα από τη θεωρία γραφημάτων στα μαθηματικά, όπου εκείνα τα γραφήματα αντικατοπτρίζουν σύνολα δεδομένων, χρησιμοποιώντας κόμβους-(nodes), ακμές-(edges) και ιδιότητες-(properties). Ειδικότερα, οι κόμβοι αποτελούν οντότητες δεδομένων και αντιπροσωπεύουν πολλών ειδών αντικείμενα που τίθενται προς παρακολούθηση/εξέταση, όπως για παράδειγμα κάποιο πρόσωπο, έναν λογαριασμό ή μια τοποθεσία. Στη συνέχεια, οι ακμές ή αλλιώς γραμμές, είναι οι κρίσιμες έννοιες στα graph databases και επιδεικνύουν τις σχέσεις που έχουν μεταξύ τους, δύο ή περισσότεροι κόμβοι. Πρέπει ακόμη να γνωρίζουμε, πως οι συνδέσεις αυτές δε δείχνουν μόνο προς μία κατεύθυνση (μονόδρομη), αλλά άλλοτε έχουν και αμφίδρομη κατεύθυνση (αμφίδρομη). Τέλος, οι ιδιότητες-(properties) συμπεριλαμβάνουν περιγραφικές πληροφορίες σε σχέση με τους κόμβους. Σε ορισμένες όμως περιπτώσεις, ακόμη και οι άκρες έχουν ιδιότητες-(properties).

Η σημασία που έχει μία τέτοια βάση δεδομένων είναι υψηλή, καθώς τα γραφήματα είναι καλά στο χειρισμό σχέσεων και ορισμένες βάσεις δεδομένων αποθηκεύουν δεδομένα κατά αυτόν τον τρόπο, δηλαδή με τη μορφή γραφήματος.



2.2 ΓΙΑΤΙ ΝΑ ΕΠΙΛΕΞΕΙ ΚΑΝΕΙΣ ΤΙΣ Β.Δ.Γ. (ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ΓΡΑΦΩΝ) ;

Τον τελευταίο καιρό, τον πρωταγωνιστικό ρόλο έχουν τα Big Data με την εμφάνισή τους, όπου καθιστούσαν δύσκολη τη διαχείριση μεγάλου όγκου δεδομένων, εμπλεκόμενα σε εφαρμογές, στις οποίες αποθηκεύονται μεγάλα σύνολα διασυνδεδεμένων δεδομένων. Παράλληλα όμως, απαιτούνταν και η αξιοποίηση των σχέσεων αυτών μεταξύ τους.

Τα προαναφερθέν συνδεδεμένα δεδομένα, αποτελούν δεδομένα από τα οποία πρέπει εμείς οι χρήστες να κατανοήσουμε τις μεταξύ τους σχέσεις. Για να επιτευχθεί όμως αυτό, χρειάζεται να ορίζονται και να ονομάζονται αυτές οι σχέσεις, σε μία βάση.

Όσον αφορά τη σωστή και αποτελεσματική διαχείριση τέτοιων δεδομένων, πρέπει ακόμη να διευθετήσουμε και να προσδιορίσουμε τη σημασιολογία των σχέσεων που συνδέουν οντότητες, αλλά συγχρόνως και το βάρος μιας σχέσης.

Έχουμε καταλήξει στο συμπέρασμα, πως οι Β.Δ.Γ. αντιστοιχούν στον βέλτιστο τρόπο διαχείρισης συνδεδεμένων δεδομένων, για οποιουδήποτε μεγέθους δεδομένα.

Τα graph databases είναι τα καταλληλότερα για την αποτύπωση δεδομένων, με τη μορφή **ιεραρχίας**, ή για εφαρμογές που χρησιμοποιούν περισσότερο τις σχέσεις μεταξύ δεδομένων . Και αυτό, γιατί, αντικαθιστούν σπάταλες λειτουργίες (πχ. αναδρομικά joins), με πιο αποτελεσματικές διασχίσεις γράφων (traversals).

Έτσι, επιτυγχάνεται η αναπαράσταση των εμπλεκόμενων δεδομένων, άκρως ρεαλιστικά (όπως στην πραγματικότητα).

Η λογική που ακολουθεί ένα τέτοιο σύστημα, διαφέρει από αυτήν των συστημάτων αποθήκευσης. Για την εμπέδωση αυτής της λογικής, μπορούμε να προχωρήσουμε στους περιορισμούς των Σχεσιακών Βάσεων (Relational Databases) και των NoSQL Databases στη διαχείριση δεδομένων.

Περιορισμοί Σχεσιακού Μοντέλου

- Δεν είναι δυνατή η αποδοτική διαχείριση των συνδέσεων μεταξύ των δεδομένων από τις Σχεσιακές Βάσεις Δεδομένων. Πιο αναλυτικά, τα δεδομένα διαιρούνται και αποθηκεύονται σε ασύνδετες συλλογές δεδομένων. Η σύνδεση δεδομένων επιτυγχάνεται με τη χρήση ξένων κλειδιών (foreign keys) ή με χρήση πινάκων (join table), αλλά έτσι επιβαρύνεται η απόδοση του συστήματος και δυσχεραίνει τη διαχείριση πολυσύνθετων ερωτημάτων.
Σε αντίθεση με τις βάσεις δεδομένων γράφων, οι συνδέσεις δεδομένων αποθηκεύονται όπως είναι ως σχέσεις στη βάση δεδομένων.

- Ο κυριότερος λόγος για να επιλέξει κανείς τις Βάσεις Δεδομένων Γράφων, είναι η μεγάλη απόδοση που φέρουν(παραμένει σχεδόν σταθερή), κατά τη διαχείριση διασυνδεδεμένων δεδομένων.
Τα ερωτήματα προς τις Β.Δ.Γ., αφορούν ένα ειδικευμένο κομμάτι του γράφου.

Σαν αποτέλεσμα, παρατηρούμε ότι ο χρόνος για να εκτελεστεί κάθε ερώτημα είναι ανάλογο του μεγέθους του γράφου που θα ασχοληθεί με το συγκεκριμένο ερώτημα και δεν αφορά το συνολικό μέγεθος του γράφου.

➢ Βασιζόμενοι στο νέο τους βιβλίο, οι Jonas Partner και Aleksa Vukotic, επιχείρησαν να πειραματιστούν με μια θεωρία βασιζόμενη στη διαφορά της ταχύτητας απόκρισης των Β.Δ.Γ. σε σχέση με τα σχεσιακά συστήματα, στη διέλευση γραφήματος.
Στο πείραμα αυτό, υπάρχει ένα κοινωνικό δίκτυο, στο οποίο γίνεται εύρεση όλων των φίλων, των φίλων ενός χρήστη, σε βάθος 5 επιπέδων. Το συγκεκριμένο κοινωνικό δίκτυο αποτελείται από 1 εκατομμύρια ανθρώπους και ο καθένας από αυτούς έχει 50 φίλους περίπου. Τα αποτελέσματα φαίνονται στο διάγραμμα παρακάτω:

Depth	Execution Time – MySQL	Execution Time –Neo4j
2	0.016	0.010
3	30.267	0.168
4	1,543.505	1.359
5	Not Finished in 1 Hour	2.132

Για τα απλά ερωτήματα φίλων φίλων (ερώτημα βάθους 2), το Neo4j είναι 60% ταχύτερο από το MySQL.
Για φίλους φίλων φίλων (ερώτημα βάθους 3), το Neo είναι 180 φορές πιο γρήγορο.
Και για το ερώτημα βάθους τέσσερα, το Neo4j είναι 1.135 φορές πιο γρήγορο.
Και η MySQL απλώς δυσκολεύεται στο ερώτημα βάθους 5.
Τα αποτελέσματα είναι συναρπαστικά.

➢ Άλλο ένα προτέρημα των Β.Δ.Γ., είναι η ευελιξία που δίδουν για τη σταδιακή τυχόν ανάπτυξη εφαρμογών από άλλους, με βάση τις απαιτήσεις που εμφανίζονται διαρκώς κατά την εκπόνηση μιας εφαρμογής.
Αυτό συμβαίνει, καθώς υπάρχει η δυνατότητα προσθήκης νέων τύπων σχέσεων ή κόμβων, στον ίδιο γράφο, χωρίς να επηρεάζεται η εφαρμογή ή τα ερωτήματα. Κι έτσι, επιτυγχάνεται ταυτόχρονα η σταδιακή εξέλιξη του μοντέλου και η προσθήκη εξτρά παραμέτρων.

2.3 ΠΟΙΑ ΤΑ ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΚΑΙ ΤΑ ΜΕΙΟΝΕΚΤΗΜΑΤΑ ΤΟΥΣ ;

ΠΛΕΟΝΕΚΤΗΜΑΤΑ

- ✓ Οι βάσεις δεδομένων γράφων διαχειρίζονται τις σχέσεις με τον ίδιο βαθμό που διαχειρίζονται και τα δεδομένα, αποφεύγοντας έτσι την ανάγκη για σύνδεση των δεδομένων μέσω ιδιοτήτων, όπως είναι για παράδειγμα τα foreign keys.

- ✓ Ακολουθούν το ACID μοντέλο συναλλαγών και είναι σχεδιασμένες με στόχο την διαθεσιμότητα και την ακεραιότητα συναλλαγών.

- ✓ Οι Β.Δ.Γ. αποτελούνται από κορυφές και άκρες. Οι κορυφές αντιπροσωπεύουν τις οντότητες του μοντέλου δεδομένων ενώ οι άκρες τις μεταξύ τους σχέσεις. Η δημιουργία τους και η σύνθεσή τους σε συνδεδεμένες δομές, επιτρέπουν τη μοντελοποίηση απλών και πιο περίπλοκων μοντέλων, τα οποία δεδομένα, μένουν απαράλλακτα στη μορφή τους όπως και στην πραγματικότητα.

- ✓ Η απόδοση των ερωτημάτων φαίνεται να είναι μερικές κλάσεις καλύτερες από το Relational DataBase Management System (RDBMS) ή NoSQL. Όσο δηλαδή τα δεδομένα αυξάνονται, η απόδοση πέφτει. Ακόμη, οι διασχίσεις των γραφημάτων εντοπίζονται μόνο σε ένα τμήμα του γραφήματος και για αυτό ο χρόνος εκτέλεσης είναι ανάλογος με τον αριθμό τον κόμβων που επισκέφτηκε. Δεν έχει σχέση με τη συνολική ποσότητα των αποθηκευμένων δεδομένων. Οπότε, ακόμα και με σημαντική αύξηση των δεδομένων μας, η απόδοση του ερωτήματος παραμένει σταθερή καθ' όλη τη χρονική διάρκεια ασχολίας μας με αυτό.

- ✓ Εξαιτίας των αυξανόμενων αναγκών των επιχειρήσεων, ως προς τη εύρυθμη λειτουργία τους, οι ασχολούμενοι προγραμματιστές χρειάζονται ένα δυνατό εργαλείο, το οποίο θα τους επιτρέπει να εξετάζουν και να τροποποιούν σταδιακά το μοντέλο όπου και ασχολούνται, σε οποιοδήποτε σημείο.

  Σε αυτό το πρόβλημα, εμφανίζονται οι Β.Δ.Γ., όπου και επιτρέπουν να προστίθενται σχέσεις, τύποι και ιδιότητες, δίχως αλλαγές στη ήδη υπάρχοντα ερωτήματα και με αυτό καταλήγουμε να έχουμε ένα μοντέλο όπου μπορεί να εφαρμοστεί πιο περίπλοκη αναζήτηση. Κατά αυτόν τον τρόπο, μας επιτρέπεται να μην σκεφτόμαστε τις πολλαπλές μετεγκαταστάσεις, καθώς η εφαρμογή τους μπορεί να γίνει και χωρίς τη λήψη της βάσης δεδομένων εκτός σύνδεσης. Επιτρέπει δηλαδή σε ομάδες προγραμματιστών να

παραδίδουν το λογισμικό ταχύτερα, δίνοντας σημασία μόνο στον τομέα και όχι στην υποδομή και επικοινωνία.

- ✓ Για το λόγο ότι οι Β.Δ.Γ. δεν περιέχουν κάποιο σχήμα, αυτό δίδεται από την ίδια την εφαρμογή κι έτσι εξασφαλίζεται η σωστή και επικαιροποιημένη εμφάνιση του σχήματος, ενώ παράλληλα επιλέγει και την καλύτερη σχεδιαστική σκέψη από τους προγραμματιστές.

- ✓ Επιτρέπει στους προγραμματιστές να επικεντρωθούν στο αναπτυξιακό κομμάτι και όχι στο σχεδιαστικό, εξοικονομώντας έτσι χρόνο. Ένα χαρακτηριστικό της Β.Δ.Γ. είναι η σχέση που έχει με τους πίνακες. Ένα μοντέλο δεδομένων μπορεί να δημιουργηθεί σε έναν πίνακα χωρίς να δημιουργηθεί περαιτέρω προβληματισμός για να μετασχηματιστεί σε έναν σύνολο πινάκων.

ΜΕΙΟΝΕΚΤΗΜΑΤΑ

- ✓ Σημαντικό μειονέκτημα των Β.Δ.Γ. αποτελεί η δυσκολία επίτευξης της οριζόντιας κλιμάκωσης, επειδή δεν υπάρχει κάποιο διακριτό σημείο διαχωρισμού σε συνδεδεμένο γράφο.

2.4 ΜΕ ΠΟΙΟΥΣ ΤΡΟΠΟΥΣ ΑΠΟΘΗΚΕΥΟΝΤΑΙ ΚΑΙ ΕΠΕΞΕΡΓΑΖΟΝΤΑΙ ΤΑ ΓΡΑΦΗΜΑΤΑ ;

- Δύο από τις θεμελιώδης ιδιότητες με τις οποίες θα ασχοληθούμε είναι:
  - Ο τρόπος αποθήκευσης
    Πώς αποθηκεύονται τα δεδομένα και πώς αντιπροσωπεύονται όταν ανακτώνται
  - Ο τρόπος επεξεργασίας των ερωτήσεων σε έναν γράφο
    Πώς ανακτώνται και τροποποιούνται τα δεδομένα

- Οποιαδήποτε από τις προαναφερθέντες ιδιότητες, μπορεί να είναι native ή non-native.
- Ανάλογα το πως θα αποθηκευτούν τα δεδομένα και πως θα επεξεργαστούν οι ερωτήσεις για δεδομένα γραφήματος (graph data), καθορίζουν και το είδος αποθήκευσης της Βάσης Δεδομένων.

Εγγενής αποθήκευση γραφημάτων - (native)

→ Η αποθήκευση γραφήματος αναφέρεται συνήθως στη δομή της βάσης δεδομένων που εμπεριέχονται δεδομένα γραφήματος. Σε περίπτωση που έχει κατασκευαστεί ειδικά για την αποθήκευση δεδομένων που μοιάζουν με graph, τότε εννοούμε την εγγενή αποθήκευση γραφημάτων (native graph storing).

→ Οι Β.Δ.Γ. με εγγενή αποθήκευση γραφημάτων είναι βελτιστοποιημένες για γραφήματα, εξασφαλίζοντας ότι τα δεδομένα αποθηκεύονται αποτελεσματικά, γράφοντας κόμβους και σχέσεις κοντά το ένα στον άλλο. Έχουν σχεδιαστεί για να μεγιστοποιούν την ταχύτητα διέλευσης κατά τη διάρκεια αυθαίρετων αλγορίθμων γραφημάτων.

→ Οι συνδέσεις των δεδομένων αποθηκεύονται ακριβώς όπως είναι στη βάση δεδομένων. Στο σύνολο οι συνδέσεις που υπάρχουν στον πραγματικό κόσμο, αποθηκεύονται ως σχέσεις μεταξύ των δεδομένων της βάσης δεδομένων.

Μη εγγενής αποθήκευση γραφημάτων - (non-native)

→ Στα Relational συστήματα τα δεδομένα διασπώνται και αποθηκεύονται σε συλλογές δεδομένων οι οποίες είναι ασύνδετες μεταξύ τους. Η σύνδεση των δεδομένων γίνεται κατά την εκτέλεση των ερωτημάτων που στέλνονται στην database, με χρήση πινάκων (join tables) ή με χρήση ξένων κλειδιών (foreign keys).

→ Παρόμοια συμβαίνει και στις NoSQL βάσεις δεδομένων, όπου διατηρούνται μη συνδεδεμένα σύνολα εγγράφων στηλών ή ζεύγη κλειδιού / τιμής - (key / value).

→ Οι μη εγγενείς Β.Δ.Γ. δεν είναι βελτιστοποιημένες ,καθώς δεν έχουν βρεθεί οι βέλτιστες παράμετροι για την αποθήκευση γραφημάτων, άρα οι αλγόριθμοι που χρησιμοποιούνται στην εγγραφή δεδομένων μπορεί να αποθηκεύουν κόμβους και σχέσεις παντού.

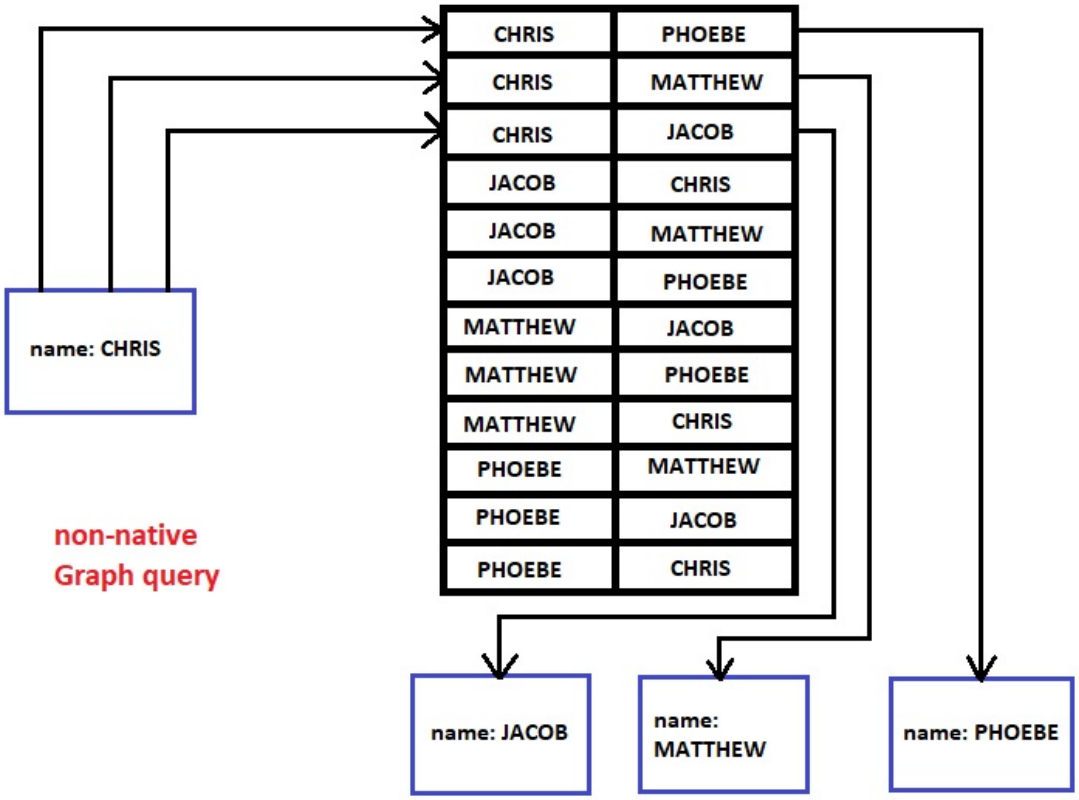
→ Προκαλούνται έτσι, προβλήματα στην απόδοση τη στιγμή που ανακτώνται τα δεδομένα (retrieve data), αφού όλοι οι κόμβοι και σχέσεις χρησιμοποιούνται μαζί για κάθε ερώτημα.

Εγγενής επεξεργασία ερωτήσεων

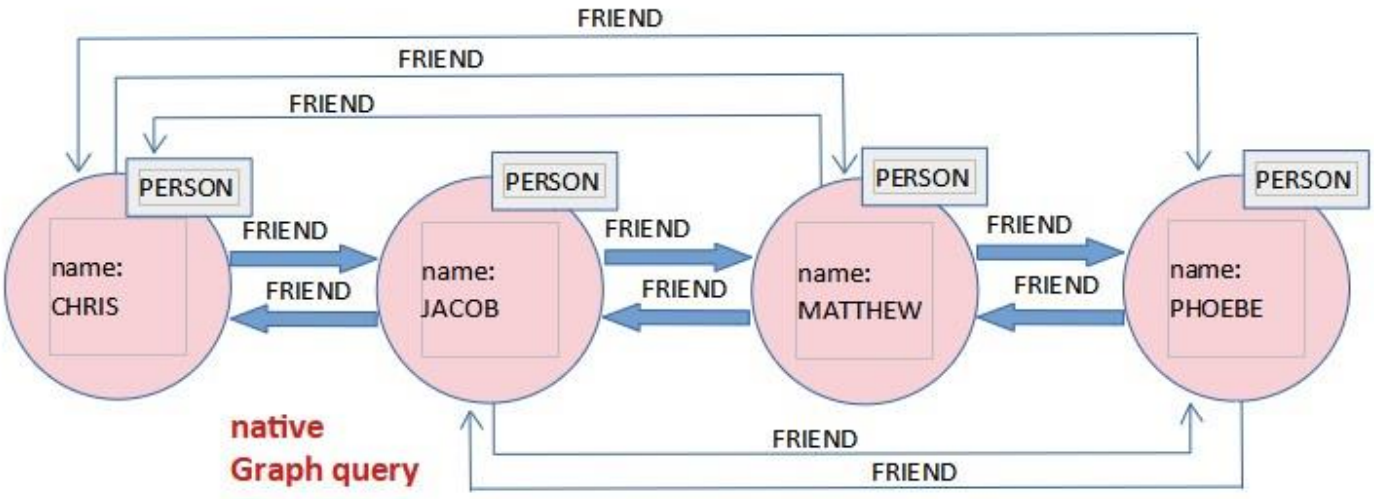
→ Όταν υπάρχει η ιδιότητα γειτνίασης χωρίς ευρετήριο (index-free adjacency), τότε λέμε ότι μία βάση δεδομένων υποστηρίζει εγγενή επεξεργασία γράφου (native graph processing).
→ Στη φυσική μνήμη υπάρχουν άμεσες αναφορές από έναν κόμβο προς έναν γειτονικό κόμβο. Δηλαδή, κάθε κόμβος λειτουργεί σαν μια ένδειξη για τους γειτονικούς όμοιούς του και οι χρόνοι είναι ανεξάρτητοι από το τελικό μέγεθος του γράφου.
→ Η index-free adjacency επιταχύνει την επεξεργασία αποθήκευσης πράγμα που σημαίνει ότι κάθε κόμβος αποθηκεύεται απευθείας στους διπλανούς κόμβους και τις σχέσεις του. Αυτά συμβαίνουν κατά το χρόνο εγγραφής.
→ Κατά το χρόνο ανάγνωσης ή αλλιώς επεξεργασίας ερωτημάτων, η γειτνίαση χωρίς ευρετήρια δίδει αστραπιαία ανάκτηση δίχως τη χρήση ευρετηρίων.

Μη εγγενής επεξεργασία ερωτήσεων

→ Στην περίπτωση αυτή, οι μη εγγενείς Β.Δ.Γ. χρησιμοποιούν δείκτες-(καθολικά ευρετήρια) για την μεταξύ σύνδεση των κόμβων. Σε κάθε διάβαση προστίθεται κι άλλο επίπεδο από τα ευρετήρια, με αποτέλεσμα να επιβραδύνεται η επεξεργασία σε μεγάλο βαθμό και παράλληλα αυξάνεται το υπολογιστικό κόστος για κάθε διάσχιση. Οπότε αυτή η μέθοδος είναι και πιο δαπανηρή.
→ Για να μπορέσουμε να αντιστρέψουμε την κατεύθυνση μιας διέλευσης, χρειάζεται να δημιουργήσουμε αντιστροφή του ευρετηρίου αναζήτησης για κάθε διέλευση ή να πραγματοποιήσουμε αναζήτηση ωμής βίας μέσω του αρχικού ευρετηρίου, διαδικασία σχεδόν απίθανη με μη εγγενή επεξεργασία ερωτήσεων.



Non-native VS Native Graph Query



2.5 ΒΆΣΕΙΣ ΔΕΔΟΜΈΝΩΝ ΓΡΆΦΩΝ

Πώς εξελίχθηκαν:

- Οι Β.Δ.Γ. πρωτοεμφανίστηκαν παρέα με το hypertext και ειδικότερα την περίοδο που αυξανόταν η χρήση εφαρμογών υπερκειμένου και Γεωγραφικών Συστημάτων, στα οποία η διασύνδεση δεδομένων είναι σημαντική.

- Αυτές οι Β.Δ.Γ. χρησιμοποιούνταν παράλληλα με αντικειμενοστραφή μοντέλα Β.Δ. προσπαθώντας να βρουν λύσεις σε διάφορους περιορισμούς που είχαν τα πιο παραδοσιακά συστήματα στη μοντελοποίηση δεδομένων.

- Το πεδίο αυτό δείχνει ιδιαίτερα χρήσιμο σε μοντέρνες εφαρμογές, καθώς εφαρμόζεται καλύτερα σε απαιτητικές περιπτώσεις με πολύπλοκα δεδομένα.

- Οι γράφοι χρησιμοποιούνται ολοένα και περισσότερο με την πάροδο του χρόνου, ιδιαίτερα τα τελευταία χρόνια, μέσω χρηστών του Διαδικτύου / Internet.

Προβλήματα – Μεταρρυθμίσεις – Βελτιώσεις

- Ένα ιδιαίτερο πρόβλημα που φέρουν οι γράφοι είναι και η **δρομολόγηση (routing)**. Μετά από εκτενής έρευνα, έχουν δοθεί βοηθητικοί αλγόριθμοι και εν συνεχεία λύσεις σε εφαρμογές, όπου απαιτούνταν. Για παράδειγμα, έχουμε τη δρομολόγηση πακέτων στο δίκτυο σε πραγματικό χρόνο, υπολογίζοντας πληροφορίες που έχουν συλλεχθεί διεθνώς.

- Τα γραφήματα εμφανίζονται και χρησιμοποιούνται στα **κοινωνικά δίκτυα (social networks)**, βοηθούν στη σύνδεση χρηστών και για το Αν θα έχουν πρόσβαση σε δεδομένα ή όχι. Όπως για παράδειγμα το Twitter και το Facebook χρησιμοποιούν την αναζήτηση γραφήματος όπου βοηθάει τους χρήστες με τις αναζητήσεις τους.

- Ακόμη ένα θέμα που αντιμετωπίζουν οι γράφοι, είναι αυτό της **σύστασης (Recommendation)** που μπορεί να λυθεί με χρήση των Β.Δ.Γ.

- Πέρα από τη χρήση Β.Δ.Γ. για τη λύση απλών προβλημάτων γράφων, οι εταιρείες τα χρησιμοποιούν και για απλούστερες ενέργειες, όπως αναζήτηση στο Διαδίκτυο. (π.χ. η Google χρησιμοποιεί (Knowledge Graph) έτσι ώστε να μας εμφανίσει αποτελέσματα σχετικά με αυτό που ψάχνουμε. Η σχέση μεταξύ όρου αναζήτησης και αποτελέσματος πρέπει να έχουν μεγάλη ομοιότητα -δηλαδή να έχουν μία άμεση σχέση- και εμφανίζονται σε φθίνουσα σειρά, ανάλογα τη σχετικότητά τους.

◆ Ακόμα και σε ιατρικά δεδομένα χρησιμοποιούνται γραφήματα. Αυτά τα δεδομένα είναι κατά πολύ διασυνδεδεμένα κι έτσι οι χρήστες/επιβλέποντες αυτών μπορούν να εκμεταλλευτούν τη χρήση των Β.Δ.Γ. Στην εποχή μας, οι εταιρείες χρησιμοποιούν τέτοιες βάσεις, τόσο για την καταγραφή και αποθήκευση πληροφοριών των φαρμάκων, όσο και για την ανακάλυψή τους.

◆ Η αποθήκευση των οντολογιών σε βάσεις δεδομένων γραφημάτων βρίσκει γρήγορα εφαρμογές στη μηχανική μάθηση - (machine learning) και στην ανάλυση δεδομένων (analytics).

◆ Πολλές είναι οι εταιρείες που κάνουν χρήση Β.Δ.Γ. σε τομείς όπως για παράδειγμα η παροχή ενέργειας και η μεταφορά.

Τομείς

◆ Πλήθος εφαρμογών συνδυάζονται και ελέγχονται από Β.Δ.Γ. και υπόσχονται σημαντική αύξηση στην απόδοση και στον καλύτερο τρόπο διαχείρισής τους. Μερικές από αυτές είναι:

- Social Network (Κοινωνικά Δίκτυα)
- Fraud Detection (Ανίχνευση για απάτη)
- Master Data Management (Διαχείριση κρίσιμων δεδομένων)
- Real Time Recommendation Engine (Συστήματα συστάσεων σε πραγματικό χρόνο)
- Graph Based Search (Αναζήτηση με χρήση δομών γράφων)

◆ Η Neo4j είναι από τις πιο γνωστές Β.Δ.Γ. (Βάσεις Δεδομένων Γράφων)

2.6 ΓΝΩΣΤΕΣ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ΓΡΑΦΩΝ (Β.Δ.Γ.) :

- Redis, Riak

  Data Model:
  - Global key-value mapping
  - Big Scalable HashMap

  Pros:
  - Simple Data Model
  - Scalable

  Cons:
  - Create your own foreign keys

- Poor for complex data

- CouchDB, MongoDB

  Data Model:
  - A collection of documents
  - A document is a key value collection

  Pros:
  - Simple, Powerful data model
  - Scalable

  Cons:
  - Poor for interconnected data
  - Query limited to keys and indexes

- HBase, HyperTable, Cassandra

  Data Model:
  - A big table, with column families

  Pros:
  - Supports Semi-Structured Data
  - Naturally Indexed (columns)
  - Scalable

  Cons:
  - Poor for interconnected data

- OrientDB

  Data Model:
  - Nodes and Relations

  Pros:
  - Powerful data model
  - Easy to query

  Cons:
  - Sharding
  - Scales UP reasonably well

2.7 ΕΙΣΑΓΩΓΉ ΣΤΟ ΒΑΣΙΚΌ ΠΡΌΓΡΑΜΜΑ NEO4J

2.7.1 ΠΟΤΕ ΚΑΙ ΠΟΥ ΠΡΩΤΟΕΜΦΑΝΙΣΤΗΚΕ Η NEO4J

Αρχικά, η ανακάλυψη της Neo4j αποτελεί και τη μεγαλύτερη επένδυση στην ιστορία της βάσης δεδομένων ως μία Open-Source, NoSQL βάση δεδομένων τύπου γράφου.

Η Neo Technology Inc. ευθύνεται για την ανάπτυξή της και πλέον χρησιμοποιείται από χιλιάδες οργανισμούς όπως Airbus, Comcast, Ebay, NASA, UBS, Walmart και άλλα, σε κρίσιμες εφαρμογές παραγωγής.

Η ανάπτυξή της χρονολογείται πίσω στο 2003, αλλά δε δημοσιεύτηκε πριν το 2007 για το κοινό. Ο πηγαίος κώδικας (source code) είναι γραμμένος σε Java και σε Scala. Η πρώτη κυκλοφορία της εφαρμογής ήταν στις αρχές του έτους 2010.

Πέρα από τη χρήση της Neo4j ως μιας ενσωματωμένης βάσης δεδομένων σε μια εφαρμογή, μπορεί να λειτουργήσει και αυτόνομα ως Server σε συνεργασία με τη REST διεπαφή, για να μπορεί να συνεργάζεται με περιβάλλοντα που χρησιμοποιούν τις εξής γλώσσες προγραμματισμού PHP, .NET, Javascript.

Η Neo4j διαθέτει δύο ειδών αδειοδοτήσεις, μία δωρεάν έκδοση για την κοινότητα (GPL Style - Community Edition) και αντίστοιχα μία εμπορική έκδοση (Enterprise Edition), η οποία είναι επί πληρωμή.

- Η έκδοση επί πληρωμή απευθύνεται σε εφαρμογές μεσαίας κλίμακας σε προηγμένα περιβάλλοντα ανάπτυξης ή παραγωγής, ενώ η δωρεάν έκδοση είναι ιδανική για μικρά αναπτυξιακά έργα, μάθηση, πειραματισμό και δημιουργία πρωτοτύπων. Επιπρόσθετα, η Enterprise Edition προσφέρει μερικές πρόσθετες εταιρικές απαιτήσεις όπως backup δεδομένων, clustering και failover ιδιότητες.

2.7.2 ACID ΣΥΝΑΛΛΑΓΕΣ

Με απώτερο σκοπό να καταφέρουμε και να διατηρήσουμε την ακεραιότητα των δεδομένων και να ασφαλιστεί η συναλλακτική συμπεριφορά, το Neo4j συνοδεύεται μαζί με κάποιες ιδιότητες όπως τις λέμε ACID οι οποίες είναι:

1. Atomicity / Ατομικότητα
   Εάν οποιοδήποτε μέρος μιας συναλλαγής αποτύχει, η κατάσταση της βάσης δεδομένων παραμένει αμετάβλητη.

2. Consistency / Συνέπεια
   Οποιαδήποτε συναλλαγή θα αφήσει τη βάση δεδομένων σε συνεπή κατάσταση.

3. Isolation / Απομόνωση
   Κατά τη διάρκεια μιας συναλλαγής, δεν είναι δυνατή η πρόσβαση στα τροποποιημένα δεδομένα από άλλες λειτουργίες.

4. Durability / Ανθεκτικότητα
   Το DBMS μπορεί πάντα να ανακτήσει τα αποτελέσματα μιας δεσμευμένης συναλλαγής.

Για να γίνει σωστά η αποθήκευση μιας σύνδεσης μεταξύ 2 κόμβων, δεν αρκεί μόνο μια εγγραφή σχέσης, αλλά πρέπει επίσης να ενημερώσουμε τους κόμβους στην αρχή και στο τέλος της σχέσης. Σε περίπτωση αποτυχίας ανάμεσα σε αυτές τις λειτουργίες εγγραφής, το γράφημα θα καταστραφεί.

Για να μην παρατηρηθεί κάποια αλλοίωση σε αυτά τα γραφήματα με το χρόνο, χρειαζόμαστε την πλήρη υποστήριξη ACID συναλλαγών.

Η Neo4j συνεργάζεται πλήρως με τις λειτουργίες της ACID:

- Χάρη σε αυτή τη συνεργασία, επιτρέπεται η ενημέρωση ενός μέρους του γράφου σε κάποιο απομονωμένο περιβάλλον, μη αποκαλύπτοντας τις αλλαγές σε άλλες διεργασίες μέχρι το πέρας της συναλλαγής.

- Εάν οι εμπλεκόμενες συναλλαγές δοκιμάσουν να τροποποιήσουν τα ίδια δεδομένα, ο kernel της Neo4j θα τις συγχρονίσει.

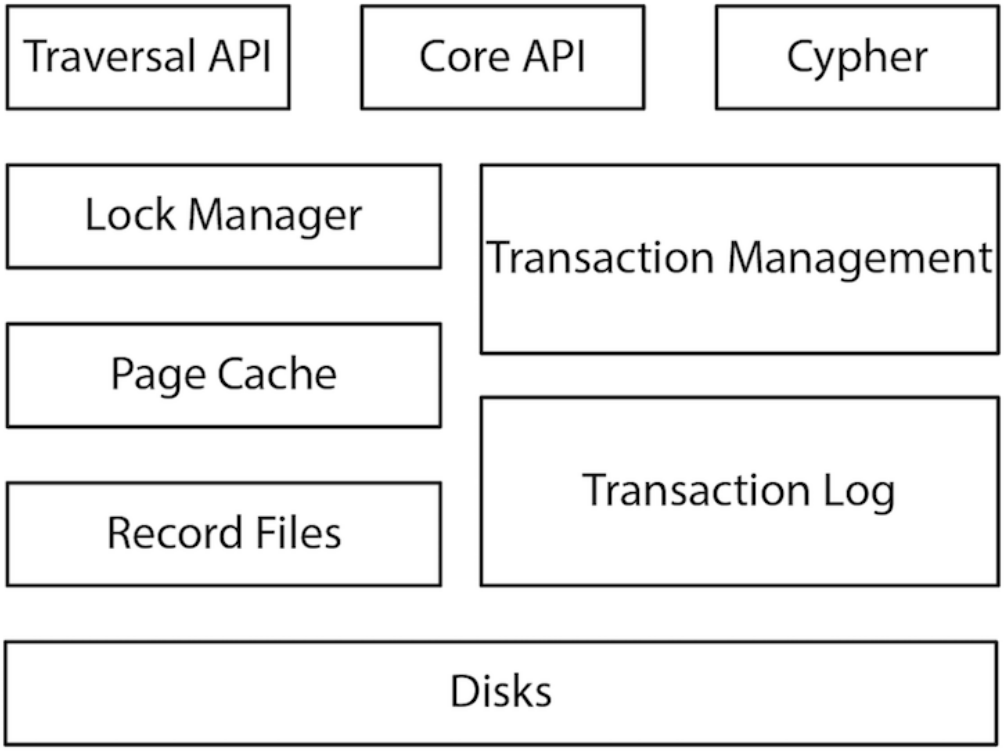
- Ακόμη, εάν προκύψει κάποια αδιέξοδος στην αλληλεξάρτηση μεταξύ των συναλλαγών, η ενέργεια αυτή θα εντοπιστεί και θα αποφευχθεί.

2.7.3 ΕΓΓΕΝΗΣ ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ ΓΡΑΦΩΝ

Αρχικά να επισημάνουμε πως μία βάση δεδομένων γραφήματος έχει εγγενείς δυνατότητες επεξεργασίας εάν χρησιμοποιεί γειτνίαση χωρίς ευρετήριο. Αυτό σημαίνει ότι κάθε κόμβος αναφέρεται απευθείας στους παρακείμενους κόμβους του, λειτουργώντας ως μικροευρετήριο για όλους τους κοντινούς κόμβους.

Η Neo4j – μια εγγενής βάση δεδομένων γραφημάτων με εγγενής αποθήκευση γραφημάτων είναι δομημένη με τέτοιο τρόπο ώστε κάθε επίπεδο αυτής της αρχιτεκτονικής – από τη γλώσσα ερωτημάτων Cypher έως τα αρχεία στο δίσκο –

να έχει βελτιστοποιηθεί για την αποθήκευση δεδομένων γραφήματος και κανένα τμήμα της αρχιτεκτονικής αυτής δεν αντικαθίσταται με τμήματα από άλλες τεχνολογίες Β.Δ. χωρίς γραφήματα.



- Η Neo4j υποστηρίζει εγγενής αποθήκευση γράφων (native graph storage) το οποίο έχει ως αποτέλεσμα την υψηλή αποδοτικότητα στα ερωτήματα διάσχισης (traversal).

- Στη Neo4j οι ιδιότητες αποθηκεύονται ως μια συνδεδεμένη λίστα εγγραφών ιδιοτήτων, καθεμία από τις οποίες περιέχει ένα κλειδί και μια τιμή και δείχνει την επόμενη ιδιότητα. Κάθε κόμβος και σχέση αναφέρεται στην πρώτη του εγγραφή ιδιοτήτων. Οι κόμβοι αναφέρονται επίσης στην πρώτη σχέση στην αλυσίδα σχέσεών τους. Κάθε σχέση αναφέρεται στον κόμβο έναρξης και λήξης της.

2.7.4 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΚΑΝΟΝΤΑΣ ΧΡΗΣΗ ΤΟΥ ΝΕΟ4J

1. Easy to Use / Εύκολο προς χρήση

Εμπεριέχει την Cypher, την πιο ισχυρή και παραγωγική γλώσσα ερωτημάτων γραφήματος στον κόσμο και το εγγενές Java API για τη σύνταξη προσαρμοσμένων επεκτάσεων ειδικού σκοπού. Υπάρχουν πληθώρα από API και προγράμματα οδήγησης για όλες τις κύριες γλώσσες που βελτιώνουν την παραγωγικότητα του προγραμματιστή μέσω του διαισθητικού περιβάλλοντος διεπαφής.

2. Great Availability / Μεγάλη Διαθεσιμότητα

Χρησιμοποιείται από πολλά και μεγάλα στελέχη, καθώς επίσης και από μεγάλες επιχειρήσεις σε εφαρμογές πραγματικού χρόνο.

3. Adaptive Data Model / Προσαρμοστικό Μοντέλο Δεδομένων

Παρέχει ένα ισχυρό μοντέλο δεδομένων που μπορεί εύκολα να προσαρμοστεί σύμφωνα με τις διάφορες απαιτήσεις των εφαρμογών.

4. Επεκτασιμότητα ανάγνωσης και εγγραφής υψηλής απόδοσης

Το Neo4j προσφέρει ταχεία απόδοση ανάγνωσης και εγγραφής, ενώ παράλληλα προστατεύει την ακεραιότητα των δεδομένων. Είναι η μόνη βάση δεδομένων γραφημάτων ισχυρής επιχείρησης που συνδυάζει εγγενή αποθήκευση γραφημάτων, κλιμακούμενη αρχιτεκτονική βελτιστοποιημένη για ταχύτητα και συμμόρφωση με ACID για να διασφαλίζεται η προβλεψιμότητα των ερωτημάτων που βασίζονται σε σχέσεις.

5. Easy Recovery / Εύκολη ανάκτηση

Αναπαριστάνονται και ανακτώνται εύκολα διασυνδεδεμένα δεδομένα γρηγορότερα σε σχέση με άλλες Β.Δ.

6. Real-Time Results / Παρέχει αποτελέσματα σε πραγματικό χρόνο

Insights σε πραγματικό χρόνο.

7. No joins / Χωρίς συνδέσεις

Χρησιμοποιώντας το Neo4j, δεν απαιτούνται σύνθετες συνδέσεις για την ανάκτηση συνδεδεμένων/σχετικών δεδομένων, καθώς είναι πολύ εύκολο να ανακτηθούν οι λεπτομέρειες του παρακείμενου κόμβου ή της σχέσης του χωρίς συνδέσεις ή ευρετήρια.

8. Fast Data Loading / Γρήγορη φόρτωση δεδομένων

Αστραπιαία ταχύτητα φόρτωσης τεράστιων μεγεθών δεδομένων, με μικρή κατανάλωση μνήμης.

2.7.5 ΟΡΙΣΜΕΝΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ NEO4J

- **Μοντέλο Δεδομένων**
  Neo4j → Μοντέλο γραφήματος εγγενούς ιδιότητας
  Το γράφημα περιέχει κόμβους (entities), οι οποίοι συνδέονται μεταξύ τους (εμφανίζονται με σχέσεις – relationships). Οι κόμβοι και οι σχέσεις αποθηκεύουν δεδομένα σε ζευγάρια key-value, τις λεγόμενες ιδιότητες.

- Δεν είναι απαραίτητο να ακολουθηθεί κάποιο συγκεκριμένο και σταθερό σχήμα. Είναι εύκολη η προσθαφαίρεση ιδιοτήτων ανάλογα με τις απαιτήσεις.

- Ιδιότητες **ACID**

Το Neo4j υποστηρίζει πλήρως τους κανόνες ACID.

- **Extensibility and reliability**
  Ο αριθμός των read/write μπορούν να αυξηθούν, όπως και ο όγκος, χωρίς να επηρεάζεται η ταχύτητα επεξεργασίας ερωτημάτων αλλά ούτε και η ακεραιότητα των δεδομένων.

- **Cypher Query language**

Περιέχεται στη Neo4j μια ευρέως γνωστή και δυναμική γλώσσα ερωτημάτων, η Cypher. Η συγκεκριμένη γλώσσα χρησιμοποιεί ASCII-art για την απεικόνιση γραφημάτων. Η Cypher είναι εύκολο στην εκμάθηση και μπορεί να χρησιμοποιηθεί για τη δημιουργία και την ανάκτηση σχέσεων μεταξύ δεδομένων χωρίς τη χρήση πολύπλοκων ερωτημάτων όπως τα **joins**.

- **Ενσωματωμένη εφαρμογή web**
  Διαθέτει μια ενσωματωμένη εφαρμογή web του Neo4j, η οποία μας επιτρέπει να δημιουργήσουμε και να ρωτήσουμε τα δεδομένα ενός γραφήματος.

- **Ευρετήρια πλήρους κειμένου**
  Υποστηρίζονται ευρετήρια κάνοντας χρήση του Apache Lucene.Τα ευρετήρια πλήρους κειμένου τροφοδοτούνται από τη βιβλιοθήκη ευρετηρίου και αναζήτησης του Apache Lucene και μπορούν να χρησιμοποιηθούν για την ευρετηρίαση κόμβων και σχέσεων με ιδιότητες συμβολοσειράς.

2.7.6 ΤΕΛΙΚΉ ΑΠΌΔΟΣΗ

i. Δεν είναι εύκολο να υπολογίσουμε την απόδοση και να την αριθμήσουμε, διότι εξαρτάται από πολλούς παράγοντες, όπως το hardware, το dataset που χρησιμοποιείται και άλλα.

ii. Έχει την ικανότητα να διαχειρίζεται πλήρως γράφους αρκετών δισεκατομμυρίων κόμβων, σχέσεων και ιδιοτήτων-γνωρισμάτων. Μπορεί να διαβάσει 2000 σχέσεις ανά χιλιοστό του δευτερολέπτου.

iii. Συγκρίνοντας τη Neo4j με την SQL, ο αλγόριθμος εύρεσης της πιο κοντινής διαδρομής Shortest Path δείχνει πολύ πιο γρήγορος στην Neo4j από την SQL κατά 1000 φορές, ακόμα και σε μικρούς γράφους με χιλιάδες κόμβους. Και το συναρπαστικό με αυτούς τους αριθμούς είναι ότι όσο μεγαλώνει το μέγεθος του γράφου, η διαφορά αυξάνεται εκθετικά.

iv. Η διάσχιση του γράφου γίνεται με σταθερή ταχύτητα χωρίς να μας επηρεάζει το μέγεθός του. Δεν περιλαμβάνονται σετ εντολών που μειώνουν την απόδοση, όπως τυχαίνει στις συνενώσεις – joins στα RDBMS (Relation Database Management system).

v. Οι κόμβοι και οι σχέσεις αρχικά εξετάζονται / προσπερνιούνται και επιστρέφουν μόνο όταν ζητηθούν, με αποτέλεσμα την αύξηση της απόδοσης σε μεγάλες και βαθιές διασχίσεις traversals του γράφου.

vi. Η ταχύτητα εγγραφής εξαρτάται σε μεγάλο βαθμό από τον χρόνο αναζήτησης του συστήματος αρχείων και του υλικού. Στη Neo4j το σύστημα αρχείων Ext3 και οι δίσκοι SSD είναι ένας καλός συνδυασμός και έχουν ως αποτέλεσμα ταχύτητες εγγραφής συναλλαγών περίπου 100.000 λειτουργίες ανά δευτερόλεπτο.

2.7.7 ΕΦΑΡΜΟΓΕΣ ΧΡΗΣΗΣ ΤΟΥ NEO4J

Το Neo4j είναι ευρέως διαδεδομένο και χρησιμοποιείται από πολλές και γνωστές επιχειρήσεις και εταιρείες στους περισσότερους κλάδους, ιδιαίτερα στις χρηματοοικονομικές υπηρεσίες, στην κυβέρνηση, στην ενέργεια, στην τεχνολογία, στο λιανικό εμπόριο και λοιπά.

Τελευταίο αλλά μη εξαιρετέο, αξιοποιείται από εφαρμογές κοινωνικής δικτύωσης όπως Facebook, Instagram, Twitter κλπ. Σε πολλές άλλες περιπτώσεις το Neo4j υπερτερεί ως προς τις ικανότητές του.

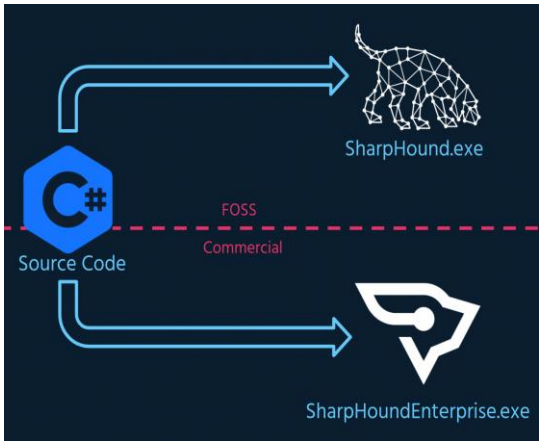
Πού αλλού μπορεί να φανεί χρήσιμο το Neo4j ;

- Διαγράμματα δικτύου
- Διαχείριση πρόσβασης
- Διαχείριση δεδομένων
- Αναζήτηση ψηφιακών στοιχείων βάσει γραφήματος
- Ανίχνευση απάτης
- Κοινωνικά δίκτυα
- Προτάσεις προϊόντων σε πραγματικό χρόνο

2.7.8 BLOODHOUND APPLICATION

Το BloodHound Enterprise είναι μια λύση διαχείρισης διαδρομών επίθεσης που χαρτογραφεί και ποσοτικοποιεί συνεχώς τις διαδρομές επίθεσης της υπηρεσίας καταλόγου Active Directory. Μπορείτε να αφαιρέσετε εκατομμύρια, ακόμη και δισεκατομμύρια Διαδρομές επίθεσης στην υπάρχουσα αρχιτεκτονική σας και να εξαλείψετε τις πιο εύκολες, πιο αξιόπιστες και ελκυστικές τεχνικές του εισβολέα.

Το **BloodHound** είναι μια single page Javascript διαδικτυακή εφαρμογή (web application) που ενσωματώνει τις τεχνολογίες Linkurious και Electron καθώς και την βάση δεδομένων του Neo4j, πάνω στο οποίο είναι χτισμένο, και ο λόγος που είναι ευρέως διαδεδομένο. Το BloodHound χρησιμοποιεί την ιδέα του γραφήματος για να βοηθήσει τους αμυνόμενους και τους εισβολείς να ανακαλύψουν ακούσιες και κρυφές σχέσεις μέσα στο Active Directory. Αναπτύσσει έναν ingestor για να αποκτήσει δεδομένα. Ένας ingestor είναι το **SharpHound** (ο συλλέκτης δεδομένων του Bloodhound, γραμμένος στην C#) με γραμμή εντολών ".exe" ή ένα σενάριο PowerShell που έχει παρόμοια δομή με το ".exe".



Μόλις αναπτυχθεί, ο ingestor συλλέγει όλες τις εγγραφές από το Active Directory αλλά επίσης και από τους χρήστες, τις ομάδες και τους υπολογιστές. Το SharpHound είναι ένα καλό και ισχυρό πρόγραμμα εισαγωγής που αποκαλύπτει τις πληροφορίες των αδειών διαφημίσεων, των ζωντανών περιόδων σύνδεσης και των διαφορετικών εγγραφών μέσω της άδειας ενός τακτικού καταναλωτή. Παράλληλα, παρέχει έγγραφα JSON στη βάση δεδομένων Neo4j, η οποία τα οπτικοποιεί μέσω μιας γραφικής διεπαφής καταναλωτή.

Το γεγονός ότι το πρόγραμμα αυτό ενσωματώνει την Javascript, κάνει compile με την Electron και χρησιμοποιεί την βάση δεδομένων του Neo4j, προσφέρει στο Bloodhound πολλά πλεονεκτήματα, όπως online backup και αδειοδοτημένες επεκτάσεις υψηλής διαθεσιμότητας. Στηριζόμενο στην πανίσχυρη και σύγχρονη βάση δεδομένων του, το BloodHound ψάχνει τις σχέσεις και υπολογίζει την συντομότερη διαδρομή η οποία όμως παίρνει και τον λιγότερο χρόνο μεταξύ δύο αντικειμένων. Αυτός ο υπολογισμός είναι εφικτός μέσω συνδέσμων.

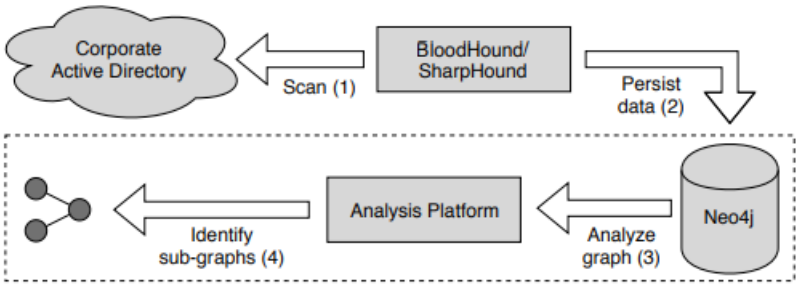


Figure 8: Technical perspective of the investigation process

2.8 ΑΡΧΙΤΕΚΤΟΝΙΚΗ NEO4J

2.8.1 ΣΥΣΤΑΤΙΚΟ ΚΟΜΜΑΤΙ ΤΗΣ NEO4J

Το Neo4j αποτελείται από:

- **Το Neo4j Graph Database:** είναι η κύρια βάση δεδομένων γραφημάτων και υπάρχει για να αποθηκεύει και να ανακτά τα συνδεμένα δεδομένα. Βγαίνει σε δύο εκδόσεις: την Community και την Enterprise. Οι κύριες διαφορές τους είναι ότι η Enterprise περιέχει όλες τις λειτουργίες της Community έκδοσης, αλλά περιέχει επιπλέον εταιρικές απαιτήσεις σαν backup, clustering, και δυνατότητες failover.

- **Neo4j Desktop:** Είναι μια εφαρμογή η οποία διαχειρίζεται τις τοπικές παρουσιάσεις του Neo4j, οι οποίες είναι:
    - Δημιουργία και διαγραφή έργου
    - Ρυθμίσεις εφαρμογής
    - Προφίλ
    - Δημιουργία βάσης δεδομένων γραφημάτων
    - Διαχείριση βάσης δεδομένων

- **Neo4j Brower:** Έχει τις κύριες ικανότητες οπτικοποίησης χρησιμοποιώντας την γλώσσα ερωτημάτων Cypher, και είναι μια διαδικτυακή διεπαφή ενός προγράμματος περιήγησης, υπεύθυνο για την αναζήτηση και την προβολή των δεδομένων στην βάση δεδομένων.

- **Neo4j Bloom:** είναι ένα εργαλείο οπτικοποίησης, όμως για επαγγελματίες χρήστες, και η διαφορά τους είναι ότι αυτοί δεν χρειάζονται κώδικα ή προγραμματισμό για να προβάλουν και να αναλύσουν δεδομένα.

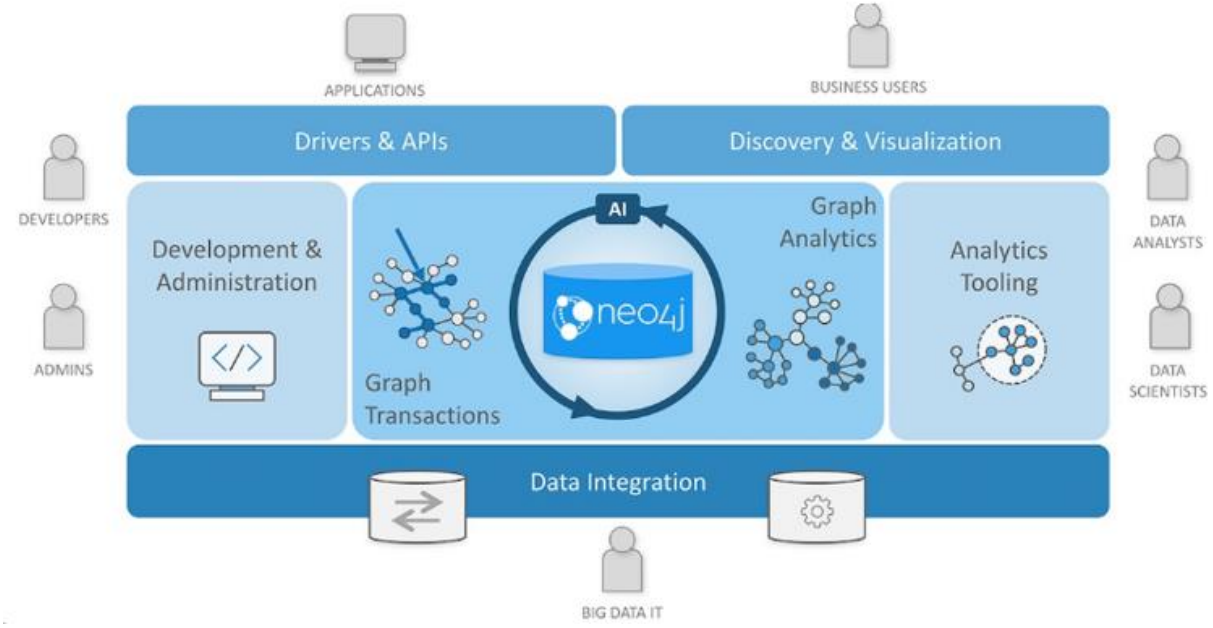
- **Neo4j ETL Tool:** Λαμβάνει τα δεδομένα από μια σχεσιακή βάση δεδομένων ή από μια μορφή πίνακα και εισαγωγής στο Neo4j

- Εκτός από τα παραπάνω, το Neo4j αποτελείται από **μια ποικιλία βιβλιοθηκών** επεκτάσεων και εργαλείων προγραμματιστών τα οποία έχουν την δυνατότητα να προστίθενται σε ήδη υπάρχοντα προϊόντα με σκοπό την καλυτέρευση της λειτουργικότητάς τους.

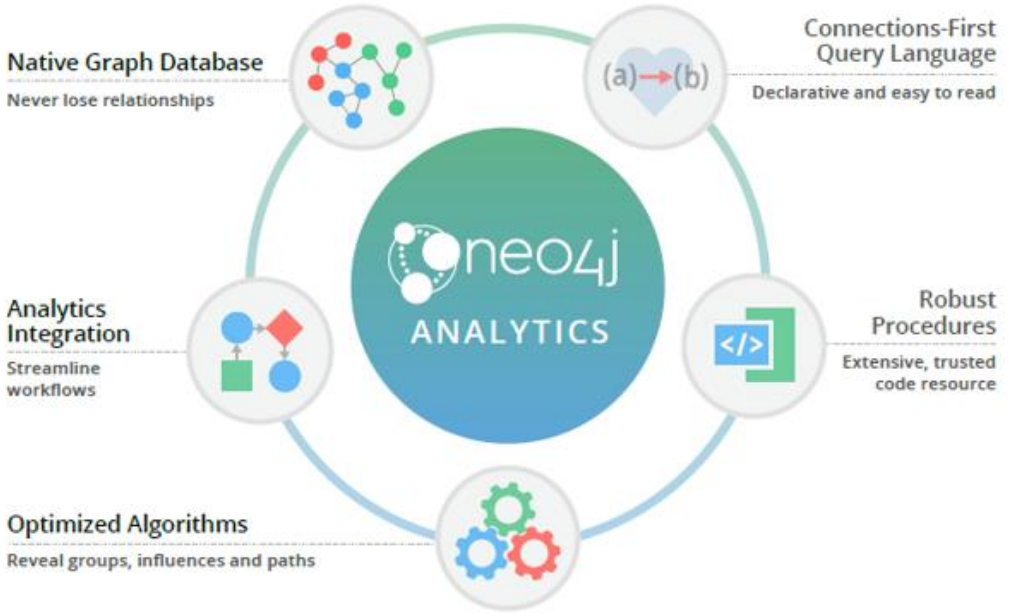
- **Από Αλγόριθμους γραφικών Neo4j:**
    - **Εύρεση διαδρομής (Path finding):** Είναι αλγόριθμοι οι οποίοι βοηθούν στο να βρεθεί η πιο σύντομη διαδρομή, στο να αξιολογηθεί η διαθεσιμότητα των διαδρομών και στην ποιότητά τους.
    - **Κεντρικότητα (Centrality):** Αλγόριθμοι οι οποίοι καθορίζουν τη σημασία των διακριτών κόμβων σε ένα δίκτυο.

- **Ανίχνευση κοινότητας (community detection):** Είναι αλγόριθμοι οι οποίοι κρίνουν τον τρόπο που ομαδοποιείται και διαμοιράζεται μια ομάδα, καθώς και το πώς ενισχύεται ή διασπάται.
- **Ομοιότητα (Similarity):** Είναι αλγόριθμοι οι οποίοι συνεισφέρουν στον υπολογισμό της ομοιότητας των κόμβων.

- **Neo4j Graph Platform**



- **Advanced Analytics in Neo4j**



2.8.2 ΜΟΝΤΕΛΟΠΟΊΗΣΗ ΔΕΔΟΜΈΝΩΝ - ΜΟΝΤΈΛΟ ΓΡΆΦΩΝ ΜΕ ΕΤΙΚΈΤΕΣ ΚΑΙ ΙΔΙΌΤΗΤΕΣ ΚΑΙ ΤΕΧΝΙΚΉ ΜΟΝΤΕΛΟΠΟΊΗΣΗΣ

Η βάση δεδομένων Neo4j Graph χρησιμοποιεί το **Μοντέλο γράφων με ετικέτες και ιδιότητες** (labeled property graph model) για την αποθήκευση και τη διαχείριση των δεδομένων. Τα βασικά χαρακτηριστικά του μοντέλου γράφων με ετικέτες και ιδιότητες περιλαμβάνουν κόμβους, Σχέσεις, Ετικέτες και Ιδιότητες.

- Κόμβοι ενώνονται με σχέσεις, αναπαριστώνται με κύκλους και περιέχουν τις ιδιότητες, ενώ τα δεδομένα αποθηκεύονται μέσω ιδιοτήτων στους κόμβους.
- Οι Σχέσεις περιέχουν ιδιότητες ενώνουν τους κόμβους με τις ακμές και μπορεί να είναι και μονής και διπλής κατεύθυνσης. Κάθε σχέση περιέχει έναν "Κόμβο έναρξης» και έναν «Κόμβο τερματισμού», Τύπο και Ιδιότητες.
- Για να γίνει ομαδοποίηση, οι ετικέτες δηλώνονται σε κόμβους.
- Οι ιδιότητες είναι ζεύγη κλειδιού-τιμής και τα κλειδιά είναι αλφαριθμητικά και οι τιμές πρωτογενής τύποι δεδομένων.

Συχνά, όταν σχεδιάζουν ένα μοντέλο δεδομένων, οι άνθρωποι σχεδιάζουν παραδείγματα δεδομένων σε έναν πίνακα και τα συνδέουν με άλλα δεδομένα που έχουν σχεδιαστεί με σκοπό να δείχνουν πώς συνδέονται διαφορετικά στοιχεία. Το μοντέλο του λευκού πίνακα (whiteboard model) αλλάζει και αναδομείται ώστε να ταιριάζει σε κανονικοποιημένους πίνακες για ένα σχεσιακό μοντέλο. Στη μοντελοποίηση δεδομένων γραφήματος, τα δεδομένα μοντελοποιούνται σε έναν πίνακα. Αντί να αλλάξει το μοντέλο για να ταιριάζει σε έναν κανονικοποιημένο πίνακα, το μοντέλο δεδομένων γραφήματος παραμένει το ίδιο και τα δεδομένα χειρίζονται για να χωρέσουν. Για αυτό το λόγο, το μοντέλο παίρνει το όνομα του ως «φιλικό προς τον πίνακα».

Κατά συνέπεια, με την εξής λογική, τα βασικά βήματα μοντελοποίησης γραφημάτων μπορούν να οργανωθούν ως εξής:

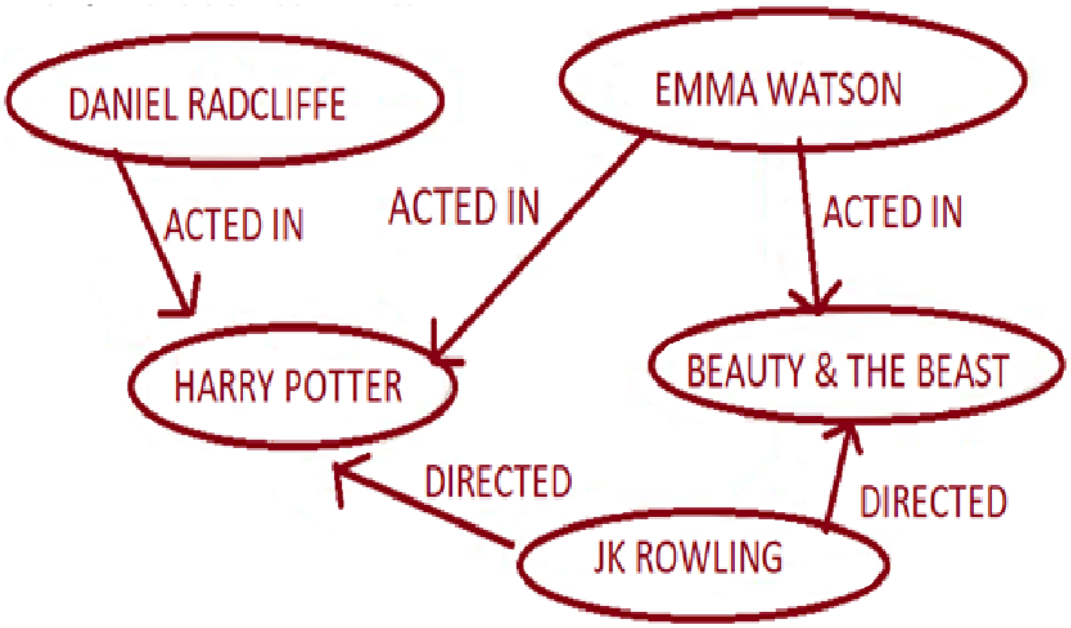
- Εντοπίζουμε και καταλαβαίνουμε καλά το πρόβλημά μας.
- Γράφουμε στον πίνακα στις απαιτήσεις του προβλήματος μας, δίνοντας πολύ μεγάλη σημασία στα εξής:
    - Στα διασυνδεμένα δεδομένα.
    - Στις σχέσεις μεταξύ τους.
    - Στα ερωτήματα που πρέπει να απαντά η εφαρμογή μας.
- Μπορούμε να βελτιώσουμε το σχήμα μας με πεπερασμένα περάσματα – βήματα έως ότου να αποφασίσουμε ότι αναπαριστά το καλύτερο σενάριο της πραγματικής εφαρμογής. Για να γίνει αυτό, μπορούμε να χρησιμοποιήσουμε τα παρακάτω βήματα σχεδιασμού:
    - Βάζουμε στον πίνακα τους κόμβους,
    - Βάζουμε τις ετικέτες στους κόμβους.
    - Ενώνουμε τους κόμβους με τις σχέσεις.
    - Βάζουμε όλα τα απαραίτητα δεδομένα στον γράφο εισάγοντας ιδιότητες στους κόμβους και στις σχέσεις.

2.8.3 ΠΑΡΑΔΕΊΓΜΑΤΑ

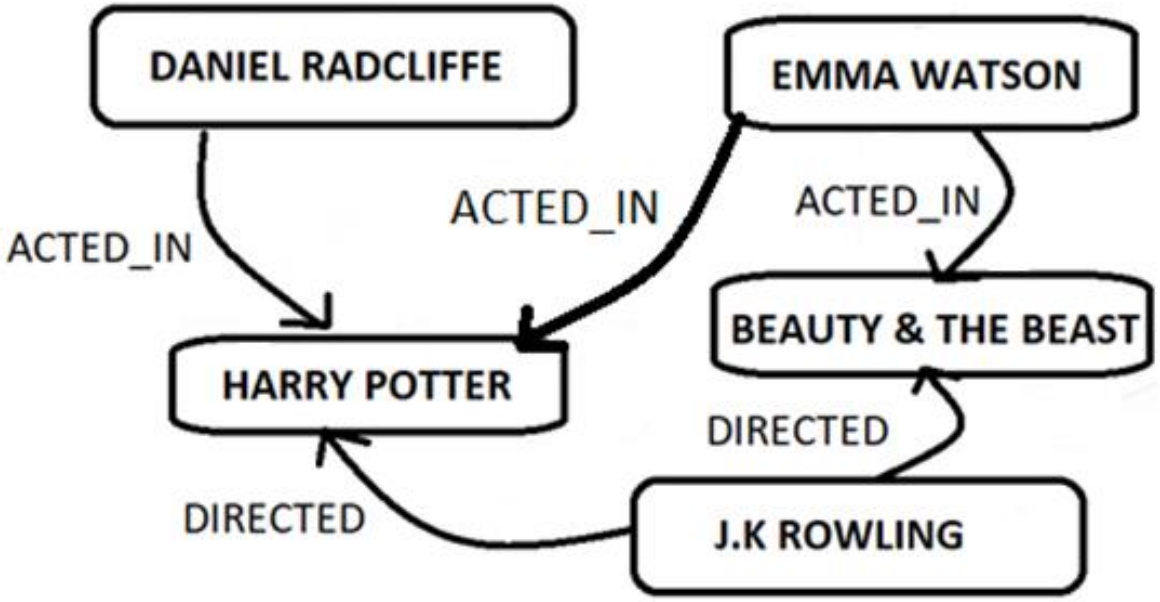
ΠΑΡΑΔΕΙΓΜΑ 1ο

Μοντελοποιήστε μια εφαρμογή με τους χαρακτήρες του HARRY POTTER και τις ταινίες που έπαιξαν χωρίς να ακολουθήσετε κάποια τεχνική στα βήματα μοντελοποίησης.

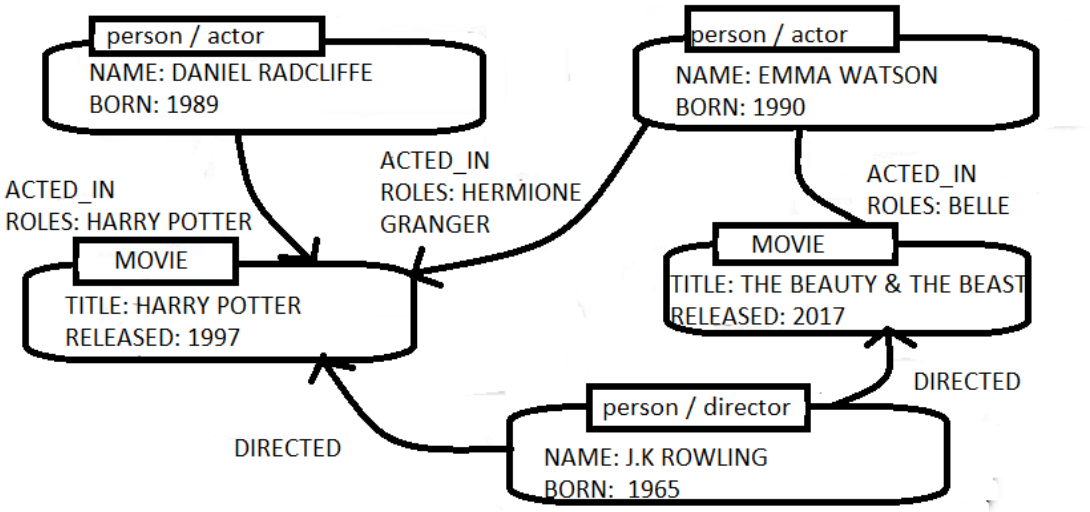
Βήμα 1ο: Αναγράφουμε στον πίνακα τα δεδομένα και τις πληροφορίες μας σε μορφή κόμβων και σχέσεων:



Βήμα 2ο : Ύστερα καθαρογράφουμε το σχήμα ώστε να του δώσουμε μία σαφή μορφή που να αντιστοιχεί σε γράφο ορίζοντας με σαφήνεια και ακρίβεια τους κόμβους και τις σχέσεις (ονοματολογία, κοινή όπου χρειάζεται στις σχέσεις…) :



Βήμα 3ο : Για να ομαδοποιήσουμε τους κόμβους τους βάζουμε ετικέτες και καθορίζουμε τις ιδιότητες σε όλους τους κόμβους και στις σχέσεις ώστε να μπορούμε να εισάγουμε όλα τα αναγκαία δεδομένα στον γράφο. Έτσι ο γράφος παίρνει την τελική του μορφή ως **γράφος με ετικέτες και ιδιότητες** (labeled property graph).



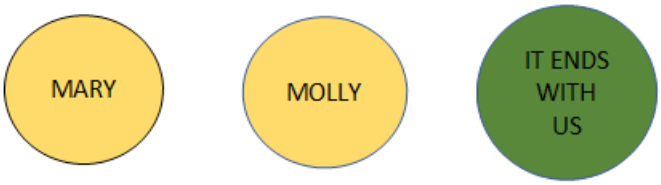
Τέλος, δημιουργούμε και προβάλουμε το μοντέλο στη βάση δεδομένων γράφων Neo4j και σιγουρευόμαστε ότι ταιριάζει με αυτό που σχεδιάσαμε στον αρχικό πίνακα.

- Η απλοποίηση του μοντέλου δεδομένων σας στον πίνακα καθιστά το μοντέλο γραφήματος απίστευτα οπτικό και εύκολο στην κατανόηση.
- Δεν χρειάζεται να γράψουμε διαφορετικές εκδόσεις του επιχειρηματικού μοντέλου ή να το γράψουμε καθόλου.
- Το μοντέλο δεδομένων γραφήματος είναι εύκολο να κατανοήσει ο καθένας, ενώ οι όροι σε ένα ERD θα πρέπει να εξηγηθούν στους χρήστες.

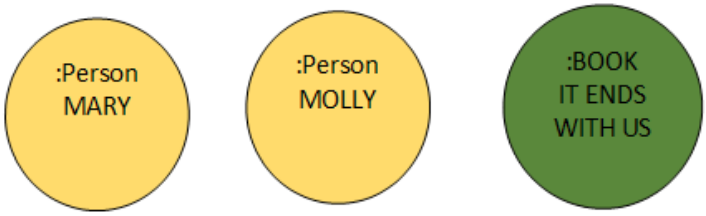
ΠΑΡΑΔΕΙΓΜΑ 2ο

Στο παρακάτω παράδειγμα σε αντίθεση με το παραπάνω παράδειγμα, θα ακολουθήσουμε κάποια τεχνική στα βήματα μοντελοποίησης. Έστω, δύο άνθρωποι, η Mary και η Molly, είναι φίλοι. Και οι δύο έχουν διαβάσει το βιβλίο IT ENDS WITH US. Πρώτα θα μπουν οι κόμβοι, μετά οι σχέσεις, και στη συνέχεια οι ετικέτες και τέλος οι ιδιότητες.

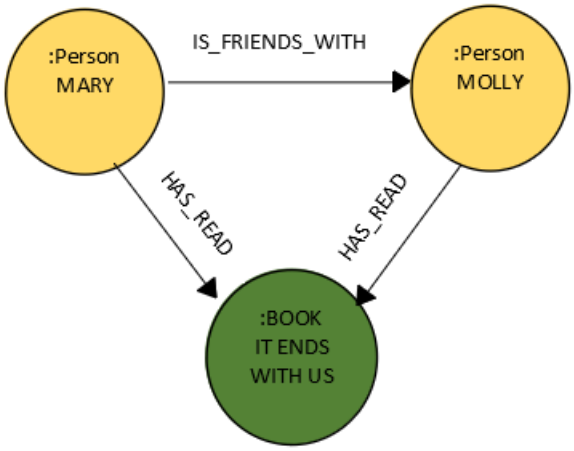
1Ο ΒΗΜΑ: Πρώτα φτιάχνουμε τους κόμβους



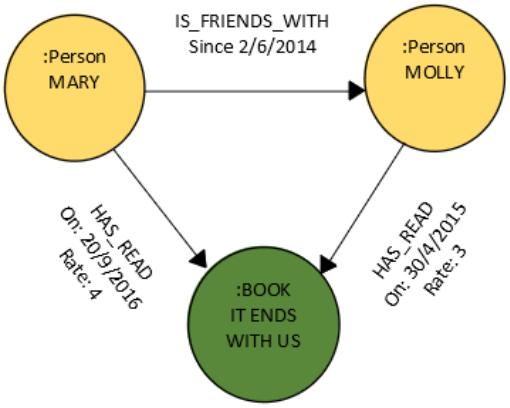
2Ο ΒΗΜΑ: Προσθέτουμε τα labels



3Ο ΒΗΜΑ: Μετά προσθέτουμε τις σχέσεις



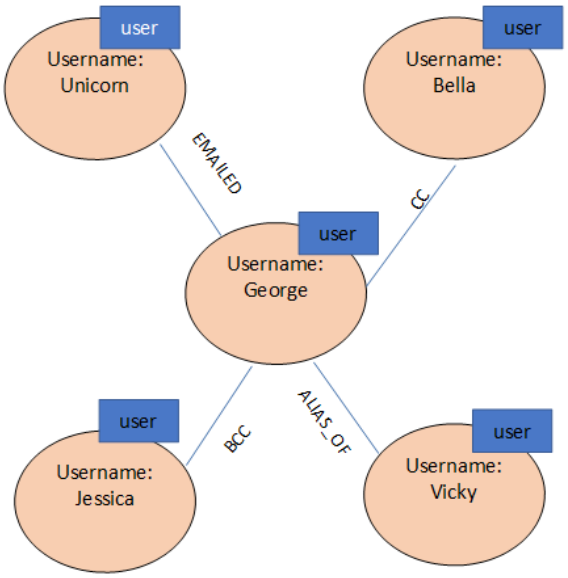
4ο ΒΗΜΑ: Τέλος προσθέτουμε τις ιδιότητες.



ΠΑΡΑΔΕΙΓΜΑ 3ο

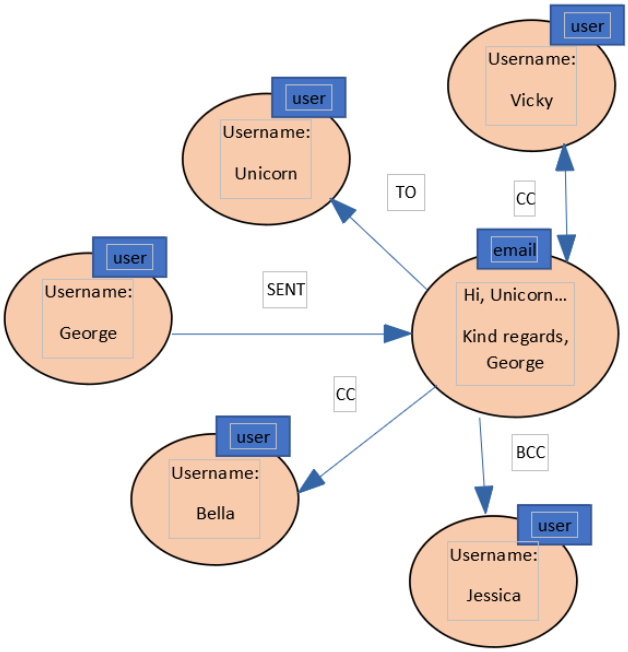
Σε αυτό το παράδειγμα, θα εξετάσουμε μια εφαρμογή ανίχνευσης απάτης που αναλύει τα μηνύματα στην πλατφόρμα του Instagram των χρηστών.

1ο ΒΗΜΑ: Αρχικά, στο μοντέλο δεδομένων θα εντοπίσουμε και θα καταγράψουμε ορισμένους χρήστες, τις δραστηριότητές τους και τα γνωστά τους ψευδώνυμα, συμπεριλαμβανομένης μιας σχέσης που περιγράφει την Βίκυ ως ένα από τα γνωστά ψευδώνυμά του Τζόρτζ. Το αποτέλεσμα είναι ένα γράφημα σε σχήμα αστεριού με τον Τζόρτζ στο κέντρο.



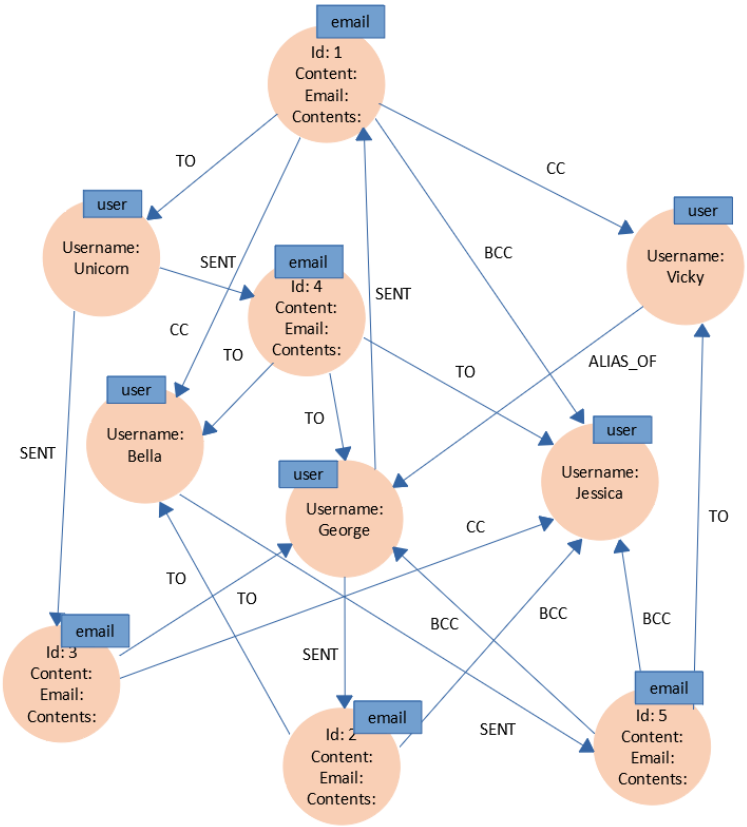
Με μια πρώτη ματιά το παραπάνω σχήμα μοντελοποίησης των δεδομένων μοιάζει με την σαφή απεικόνιση της επικοινωνίας του George με τους άλλους μέσω των direct messages, ωστόσο δεν είναι μια μακροχρόνια λύση. Ακόμη και με ιδιότητες που συνδέονται με κάθε σχέση EMAILED, δεν θα μπορούσαμε να του δώσουμε την δυνατότητα να κάνει ανταλλαγή των σχέσεων EMAILED, CC, και BCC και για αυτό το λόγο, χρειαζόμαστε τις σχέσεις συσχέτισης, ώστε να δώσουμε τη λύση στην ανίχνευση απάτης. Ωστόσο, δεν μπορούμε να δούμε το πιο σημαντικό μέρος όλων: το ίδιο το direct message.

2Ο ΒΉΜΑ: Έτσι ώστε να φτιάξουμε το αδύναμο μοντέλο μας, πρέπει να προσθέσουμε κόμβους στο μοντέλο γραφήματος που εκπροσωπούν καθένα από τα direct messages που ανταλλάσσονται. Ύστερα, πρέπει να προσθέσουμε τις νέες σχέσεις για να παρακολουθούμε ποιος έγραψε το email και σε ποιον στάλθηκε, έτσι βγαίνει το CC' και BCC' κατά συνέπεια.



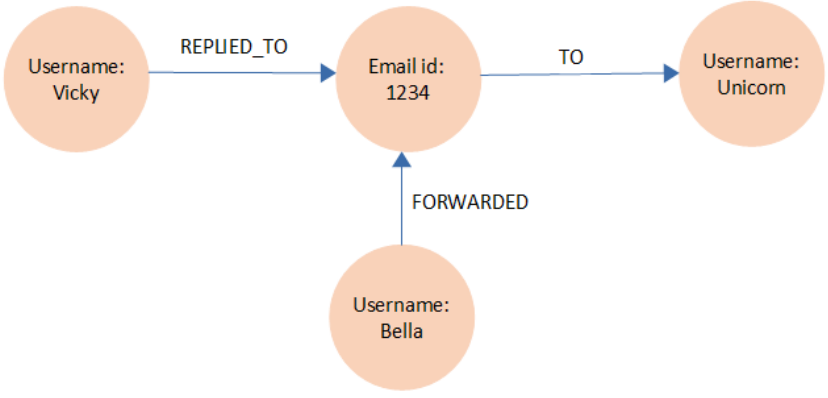
Το αποτέλεσμα είναι ένα άλλο γράφημα σε σχήμα αστεριού, αλλά αυτή τη φορά το email βρίσκεται στο κέντρο, επιτρέποντάς μας να παρακολουθούμε επιτυχώς τη σχέση του με τον Μπομπ και από εκεί πιθανώς κάποια ύποπτη συμπεριφορά.

3ο ΒΉΜΑ: Ωστόσο να επιβλέπουμε παραπάνω από ένα direct messages όπου το καθένα με το δικό του δίκτυο αλληλεπιδράσεων προς εξερεύνηση.



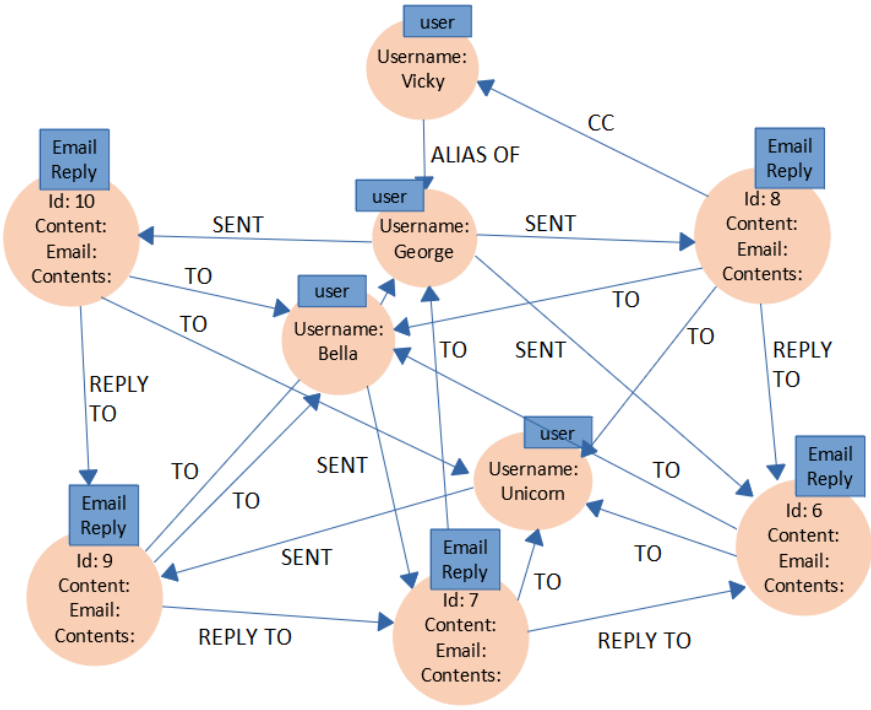
4ο ΒΉΜΑ: Το παραπάνω σχήμα αν και είναι πιο εύκολο στην κατανόηση με τους κατάλληλους κόμβους και τις συνδέσεις τους, ωστόσο δεν είναι έτοιμο. Αν και μπορούμε να παρατηρήσουμε ποιος έστειλε email σε ποιον, και ποιος έλαβε email από ποιόν καθώς και το τι πληροφορίες περιέχουν αυτά τα mail, δεν μπορούμε να δούμε τις απαντήσεις τους, η τις προωθήσεις των δεδομένων επικοινωνιών μας μέσω των email.

Είναι σημαντικό να ελέγχουμε για διαρροές πληροφοριών λόγω απάτης ή απόπειρες εισβολής. Για αυτόν τον λόγο πρέπει να γνωρίζουμε εάν έχουν παραβιαστεί σημαντικές επιχειρηματικές πληροφορίες. Αν είχαμε σχέσεις FORWARDED και REPLIED_TO στο παραπάνω γράφημα τότε ούτε αυτό θα ήταν απολύτως σωστό καθώς όπως και το email οι ιδιότητες στη σχέση είναι ανίκανες να μας παρέχουν πληροφορίες για τις γειτονικές τους σχέσεις.



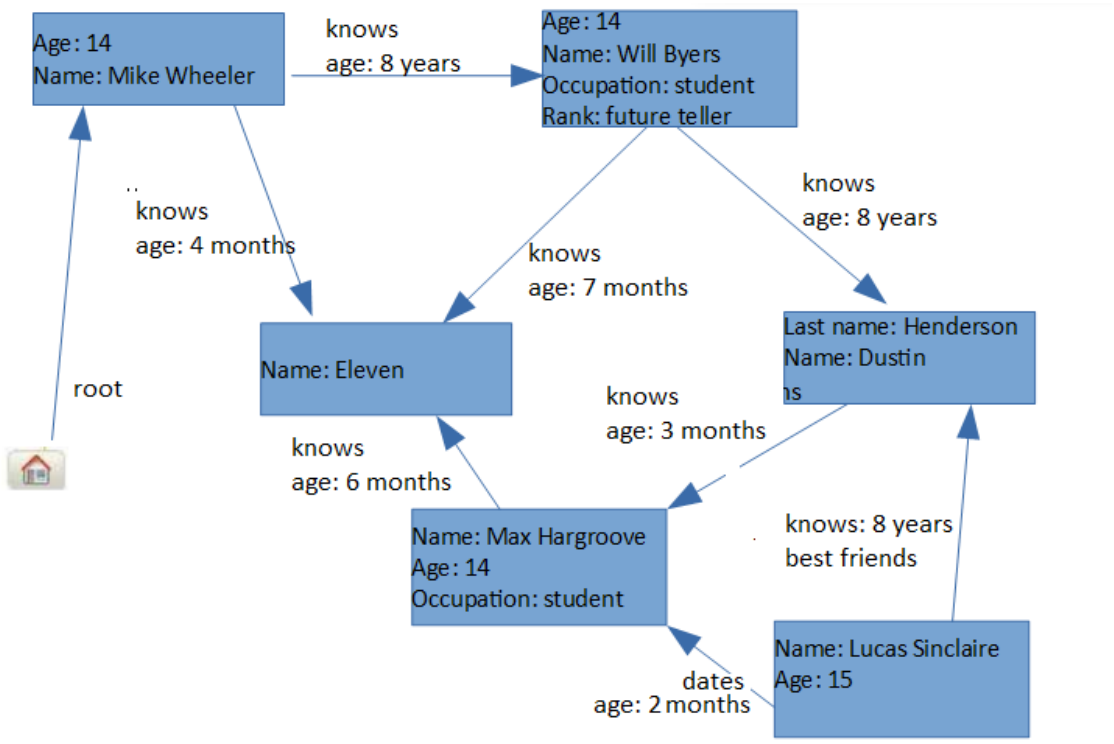
5ο ΒΗΜΑ: Έτσι, τα στοιχεία επικοινωνίας με απάντηση και προώθηση είναι σημαντικός παράγοντας στο να έχουμε ένα καλύτερο μοντέλο δεδομένων.

Η απάντηση σε ένα email είναι ένα νέο email που επισυνάπτεται στο πρωτότυπο. Αυτό μπορεί να αναπαρασταθεί χρησιμοποιώντας δύο ετικέτες: Email και Reply. Αντίστοιχα με τις σχέσεις TO, CC και BCC, μια απάντηση μπορεί να σταλεί στον αποστολέα, σε όλους τους παραλήπτες ή ακόμα και στον αρχικό παραλήπτη καθώς και σε ένα μικρό υποσύνολο της λίστας παραληπτών. Έτσι μπορούμε να συνδέσουμε το αρχικό email με σχέση REPLY_TO.



ΠΑΡΑΔΕΙΓΜΑ 4ο

Έστω ένας γράφος, με ιδιότητες και ετικέτες από την σειρά STRANGER THINGS.



Στο παραπάνω παράδειγμα βλέπουμε ότι κάθε άτομο εκπροσωπείται από έναν κόμβο ο οποίος έχει μέσα τις πληροφορίες τους. Ορισμένοι κόμβοι περιέχουν τόσο προσωπικές όσο και μη προσωπικές πληροφορίες, ενώ άλλοι κόμβοι περιέχουν μόνο προσωπικές πληροφορίες. Κάθε κόμβος περιέχει δύο διαφορετικά μοντέλα για την αποθήκευση δεδομένων: ένα για μη προσωπικά δεδομένα και ένα για προσωπικά δεδομένα.

Η Neo4j χρησιμοποιεί ως Entry Node σε κάθε γράφημα έναν κόμβο αναφοράς με id=0, ο οποίος είναι ο αρχικός κόμβος πατέρας του γράφου. Με αυτή την λογική παρακάτω θα δημιουργήσουμε τα τμήματα κώδικα σε Java με βάση τα οποία θα υλοποιηθούν οι παραπάνω κόμβοι.

```
MikeService mike = ... // Get factory
// Create Mike Wheeler
Node mrWheeler = mike.createNode();
mrWheeler.setProperty( "name", "Mike Wheeler" );
mrWheeler.setProperty( "age", 14);

// Create Morpheus Node will = mike.createNode();
will.setProperty( "name", "Will" );
will.setProperty( "rank", "future teller" );
will.setProperty( "occupation", "student" );

// Create a relationship representing that they know each other
mrWheeler.createRelationshipTo( mike, RelTypes.KNOWS );

// ...create Eleven, Max Hargroove, Lucas Sinclaire, Dustin similarly
```

ΚΕΦΑΛΑΙΟ 3

3.1 NEO 4J - CYPHER

3.1.1 ΟΡΙΣΜΌΣ CYPHER

Η Cypher είναι μια δηλωτική και κύρια γλώσσα ερωτημάτων που χρησιμοποιείται από τη βάση δεδομένων γραφημάτων Neo4j. Επιτρέπει στους χρήστες να αποθηκεύουν και να ανακτούν δεδομένα από τη βάση δεδομένων γραφημάτων.

Ενώ μπορούμε να υποβάλουμε ερωτήματα, τα δεδομένα που είναι αποθηκευμένα σε ένα γράφημα ιδιοτήτων μπορούν να υποβληθούν σε αναζήτηση και διαχείριση μέσω ενός property graph με χρήση του Cypher. Αυτό περιλαμβάνει την τροποποίηση της δομής ενός γραφήματος, η οποία μπορεί να γίνει μέσω ενός γράφου. Η Cypher επίσης παρέχει έναν property graph εισόδου, έναν property graph εξόδου ή, διαφορετικά, τα οποία είναι εργαλεία για να τα ερωτήματα της. Όσο καλά και αν είναι αυτά της τα εργαλεία, το μεγαλύτερο πλεονέκτημα της είναι το συντακτικό το οποίο είναι τόσο απλό στην κατανόηση που μας φαίνεται λες και εμείς οι ίδιοι φτιάχνουμε τις κόμβους και τα μονοπάτια που ψάχνουμε. Ορισμένες ιδιότητες της είναι οι εξής:

- Ακολουθεί τη σύνταξη της SQL.
- Η σύνταξη της είναι απλής και η μορφή της πολύ αναγνώσιμη στον άνθρωπο.
- Η σύνταξη της Cypher παρέχει έναν οπτικό και λογικό τρόπο αντιστοίχισης μοτίβων κόμβων και σχέσεων στο γράφημα.
- Χρησιμοποιεί σύνταξη ASCII-Art.
- Είναι μια ανοιχτή προδιαγραφής γλώσσα (open source)

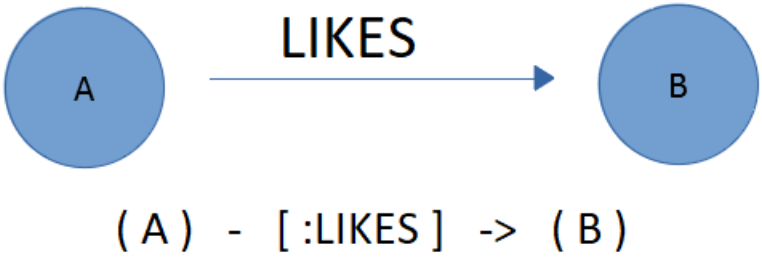
Η Cypher ακολουθεί επίσης σειριακή δομή, πράγμα το οποίο σημαίνει ότι οι συναρτήσεις που μπορεί να υπάρχουν μέσα σε ένα ερώτημα που θέτουμε εκτελούνται με την σειρά, με κατεύθυνση από τα αριστερά προς τα δεξιά. Για παράδειγμα, η περιγραφή των δεμένων θα μπορούσε να γίνει στην αρχή με την χρήση της εντολής SELECT στην SQL – στην Cypher, αυτή η περιγραφή θα γίνει στο τέλος με την χρήση της εντολής RETURN. Ως αποτέλεσμα, έχουμε την ένωση πολλών ερωτημάτων με τον όρο WITH, το οποίο επιτρέπει τη σύνδεσή τους με γραμμικότητα. Αυτό σημαίνει ότι η έξοδος ενός ερωτήματος χρησιμοποιείται ως είσοδος για ένα άλλο.

3.1.2 CYPHER – ΒΑΣΙΚΉ ΣΥΝΤΑΞΗ

Η ιδέα πάνω στην οποία χτίστηκε η Cypher είναι η αναζήτηση των μοτίβων. Ένα μοτίβο θα μπορούσε στην Cypher να αναπαρασταθεί ως εξής: (α) - [β] → (γ)

Στο παραπάνω υπάρχουν 3 πράγματα τα οποία πρέπει να θυμόμαστε:

1. Τα Α και Γ μέσα στις παρενθέσεις είναι κόμβοι
2. Το Β είναι η σχέση η οποία ενώνει το Α με το Γ
3. Το βέλος ανάμεσα στο Β και Γ αν και προαιρετικό υποδηλώνει την κατεύθυνση της σχέσης.



ΑΝΑΠΑΡΆΣΤΑΣΗ ΚΌΜΒΩΝ ΣΤΗΝ CYPHER

Όπως είδαμε στο παραπάνω παράδειγμα, στην Cypher, αν θέλουμε να χρησιμοποιήσουμε κόμβους, το κάνουμε με την χρήση παρενθέσεων. Οι παρενθέσεις μπορούν να χρησιμοποιηθούν με 2 τρόπους: μπορούμε είτε να τις αφήσουμε κενές (π.χ. ( ))και με αυτόν τον τρόπο να μπορούν να ταιριάξουν με οποιονδήποτε κόμβο έχουμε στην βάση δεδομένων μας, είτε μπορούμε μέσα στην παρένθεση να βάλουμε μια μεταβλητή (π.χ. (node)) και έτσι να αντιστοιχεί σε κάποιον συγκεκριμένο κόμβο.

Οι ετικέτες εμπεριέχονται στους κόμβους του Neo4j στην γλώσσα προγραμματισμού Cypher οι ετικέτες τοποθετούνται μετά την χρήση άνω και κάτω τελείας (:). Σαν τους κόμβους, μπορούμε να χρησιμοποιήσουμε μια ετικέτα με 2 τρόπους: στην παρένθεση μπορούμε να ακολουθήσουμε την άνω κάτω τελεία με το όνομα της ετικέτας, (π.χ. (:Label)) είτε μπορούμε να εμπεριέχουμε το όνομα του κόμβου, την άνω κάτω τελεία ακολουθούμενου από το όνομα της ετικέτας (π.χ. (node:Label)).

Ένας κόμβος εκτός από ετικέτες μπορεί επίσης να έχει μια, ή περισσότερες από μια ιδιότητες, οι οποίες στην γλώσσα Cypher, τις δηλώνουμε μέσα σε άγκιστρα. (πχ. (u: User { name: “David Joe”}).

ΑΝΑΠΑΡΆΣΤΑΣΗ ΣΧΈΣΕΩΝ ΣΤΗΝ CYPHER

Για να δηλώσουμε μια σχέση, στην Cypher, χρησιμοποιούμε δυο παύλες (- -) μαζί με τα βέλη μικρότερο ή μεγαλύτερο αντίστοιχα (> ή <). Το ποιο από τα δύο θα χρησιμοποιήσουμε εξαρτάται από την κατεύθυνση που θέλουμε να έχει η σχέση. Είναι σημαντικό να ξέρουμε την διαφορά στην χρήση των παυλών και των βελών αντίστοιχα. Για παράδειγμα όταν έχουμε (a)--(b) αυτό μας δείχνει ότι μεταξύ των κόμβων a και b υπάρχει μια σχέση στην οποία δεν μας ενδιαφέρει η κατεύθυνση ενώ αντίστοιχα αν έχουμε (a)-→(b) μας δείχνει ότι υπάρχει μια σχέση η οποία έχει φορά κατεύθυνση από τον κόμβο a στον κόμβο b.

Στις σχέσεις, όπως και στους κόμβους, μπορεί να δοθεί ένα όνομα μεταβλητής. και να ορίσουμε τις μέσα σε μια σχέση, χρησιμοποιούμε αγκύλες. Για παράδειγμα (a)-[r]→ (b), στην σχέση που ενώνει τους a και b έχουμε δώσει την μεταβλητή r.

Τα ονόματα σχέσεων, όπως οι ετικέτες κόμβων, ακολουθούνται πάντα από άνω και κάτω τελεία ":", ενώ τα χαρακτηριστικά περικλείονται σε αγκύλες, π.χ. :(a)–[l:LIKES {val:5}]->(b)

3.1.3 ΧΡΉΣΙΜΕΣ ΚΑΙ ΒΑΣΙΚΈΣ ΕΚΦΡΆΣΕΙΣ

Η Cypher έχει πάρα πολλές εκφράσεις που μπορούμε να χρησιμοποιήσουμε όταν γράφουμε κώδικα και υπάγονται στις εξής κατηγορίες:

1. Εγγραφής (Write clauses of Neo4j Cypher Query Language)
2. Ανάγνωσης (Read clauses of Neo4j Cypher Query Language)
3. Γενικές (General clauses of Neo4j Cypher Query Language)

Γενικές Εκφράσεις

	ΈΚΦΡΑΣΗ	ΧΡΉΣΗ
1	**CREATE**	Η έκφραση Create για να δημιουργήσουμε κόμβους, σχέσεις και ιδιότητες
2	**MERGE**	Η έκφραση Merge στην περίπτωση που το μοτίβο που θέλουμε υπάρχει ήδη στον γράφο. Αν όχι, το δημιουργεί.
3	**SET**	Η έκφραση Set χρησιμοποιείται για να ανανεώσει τις ετικέτες στους κόμβους, τις ιδιότητες στους κόμβους και τις σχέσεις.
4	**DELETE**	Η έκφραση Delete χρησιμοποιείται για να διαγράψει κόμβους, σχέσεις ή και μονοπάτια από τον γράφο.
5	**REMOVE**	Η έκφραση Remove χρησιμοποιείται για να αφαιρέσει ιδιότητες και στοιχεία από τους κόμβους και σχέσεις.
6	**FOREACH**	Η έκφραση Foreach χρησιμοποιείται για να ανανεώσουμε τα δεδομένα μέσα σε μια δοσμένη λίστα.
7	**CREATE UNIQUE**	Χρησιμοποιώντας τις σχέσεις CREATE και MATCH, μπορούμε να πάρουμε ένα μοναδικό μοτίβο, ταιριάζοντας ένα ήδη υπάρχοντα μοτίβο και δημιουργώντας ένα που δεν υπάρχει.
8	**IMPORTING CSV FILES WITH CYPHER**	Με την LOAD CSV μπορούμε να κάνουμε import δεδομένα από αρχεία τύπου .CSV

Χρησιμοποιούμε την εντολή CREATE όταν. . .

- θέλουμε να δημιουργήσουμε έναν μοναδικό κόμβο.

  CREATE (node_name);

- θέλουμε να δημιουργήσουμε έναν μοναδικό κόμβο.

  CREATE (node1),(node2)

- θέλουμε να δημιουργήσουμε έναν κόμβο με ετικέτα

  CREATE (node:label)

- θέλουμε να δημιουργήσουμε έναν κόμβο με πολλές ετικέτες

  CREATE (node:label1:label2:. . .labeln)

- θέλουμε να δημιουργήσουμε έναν κόμβο με ιδιότητες

  CREATE (node:label { key1:value, key2:value, ... })

- θέλουμε να επιστρέψουμε έναν κόμβο που δημιουργήσαμε

  CREATE (Node:Label{properties. . .}) RETURN Node

ΠΑΡΑΔΕΊΓΜΑΤΑ

1. Δημιούργησε σχέσεις

   CREATE (node1)-[:RelationshipType]->(node2)
   CREATE (Joy:player{name: "Joe Joe", YOB: 1905, POB: "New York"})
   CREATE (Amer:Country {name: "America"})
   CREATE (Joy)-[r:BATSMAN_OF]->(Amer)

2. Φτιάξε μια σχέση μεταξύ ήδη υπάρχουν κόμβων.

   MATCH (a:LabeofNode1), (b:LabeofNode2) WHERE a.name = "nameofnode1" AND b.name = " nameofnode2" CREATE (a)-[: Relation]->(b) RETURN a,b

   MATCH (a:player), (b:Country) WHERE a.name = "Joe Joe" AND b.name = "America" CREATE (a)-[r: BATSMAN_OF]->(b) RETURN a,b

3. Φτιάξε μια σχέση με ετικέτες και ιδιότητες

   CREATE (node1)-[label:Rel_Type {key1:value1, key2:value2, . . . n}]-> (node2)
   MATCH (a:player), (b:Country) WHERE a.name = "Joe Joe" AND b.name = "America" CREATE (a)-[r:BATSMAN_OF {Matches2, Avg:87.2}]->(b) RETURN a,b

Εκφράσεις Ανάγνωσης

	ΕΚΦΡΑΣΕΙΣ	ΧΡΗΣΕΙΣ
1	MATCH	Χρησιμοποιούμε το MATCH για να ψάξουμε στα δεδομένα με ένα συγκεκριμένο μοτίβο
2	OPTIONAL MATCH	Το Optional Match είναι ίδιο με το Match με την μόνη διαφορά οτι μπορεί να χρησιμοποιήσει τα NULLS σε περίπτωση που λείπουν μέρη από το μοτίβο.
3	WHERE	Χρησιμοποιείται για να προσθέσει περιεχόμενα στα CQL ερωτήματα
4	START	Χρησιμοποιείται για να βρούμε τα σημεία εκκίνησης μέσω των ευρετηρίων παλαιού τύπου
5	LOAD CSV	Χρησιμοποιείται για να κάνουμε import δεδομένα από αρχεία τύπου .CSV

Επιπλέον γενικές εκφράσεις οι οποίες θα μας είναι χρήσιμες στην Cypher είναι οι εξής:

	ΕΚΦΡΑΣΕΙΣ	ΧΡΗΣΕΙΣ
1	RETURN	Χρησιμοποιείται για να καθορίσουμε τι θα περιλάβουμε στο query result set.
2	ORDER BY	Χρησιμοποιείται για να διαμορφώσουμε το αποτέλεσμα μιας query σε σειρά. Συνήθως, χρησιμοποιείται με τις εκφράσεις RETURN ή WITH.
3	LIMIT	Χρησιμοποιείται για να περιορίσουμε τις σειρές στο αποτέλεσμα σε μια συγκεκριμένη τιμή.
4	SKIP	Χρησιμοποιείται για να καθορίσουμε από ποια σειρά να αρχίσουμε, συμπεριλαμβανομένου τις σειρές στο αποτέλεσμα.
5	WITH	Χρησιμοποιείται για να ενώσει τα μέρη της query.
6	UNWIND	Χρησιμοποιείται για να προεκτείνει μια λίστα σε μια ακολουθία σειρών.
7	UNION	Χρησιμοποιείται για να συνδυάσει το αποτέλεσμα πολλών query.
8	CALL	Χρησιμοποιείται για να επικαλεστεί μια διαδικασία που αναπτύσσεται στη βάση δεδομένων

3.1.4 ΣΥΝΑΡΤΗΣΕΙΣ ΣΤΗΝ CYPHER

Εκτός από εκφράσεις, η Cypher έχει και πολλές συναρτήσεις οι οποίες μας βοηθούν στην εγγραφή κώδικα:

	ΈΚΦΡΑΣΗ	ΧΡΗΣΕΙΣ
1	STRING	Την χρησιμοποιούμε όταν δουλεύουμε με String literals.
2	AGGREGATION	Την χρησιμοποιούμε για να κάνουμε αθροιστικές λειτουργίες στα αποτελέσματα της CQL Query.
3	RELATIONSHIP	Την χρησιμοποιούμε για να πάρουμε λεπτομέρειές από σχέσεις πχ τον startnode, endnode, etc.

Υπάρχουν ορισμένα πράγματα τα οποία πρέπει να ξέρουμε για τους τύπους δεδομένων στην Cypher:

1. Οι τύποι δεδομένων στην Cypher μοιάζουν πάρα πολύ με τους τύπους δεδομένων στην Java το οποίο κάνει την κατανόηση τους πιο εύκολη.
2. Τους τύπους δεδομένων τους χρησιμοποιούμε κυρίως όταν θέλουμε να ορίσουμε τις ιδιότητες ενός κόμβου ή μιας σχέσης.

3.1.5 ΤΥΠΟΙ ΔΕΔΟΜΈΝΩΝ

Οι τύποι δεδομένων που υπάρχουν στην Neo4j της Cypher είναι οι εξής:

	CQL DATA TYPE	ΧΡΗΣΗ
1	BOOLEAN	Χρησιμοποιείται στην αντιπροσώπευση των Booleans και μπορεί να πάρει τιμές η TRUE ή FALSE.
2	BYTE	Χρησιμοποιείται στην αντιπροσώπευση 8 bit ακέραιων.
3	SHORT	Χρησιμοποιείται στην αντιπροσώπευση 16 bit ακέραιων.
4	INT	Χρησιμοποιείται στην αντιπροσώπευση 32 bit ακέραιων.
5	LONG	Χρησιμοποιείται στην αντιπροσώπευση 64 bit ακέραιων.
6	FLOAT	Χρησιμοποιείται στην αντιπροσώπευση 32 bit αριθμών με υποδιαστολή.
7	DOUBLE	Χρησιμοποιείται στην αντιπροσώπευση 64 bit αριθμών με υποδιαστολή.
8	CHAR	Χρησιμοποιείται στην αντιπροσώπευση 16 bit χαρακτήρων
9	STRING	Χρησιμοποιείται στην αντιπροσώπευση strings.

3.1.6 ΤΕΛΕΣΤΕΣ - OPERATORS

Ένας τελεστής, στον προγραμματισμό, είναι ένα σύμβολο που συνήθως αντιπροσωπεύει μια ενέργεια ή μια διαδικασία. Ένας τελεστής είναι ικανός να χειριστεί μια συγκεκριμένη τιμή ή τελεστή. Όπως και σε όλες τις γλώσσες, έτσι και η Neo4j της Cypher έχει δικούς της τελεστές. Παρακάτω ακολουθούν όλοι οι τελεστές που χρησιμοποιούνται στην Neo4j:

	ΕΊΔΟΣ	ΕΡΜΗΝΕΊΑ
1	MATHEMATICAL	+, -, *, %, ^
2	COMPARISON	+ , <> , < , > , <= , >=
3	BOOLEAN	AND , OR , XOR , NOT
4	STRING	+
5	LIST	+, IN , [X] , [X . . .Y]
6	REGULAR EXPRESSION	, =~,
7	STRING MATCHING	STARTS WITH, ENDS WITH, CONSTRAINTS

Τα Booleans στην Neo4j είναι τα εξής:

	BOOLEAN ΤΕΛΕΣΤΕΣ	ΕΡΜΗΝΕΊΑ
1	AND	Είναι λέξη κλειδί στην Neo4j CQL το οποίο υποστηρίζει όλες τις πράξεις AND. Είναι το αντίστοιχο του τελεστή SQL AND.
2	OR	Είναι λέξη κλειδί στην Neo4j CQL το οποίο υποστηρίζει όλες τις πράξεις OR. Είναι το αντίστοιχο του τελεστή SQL AND.
3	NOT	Είναι λέξη κλειδί στην Neo4j CQL το οποίο υποστηρίζει όλες τις πράξεις NOT. Είναι το αντίστοιχο του τελεστή SQL AND.
4	XOR	Είναι λέξη κλειδί στην Neo4j CQL το οποίο υποστηρίζει όλες τις πράξεις XOR. Είναι το αντίστοιχο του τελεστή SQL AND.

Εκτός από τα παραπάνω, στην Neo4j υπάρχουν και λέξεις κλειδιά που χρησιμοποιούνται στις συγκρίσεις και είναι:

	BOOLEAN ΤΕΛΕΣΤΕΣ	**ΕΡΜΗΝΕΊΑ**
1	=	Στην Neo4j CQL είναι το αντίστοιχο του "Ίσου με" τελεστή.
2	<>	Στην Neo4j CQL είναι το αντίστοιχο του "όχι ίσου με" τελεστή.
3	<	Στην Neo4j CQL είναι το αντίστοιχο του "λιγότερο από" τελεστή.
4	>	Στην Neo4j CQL είναι το αντίστοιχο του "μεγαλύτερο από" τελεστή.
5	<=	Στην Neo4j CQL είναι το αντίστοιχο του "λιγότερο ή ίσο από" τελεστή.
6	>=	Στην Neo4j CQL είναι το αντίστοιχο του "μεγαλύτερο η ίσο από" τελεστή.

3.1.7 NEO4J – ΑΛΓΌΡΙΘΜΟΙ ΓΡΆΦΩΝ

ΕΊΔΟΣ ΑΛΓΟΡΊΘΜΟΥ	ΠΡΌΒΛΗΜΑ ΓΡΆΦΩΝ	ΠΑΡΑΔΕΊΓΜΑΤΑ
 PATHFINDING & SEARCH	Το πρόβλημα εδώ είναι οτι δεν μπορούμε να βρούμε το καλύτερο μονοπάτι ή να εκτιμήσουμε την διαθεσιμότητα και ποιότητα διαδρομής.	• Βρείτε την πιο γρήγορη διαδρομή μεταξύ κόμβου Α και Β • δρομολόγηση τηλεφωνικών κλήσεων
 CENTRALITY	Το πρόβλημα εδώ είναι ότι δεν μπορούμε να καθορίσουμε τη σημασία των διακριτών κόμβων στα δίκτυα	• καθορίστε τις επιρροές των μέσων κοινωνικής δικτύωσης • βρείτε πιθανούς στόχους επίθεσης στα δίκτυα επικοινωνίας και μεταφοράς
 COMMUNITY DETECTION	Το πρόβλημα εδώ είναι ότι δεν μπορούμε να αξιολογήσουμε πώς ομαδοποιείται ή κατανείμετε μια ομάδα	• τμήμα των πελατών • βρείτε πιθανά μέλη ενός κυκλώματος απάτης

3.2 ΧΡΉΣΙΜΕΣ ΕΝΤΟΛΈΣ

3.2.1 ΔΗΜΙΟΥΡΓΊΑ ΚΌΜΒΩΝ

- **ΔΗΜΙΟΥΡΓΊΑ ΕΝΌΣ ΚΌΜΒΟΥ:**

  create(n) *; δημιουργία node*

  match(n) return n *; εμφάνιση node στο πρόγραμμα*

- **ΔΗΜΙΟΥΡΓΊΑ ΕΝΌΣ ΚΌΜΒΟΥ ΜΕ ΕΤΙΚΈΤΑ:**

  create(e:employee) return e *; δημιουργία κόμβου με όνομα (ετικέτα) **employee***

- **ΔΗΜΙΟΥΡΓΊΑ ΠΟΛΛΑΠΛΏΝ ΚΌΜΒΩΝ ΜΕ ΕΤΙΚΈΤΕΣ:**

  create(e:employee),(d:department) return e,d *; έχουμε 2 κόμβους στο ίδιο **query***

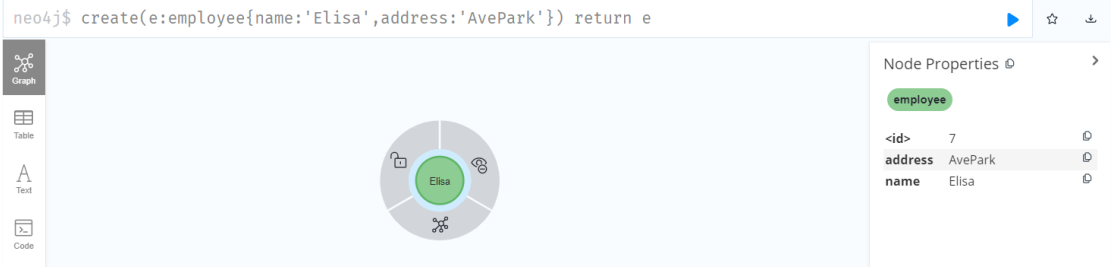
- **ΔΗΜΙΟΥΡΓΊΑ ΚΌΜΒΩΝ ΜΕ ΠΟΛΛΈΣ ΕΤΙΚΈΤΕΣ:**

  create(e:employee:professor) return e *; μπορούμε να προσθέσουμε και άλλες ιδιότητες μέσα στην παρένθεση χωρίς να χρειαστεί δεύτερη. Αυτό θα δημιουργήσει και άλλη ετικέτα*

  create(e:employee:professor:proctor) return e *; μπορούμε να προσθέσουμε περισσότερα περιεχόμενα στην παρένθεση και το πρόγραμμα θα τα προσθέσει αυτόματα σαν ετικέτες. Για παράδειγμα **employee, professor, proctor***

- **ΔΗΜΙΟΥΡΓΊΑ ΚΌΜΒΩΝ ΜΕ ΙΔΙΌΤΗΤΕΣ:**

  create(e:employee{name:'Elisa',address:'AvePark'}) return e *; οι κόμβοι, εκτός από ετικέτες μπορούν να έχουν και ιδιότητες, οι οποίες ορίζονται ως {property:'TEXT'}. Σαν και τις πολλαπλές ετικέτες, μπορούμε να βάλουμε όσες θέλουμε, αρκεί να περιέχονται μέσα στην ίδια αγκύλη και να χωρίζονται μεταξύ τους με κόμμα. Μεγάλη προσοχή στο ότι εκτός από την ίδια αγκύλη, οι ιδιότητες πρέπει να περιέχονται μέσα σε μία ετικέτα! π.χ. (e: employee{name:'Elisa'})*



- **ΔΗΜΙΟΥΡΓΊΑ ΠΟΛΛΑΠΛΏΝ ΚΌΜΒΩΝ ΜΕ ΙΔΙΌΤΗΤΕΣ:**
  create(e:employee{name:'Tom'}),(d:department{name:'Walmart'}) return e,d ***; όπως εξηγήσαμε παραπάνω, μπορούμε να έχουμε πολλές ιδιότητες οι οποίες περιγράφουν μία ετικέτα. Μπορούμε να προσθέσουμε όσες ιδιότητες θέλουμε σε μια ετικέτα αρκεί να είναι μέσα σε αγκύλες και να χωρίζονται μεταξύ τους με κόμμα. Αυτός ο κανόνας επίσης ισχύει και για πολλές ετικέτες με πολλές ιδιότητες. Πχ: (e:employee{name:'Tom'}) (ο κόμβος με ετικέτα employee και ιδιότητα το όνομα Tom) ο οποίος χωρίζεται με κόμμα από τον επόμενο κόμβο (d:department{name:'Walmart'}) με ετικέτα department και ιδιότητα το όνομα Walmart.***

3.2.2 ΕΠΙΣΤΡΟΦΉ ΚΌΜΒΩΝ

Εφόσον μάθαμε πώς μπορούμε να δημιουργήσουμε κόμβους στο πρόγραμμα Neo4J, τώρα θα προχωρήσουμε στο να μάθουμε πως να επιστρέψουμε τους κόμβους αυτούς στο πρόγραμμα.

- **ΕΠΙΣΤΡΟΦΉ ΌΛΩΝ ΤΩΝ ΚΌΜΒΩΝ**

  match(n) return n ***; το match(n) return n επιστρέφει όλους τους κόμβους που έχουμε δημιουργήσει στο πρόγραμμα.***

- **ΕΠΙΣΤΡΟΦΉ ΌΛΩΝ ΤΩΝ ΚΌΜΒΩΝ ΜΕ ΜΙΑ ΕΤΙΚΈΤΑ**

  match(e:employee) return e ***; το match, όπως αναφέραμε προηγουμένως, επιστρέφει τους κόμβους που έχουν δημιουργηθεί στο πρόγραμμα. Όταν προσθέτουμε μια ετικέτα, όπως για παράδειγμα match(e:employee) τότε το πρόγραμμα θα επιστρέψει όλους τους κόμβους που έχουν ως ετικέτα το employee. Προσοχή, το return πρέπει να περιέχει πάντα τις ετικέτες που θέλουμε να επιστρέψουμε.***

  Παράδειγμα:

  match(e:employee),(d:department) return e,d ***; γίνεται match το οι κόμβοι που έχουν ως ετικέτα το department με τους κόμβους που έχουν ως ετικέτα το employee***

- **ΕΠΙΣΤΡΟΦΉ ΌΛΩΝ ΤΩΝ ΚΌΜΒΩΝ ΕΝΌΣ ΣΥΓΚΕΚΡΙΜΈΝΟΥ ΤΎΠΟΥ**

  match(e:employee) return e.name ***; κάνουμε match τους κόμβους οι οποίοι έχουν ως ετικέτα το employee. Το πρόγραμμα επιστρέφει το όνομα τους.***

- **ΕΠΙΣΤΡΟΦΉ ΤΗΣ ΙΔΙΌΤΗΤΑΣ NAME ΌΛΩΝ ΤΩΝ ΚΌΜΒΩΝ ΜΕ ΌΛΕΣ ΤΙΣ ΕΤΙΚΈΤΕΣ**

  match(e:employee),(d:department) return e.name,d.name ***; επιστρέφει όλα τα ονόματα που είναι καταχωρημένα στους κόμβους με τις ετικέτες employee και department !***

3.2.3 ΔΙΑΓΡΑΦΉ ΚΌΜΒΩΝ

Εφόσον μάθαμε και πως να δημιουργούμε κόμβους και πως να τους επιστρέφουμε στο πρόγραμμα με τις κατάλληλες εντολές, τώρα θα μάθουμε πως τους διαγράφουμε από το database.

- **ΔΙΑΓΡΑΦΉ ΕΝΌΣ ΚΌΜΒΟΥ**

  match(e:clerk{firstname:'Tom'}) delete e ***; διαγράφουμε τον κόμβο Tom από το database.***

- **ΔΙΑΓΡΑΦΉ ΌΛΩΝ ΤΩΝ ΚΌΜΒΩΝ**

  match(n) delete(n) ***; διαγραφή όλων των κόμβων από το database.***

3.2.4 ΕΠΙΠΛΈΟΝ ΕΝΤΟΛΈΣ

- **ΕΠΙΛΟΓΉ ΕΝΌΣ ΣΥΓΚΕΚΡΙΜΈΝΟΥ EMPLOYEE**

  υπάρχουν δυο τρόποι να πάρουμε έναν συγκεκριμένο κόμβο. Στο συγκεκριμένο παράδειγμα θα πάρουμε τον κόμβο Elisa.

  1) match(e:employee{name:'Elisa'}) return e ***; επιστρέφουμε συγκεκριμένα τον κόμβο Elisa, χρησιμοποιώντας ετικέτες και ιδιότητες.***

  2) match(n) where n.name='Elisa' return n ***; επιστρέφουμε όλους τους κόμβους αλλά βάζουμε μια παράμετρο που θα επιστρέψει μόνο το κόμβο που το όνομα του είναι Elisa.***

- **ΠΩΣ ΒΆΖΟΥΜΕ ΜΙΑ ΣΥΓΚΕΚΡΙΜΈΝΗ ΙΔΙΌΤΗΤΑ ΣΕ ΈΝΑΝ ΣΥΓΚΕΚΡΙΜΈΝΟ EMPLOYEE (SET)**

  match(e:employee{name:'Tom'}) set e.salary=20000 return e ***; στον κόμβο Tom, βάζουμε την ιδιότητα μισθός, η οποία παίρνει έναν ακέραιο (Integer) αριθμό με την εντολή set.***

- **ΑΦΑΊΡΕΣΗ ΜΙΑΣ ΙΔΙΌΤΗΤΑΣ**

  match(e:employee{name:'Tom'}) remove e.salary return e ***; σε αντίθεση με το set, το remove το χρησιμοποιούμε όταν θέλουμε να αφαιρέσουμε μια ιδιότητα από το database. Για παράδειγμα, τον μισθό του Τομ.***

- **ΜΕΤΟΝΟΜΑΣΊΑ ΕΝΌΣ ATTRIBUTE / ΕΝΗΜΈΡΩΣΗ ΜΙΑΣ ΙΔΙΌΤΗΤΑΣ ΣΕ ΈΝΑΝ ΚΌΜΒΟ**

  match(e:employee{name:'Tom'}) remove e.name set e.firstname='Tom' return e ***; μπορούμε επίσης να χρησιμοποιήσουμε το remove και το set για να ενημερώσουμε / διορθώσουμε μια ιδιότητα σε έναν κόμβο, για παράδειγμα, το name σε firstname.***

- **ΕΝΗΜΈΡΩΣΗ ΠΟΛΛΏΝ ΙΔΙΟΤΉΤΩΝ ΣΕ ΈΝΑΝ ΚΌΜΒΟ**

  match(e:employee{firstname:'Tom'}) remove e:employee set e:clerk return e ***; εδώ θέλουμε να αλλάξουμε την ετικέτα employee σε clerk. Μπορούμε να το επιτύχουμε χρησιμοποιώντας το remove e:employee (δηλαδή της ετικέτας που έχουμε) και set της ετικέτας clerk. Προσοχή το e: στην αρχή πρέπει να παραμείνει ίδιο!***

3.2.5 ΧΡΉΣΗ ΛΟΓΙΚΏΝ ΠΡΆΞΕΩΝ AND, OR , IN

- **create**(e:employee{name:**'Chris'**,salary:**20000**,address:**'Liberty'**})**return** e ***; δημιουργούμε τον κόμβο Chris, ο οποίος έχει την ετικέτα employee (δηλωμένη με το e:employee) και ιδιότητες το όνομα, τον μισθό και την διεύθυνση του {name:'Chris',salary:20000,address:'Liberty'}. Για να μας επιστραφεί στο πρόγραμμα, βάζουμε το return e***

- **match**(e:employee) **where** e.name **in** [**'Chris'**, **'Ford'**] **return** e *; όταν χρησιμοποιούμε την εντολή where, το πρόγραμμα μας επιστρέφει όλους τους κόμβους οι οποίοι έχουν εξερχόμενες σχέσεις με τον κόμβο Chris και Ford.*

- **match**(e:employee) **where** e.name=**'Chris' and** e.salary=**20000 return** e ***; επιστρέφονται όλοι οι κόμβοι μόνο όταν περιέχουν τις ιδιότητες όνομα Chris και τον μισθό 20000.***

- **match**(e:employee) **where** e.name=**'Chris' or** e.name=**'Ford' return** e ***; επιστρέφονται όλοι οι κόμβοι μόνο όταν περιέχουν τις ιδιότητες όνομα Chris ή το όνομα Ford.***

3.2.6 ΑΣΚΉΣΕΙΣ

ΆΣΚΗΣΗ 1

Δημιουργήστε 3 employees τον Scott, τον Sam και τον John και 2 departments με τα ονόματα Scope και BBD. Ο Scott δουλεύει για το Scope, Ο Sam για το BBD και ο John διευθύνει το Scope και το BBD. Να φτιάξετε στο πρόγραμμα τους κόμβους και να τους ενώσετε με τις σχέσεις WORKS FOR και MANAGES.

ΛΎΣΗ

```
create(e:employee{name:'Scott',salary:30000,address:'vellore'})
```

πρώτα δημιουργούμε τον κόμβο Scott και του δίνουμε τις ιδιότητες όνομα, μισθό και διεύθυνση.

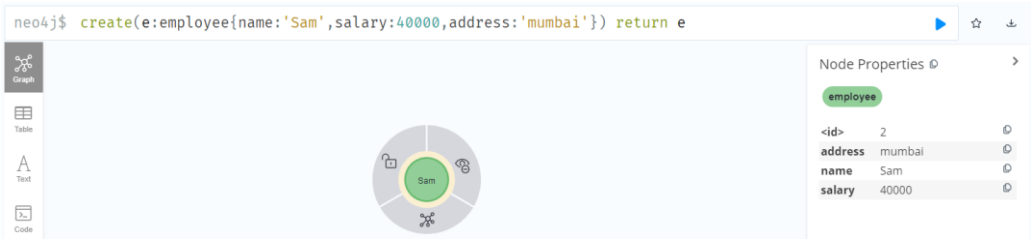
```
neo4j$ create(e:employee{name:'Scott',salary:30000,address:'vellore'})
```

Table

Added 1 label, created 1 node, set 3 properties, completed after 352 ms.

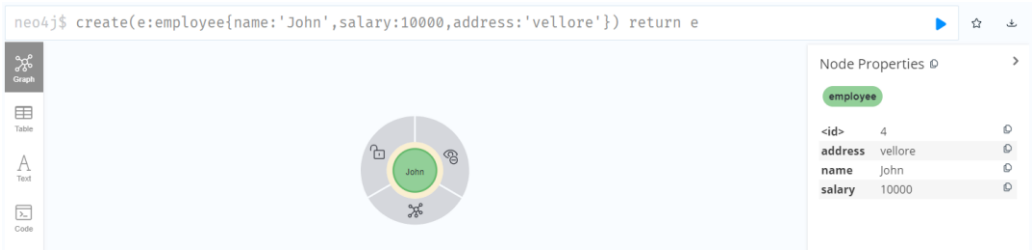
```
create(e:employee{name:'Sam',salary:40000,address:'mumbai'}) return e
```

; με τον ίδιο τρόπο φτιάχνουμε τον κόμβο Sam και του προσθέτουμε τις ίδιες ιδιότητες.




```
create(e:employee{name:'John',salary:10000,address:'vellore'}) return e
```

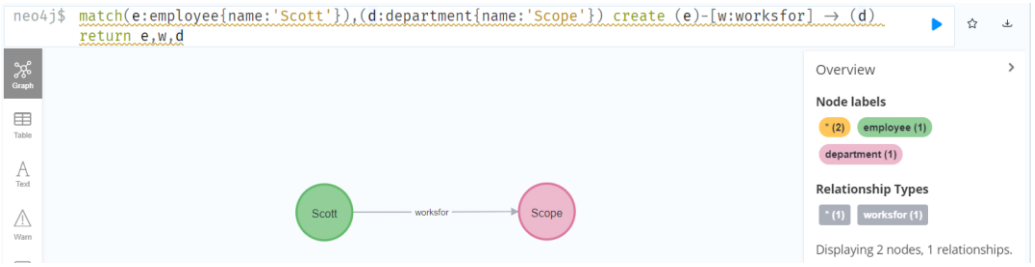
; με τον ίδιο τρόπο φτιάχνουμε και τον John και του προσθέτουμε τις ίδιες ιδιότητες με τον Sam και Scott



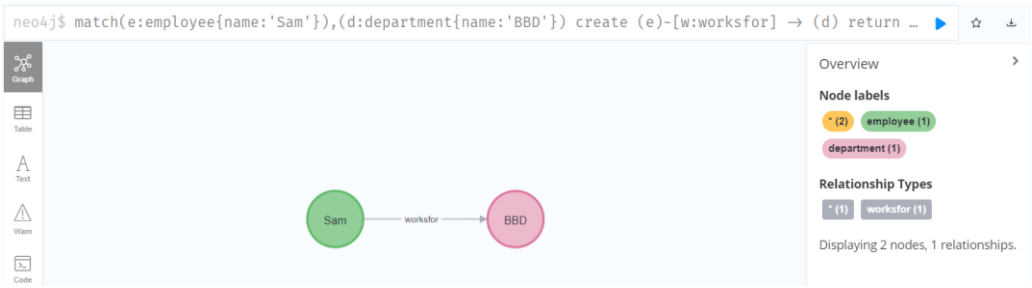
create(e:department{name:**'Scope'**}) **return** e ***; δημιουργούμε το department Scope***

create(e:department{name:**'BBD'**}) **return** e ***; παρομοίως δημιουργούμε το BBD***

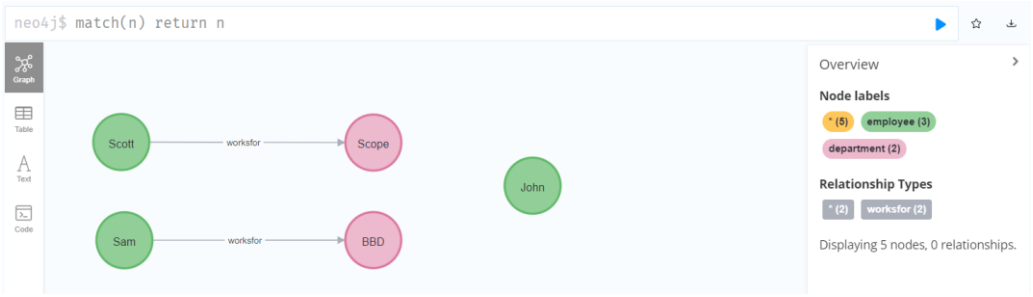
match(e:employee{name:**'Scott'**}),(d:department{name:**'Scope'**})
create (e)-[w:worksfor] -> (d) **return** e,w,d ***; ενώνουμε τους κόμβους Scott και Scope με την σχέση works for. Για να μας επιστραφεί στο πρόγραμμα, κάνουμε return το e, για τον employee, το w για την σχέση worksfor και το d για το department.***



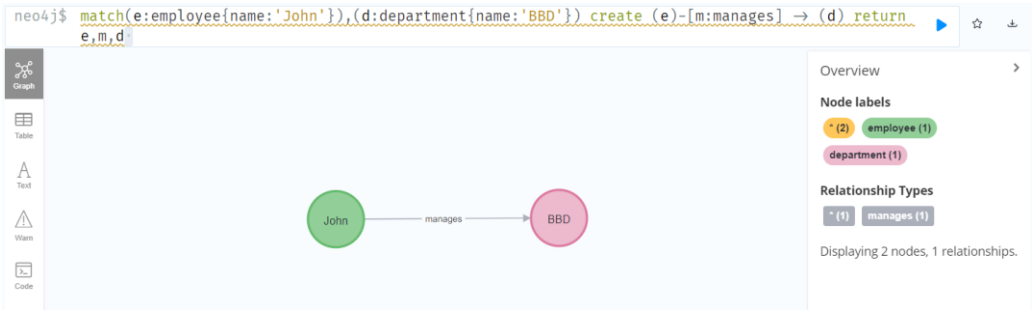
```
match(e:employee{name:'Sam'}),(d:department{name:'BBD'})
create (e)-[w:worksfor] -> (d)
return e,w,d
```
; με τον ίδιο τρόπο ενώνουμε τον Sam και το BBD με την σχέση works for.



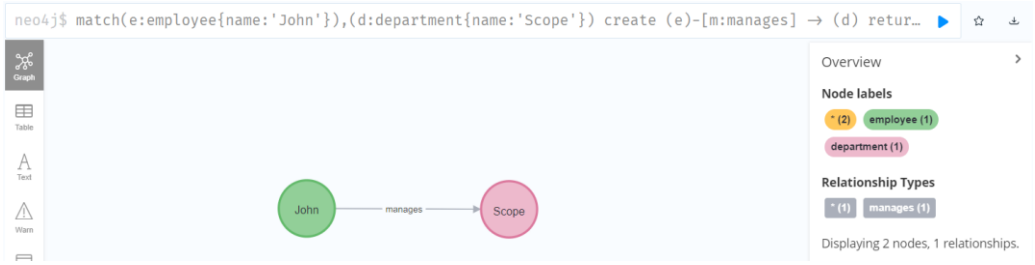

```
match(n) return n
```
; το πρόγραμμα μας επιστρέφει όλους τους κόμβους και τις σχέσεις μας μέχρι τώρα.



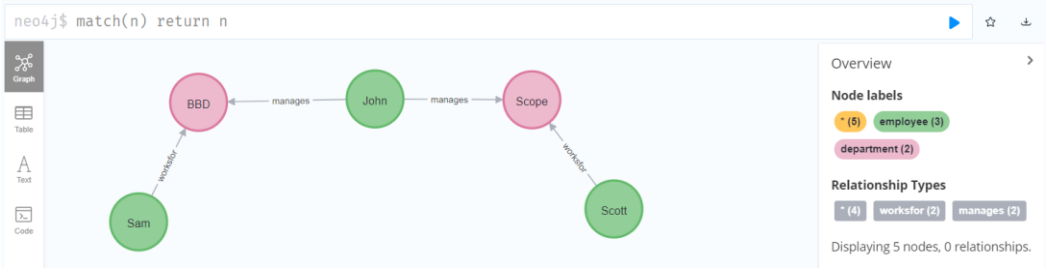

```
match(e:employee{name:'John'}),(d:department{name:'BBD'})
create (e)-[m:manages] -> (d)
return e,m,d
```
; ενώνουμε τον John και το BBD με μία σχέση manages.



match(e:employee{name:**'John'**}),(d:department{name:**'Scope'**})
create (e)-[m:manages] -> (d)
return e,m,d ***; ενώνουμε τον John και το Scope με μία σχέση manages.***

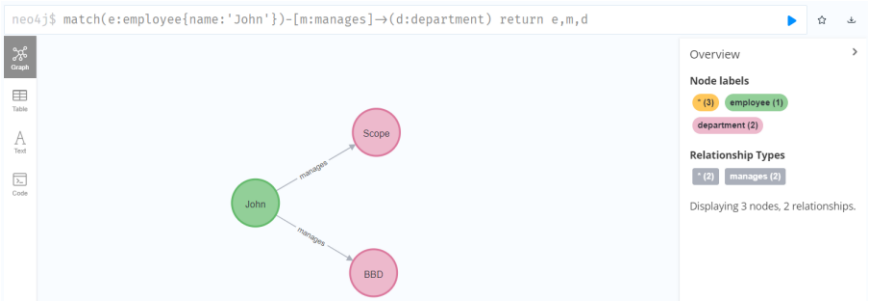


match(n) return n ***; το πρόγραμμα μας επιστρέφει όλους τους κόμβους και τις σχέσεις μας μέχρι τώρα.***



match(e:employee)-[w:worksfor]->(d:department{name:'scope'}) return e,w,d

match(e:employee{name:**'John'**})-[m:manages]->(d:department)
return e,m,d ***; εμφανίζεται μόνο ο κόμβος John και οι σχέσεις manage που έχει με όλα τα departments.***



ΠΡΌΣΘΕΣΗ ΚΑΙ ΑΦΑΊΡΕΣΗ ΙΔΙΟΤΉΤΩΝ ΣΤΙΣ ΣΧΈΣΕΙΣ

match(e:employee{name:'Sam'})-[w:worksfor]->(d:department{name:'BBD'})
set w.joineddate=2019 return e,w,d ***; στην ήδη υπάρχοντα σχέση μεταξύ του Sam και του department BBD θέλουμε να προσθέσουμε την ιδιότητα joined date (πότε μπήκε στο department). Αυτό είναι εφικτό με την χρήση της εντολής "set w.[property]=" της οποίας η τιμή μπορεί να πάρει είτε χαρακτήρα είτε integer.***

match(e:employee{name:'Sam'})-[w:worksfor]->(d:department{name:BBD'})
remove w.joineddate return e,w,d ***; αν από την άλλη θέλουμε να αφαιρέσουμε το πότε μπήκε στο department BBD, αντί για set βάζουμε remove.***

ΑΦΑΊΡΕΣΗ ΣΧΈΣΕΩΝ

match(e:employee{name:'sam'})-[w:worksfor]->(d:department{name:'sbst'})
delete w return e,d ***; για να διαγράψουμε μια σχέση, για παράδειγμα του Sam που δουλεύει στο BBD, βάζουμε delete w, το οποίο w αντιπροσωπεύει την σχέση worksfor ([w.worksfor]) και ύστερα προσθέτουμε το return e,d δηλαδή τον employee και το department εφόσον η σχέση worksfor αφαιρέθηκε.***

match(n) detach delete n ***; εάν θέλουμε να διαγράψουμε όλους τους κόμβους, πρώτα αφαιρούμε τις σχέσεις με την εντολή detach και ύστερα γράφουμε την εντολή όπως την ξέρουμε, δηλαδή match(n) delete n. Προσοχή διότι αν δεν έχουν αφαιρεθεί οι σχέσεις δεν γίνεται να διαγράψουμε κόμβους.***

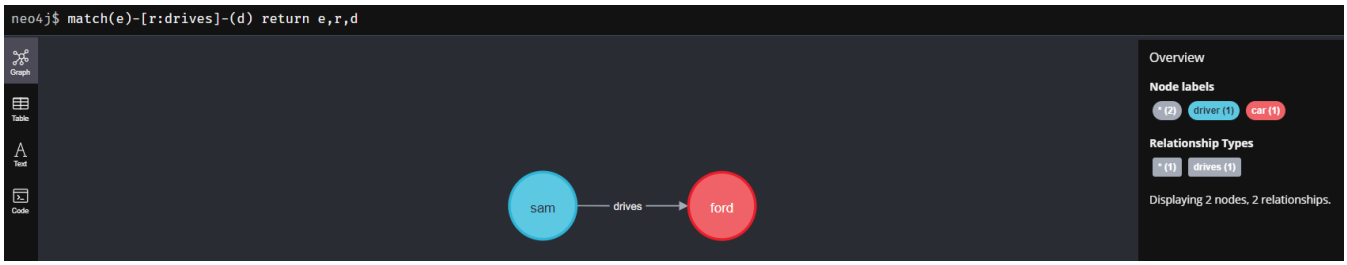
ΆΣΚΗΣΗ 2

Δημιουργήστε 2 κόμβους, τον Οδηγό και το Αυτοκίνητο και:
1) φτιάξτε την σχέση drives καθώς φτιάχνετε τους κόμβους.
2) επιστρέψτε όλους τους κόμβους που περιέχουν την σχέση drives
3) διαγράψτε την σχέση drives μεταξύ δύο οποιοδήποτε κόμβων
4) διαγράψτε την σχέση drives χωρίς να διαγράψετε τους κόμβους.

ΛΎΣΗ

1) **create**(d:driver{name:**'sam'**})-[r:drives] -> (c:car{name: **'ford'**})
return d,r,c ***; πρώτα δημιουργούμε τον sam (οδηγός), το ford (αυτοκίνητο) και τα ενώνουμε με την σχέση drives, η οποία, όπως γνωρίζουμε, μπαίνει μέσα σε αγκύλες και δείχνει προς τον κόμβο ford (το αυτοκίνητο).***

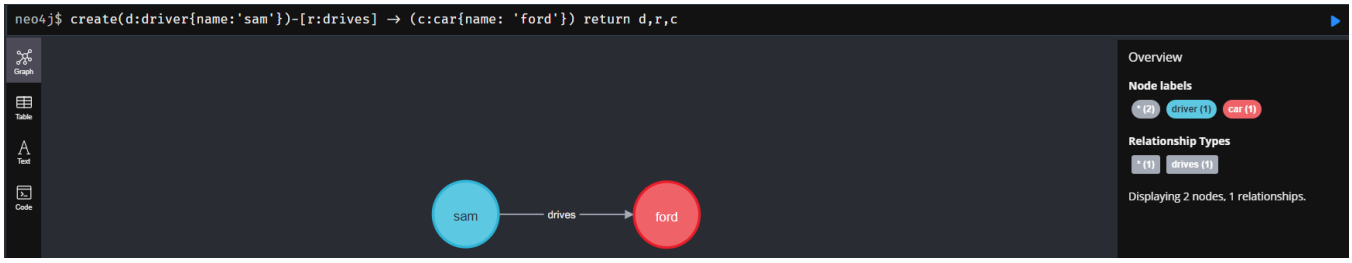
2) **match**(e)-[r:drives]-(d)
return e,r,d ***; κάνοντας match(e) και βάζοντας το drive στις αγκύλες, μας επιστρέφονται όλοι οι κόμβοι οι οποίοι συνδέονται με την σχέση drives.***



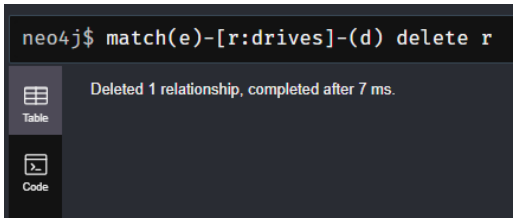
3) **match**(n) **detach delete** n ***; όπως μάθαμε για να διαγράψουμε έναν κόμβο πρέπει πρώτα να βγάλουμε την σχέση που τους ενώνει. Για να το κάνουμε αυτό, χρησιμοποιούμε την εντολή match(n) detach delete n, η οποία πρώτα θα διαγράψει την σχέση και μετά θα διαγράψει τους κόμβους.***

4)**create**(d:driver{name:**'sam'**})-[r:drives] -> (c:car{name: **'ford'**})
return d,r,c

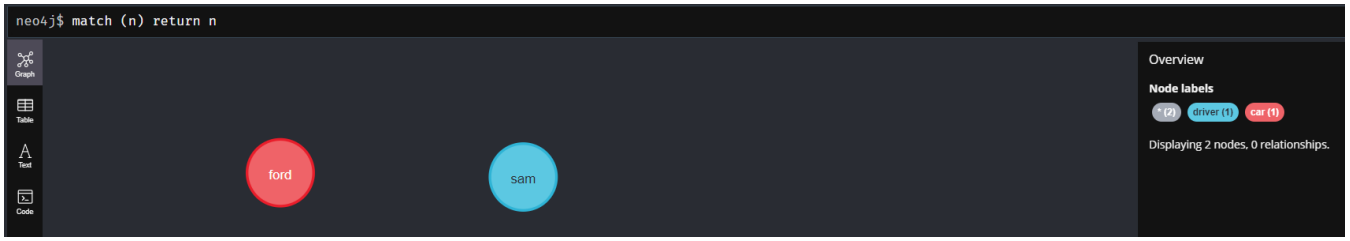
; ξαναδημιουργούμε τους κόμβους ώστε να τους χρησιμοποιήσουμε στην επόμενη εντολή



5) **match**(e)-[**r:drives**]-(d) **delete** r ***; χρησιμοποιώντας την εντολή match(e) -[r:drives]-(d) το πρόγραμμα επιστρέφει όλους τους κόμβους οι οποίοι περιέχουν την σχέση drives και, με την εντολή delete, διαγράφει την σχέση.***



6) **match** (n) **return** n ***; εφόσον κάναμε delete την σχέση, όταν κάνουμε match(n) return (n) μας επιστρέφονται όλοι οι κόμβοι, χωρίς τις σχέσεις.***



match (n) **delete** n ***; εφόσον τώρα δεν έχουμε κάποια σχέση μπορούμε να διαγράψουμε τους κόμβους με την εντολή που ξέρουμε: match(n) delete n.***

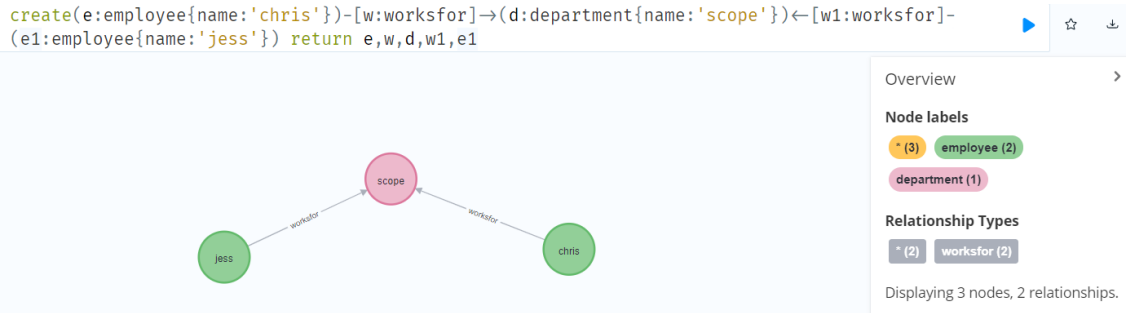
ΆΣΚΗΣΗ 3

a) Δημιουργήστε τους κόμβους JESS και CHRIS, οι οποίοι και οι δυο δουλεύουν για το department SCOPE. Ο SCOTT και ο SAM δουλεύουν για το SCOPE ενώ ο SCOTT και ο TOM διευθύνουν το BBD. Δημιουργήστε το πλήρες μονοπάτι για αυτό το σενάριο.

ΛΥΣΗ

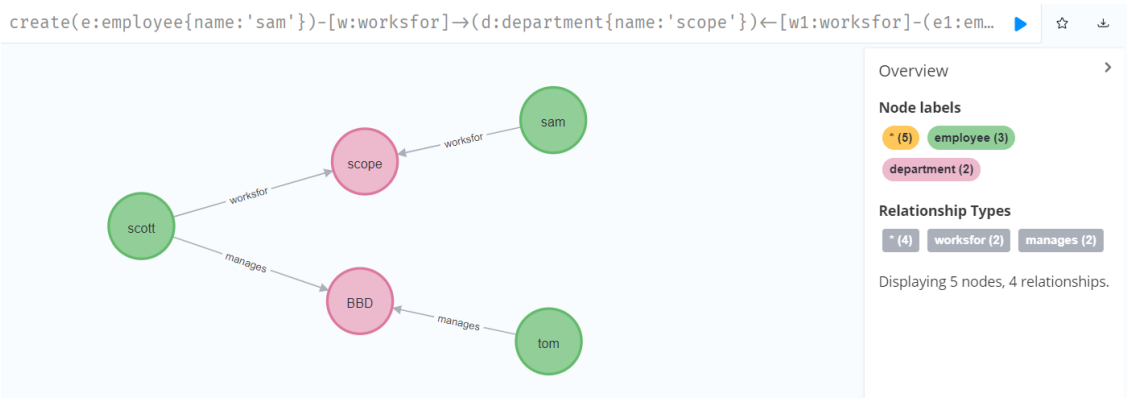
```
create(e:employee{name:'chris'})-[w:worksfor]->(d:department{name:'scope'})<-[w1:worksfor]-(e1:employee{name:'jess'}) return e,w,d,w1,e1
```

; δημιουργούμε τον κόμβο Chris και την σχέση worksfor (πάντα συνοδευμένη με το πρόθεμα w. ,τις αγκύλες της και την φορά της σχέσης προς το scope, εκεί που δουλεύει) και την Jess, την οποία με παρόμοιο τρόπο φτιάχνουμε την σχέση της με το scope (προσοχή στην σχέση πάει w1. εφόσον το w έχει χρησιμοποιηθεί ήδη, ομοίως με το e1 στο employee)

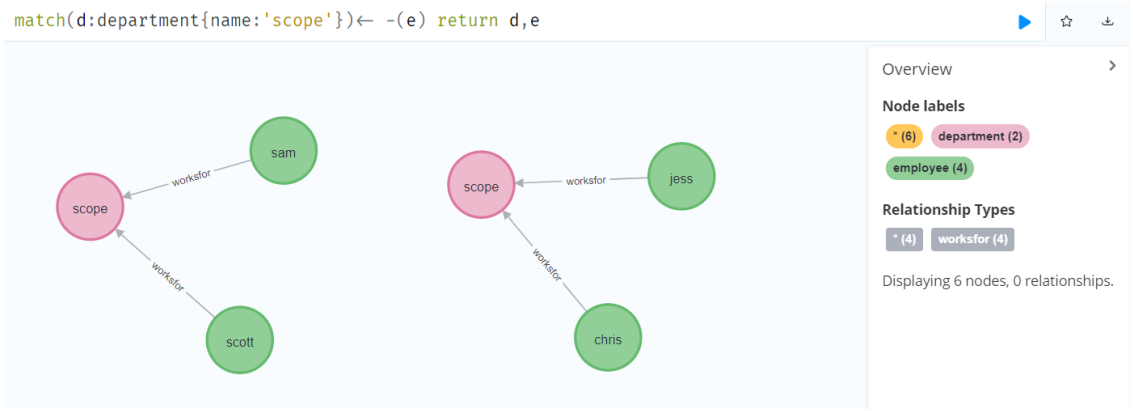



```
create(e:employee{name:'sam'})-[w:worksfor]->(d:department{name:'scope'})<-[w1:worksfor]-(e1:employee{name:'scott'})-[m:manages]->(d1:department{name:'BBD'})<-[m1:manages]-(e2:employee {name:'tom'}) return e,w,d,e1,w1,m,d1,m1,e2
```

; παρομοίως τον κόμβο Sam τον οποίο ενώνουμε (με την χρήση βέλους) με το department Scope με την σχέση worksfor. Στην ίδια εντολή δημιουργούμε τον κόμβο Scott τον οποίο ενώνουμε με το department Scope με την σχέση worksfor, και με το department BBD με την σχέση manages, εφόσον θέλουμε να το διευθύνει. Τέλος δημιουργούμε τον κόμβο Tom και τον ενώνουμε με το department BBD με την σχέση manages. Προσοχή στην χρήση των βέλων ώστε να βεβαιώσουμε την φορά της σχέσης και στο return, στο οποίο πρέπει να βάλουμε όλους τους κόμβους και τις ιδιότητες.



match(d:department{name:**'scope'**})<- -(e) **return** d,e ***; επιστρέφει όλους τους κόμβους οι οποίοι είναι συνδεμένοι με το Scope.***



match(n) **detach delete** n ***; καταστρέφουμε τις σχέσεις και διαγράφουμε τους κόμβους, εφόσον ξέρουμε ότι πρέπει πρώτα να βγάλουμε τις σχέσεις πριν διαγράψουμε τους κόμβους.***

3.2.7 MATCH

- **MATCH ΣΕ ΤΎΠΟ ΣΧΈΣΕΩΝ**

  Παράδειγμα:
  Κάντε Match όλες τις σχέσεις manages μεταξύ ενός employee και department.
  match(e)-[m:manages]->(d) **return** e,m,d

- **MATCH ΜΕ ΠΟΛΛΑΠΛΟΎ ΤΎΠΟΥΣ ΣΧΈΣΕΩΝ**

  Παράδειγμα:
  1) Επέστρεψε όλες τις σχέσεις μεταξύ του department και του employee Που είναι είτε **'worksfor'** ή **'manages'**
  match(e)-[m:manages worksfor]->(d)
  return e,m,d

  2) Επιστρέψτε τα ονόματα όλων των employees.

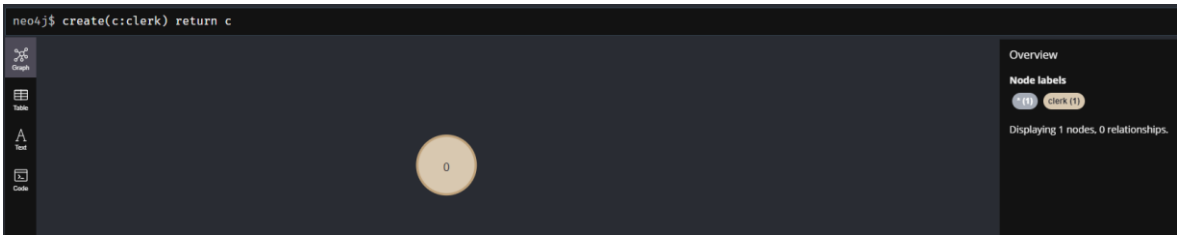
  match(e:employee) return e.name order by e.name ***; επιστρέφει τα ονόματα.***
  match(e:employee) return e.name order by e.name desc ***; επιστρέφει τα ονόματα με φθίνουσα σειρά.***
  match(e:employee) set e.salary=20000 ***; θέτουμε τον μισθό σε 20000***
  match(e:employee) return e.name,e.salary order by e.name,e.salary ***; επιστρέφει τα ονόματα με την σειρά και τον μισθό***
  match(e:emp) return e.name skip 2 ***; επιστρέφει τα ονόματα με την σειρά, κάνοντας skip τα δύο πρώτα στοιχεία της λίστας.***
  match(e:emp) return e.name limit 2 ***; επιστρέφει τα ονόματα δείχνοντας τα 2 προηγούμενα στοιχεία της λίστας.***
  match(e:emp) return e.name union match(e:emp) return e.name ***; το union επιστρέφει τα ονόματα από δυο διαφορετικά queries. Με την χρήση του union αυτά τα queries συνδυάζονται.***
  match(e:emp) return e.name union all match(e:emp) return e.name ***; όπως είπαμε παραπάνω, το union επιστρέφει τα ονόματα από δυο διαφορετικά queries. Η διαφορά του Union με το Union all είναι ότι το union all επιστρέφει τα ονόματα και κρατάει τα εις διπλούν.***

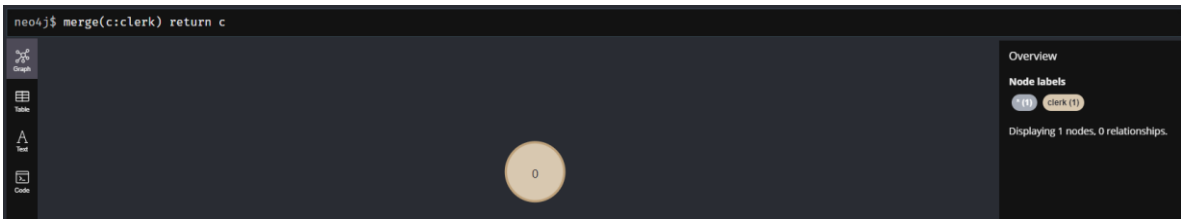
3.2.8 MERGE

Το Merge είτε ταιριάζει τους υπάρχοντες κόμβους και τους δεσμεύει, είτε δημιουργεί νέα δεδομένα και δεσμεύει αυτά. Είναι σαν ένας συνδυασμός του MATCH και του CREATE που μας επιτρέπει επιπλέον να καθορίσουμε τι θα συμβεί εάν τα δεδομένα αντιστοιχιστούν η δημιουργηθούν.

Παράδειγμα:

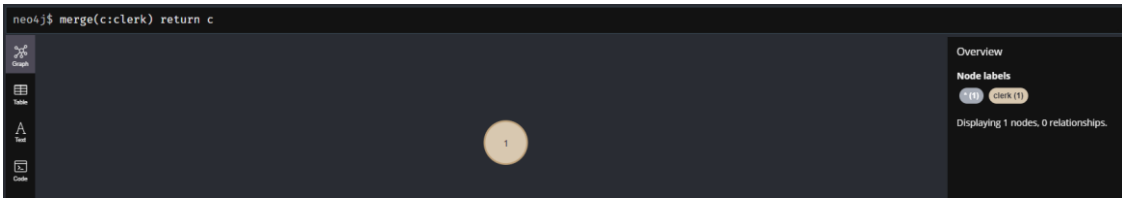
create(c:clerk) return c ***; δημιουργούμε τον κόμβο clerk***
merge(c:clerk) return c ***; εάν κάνουμε merge(c:clerk) return c, τότε δεν θα δημιουργήσει άλλον κόμβο, αλλά μας επιστρέφει αυτόν που έχουμε δημιουργήσει ήδη. Άρα δηλαδή δρα ως την εντολή MATCH.***





Τώρα ας υποθέσουμε ότι δεν έχουμε δημιουργήσει τον κόμβο clerk. Εάν βάλουμε την παρακάτω εντολή στο πρόγραμμα, τότε το MERGE θα δράσει ως CREATE και θα μας επιστρέψει τον κόμβο clerk.

match(c:clerk) delete c ***; διαγράφουμε τον κόμβο clerk που είχαμε φτιάξει***
merge(c:clerk) return c ***; το merge δρα ως την εντολή CREATE τώρα διότι ο κόμβος clerk που θέλουμε να κάνουμε merge δεν υπάρχει.***



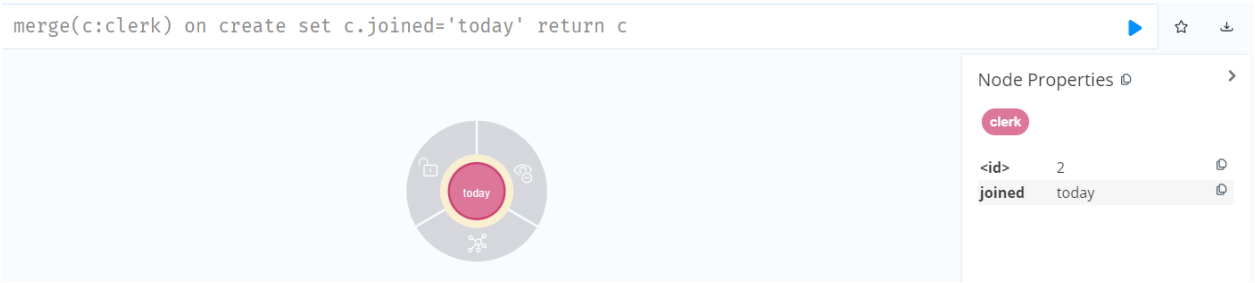
Με βάση τα screenshots, βλέπουμε ότι όντως στο πρώτο παράδειγμα το MERGE έδρασε ως την εντολή MATCH ενώ στο δεύτερο παράδειγμα, που δεν είχαμε δημιουργήσει ήδη τον κόμβο clerk, τότε το MERGE έδρασε ως την εντολή CREATE.

- ## Η ΕΝΤΟΛΉ MERGE ON CREATE

Συγχωνεύει έναν κόμβο και ορίζει ιδιότητες εάν χρειάζεται να δημιουργηθούν οι κόμβοι. Μια ιδιότητα μπορεί να οριστεί για έναν νέο κόμβο **μόνο** όταν δημιουργείται χρησιμοποιώντας την εντολή MERGE.

Παράδειγμα:

merge(c:clerk) on create set c.joined='today' return c ***; εδώ το πρόγραμμα θα ελέγξει εάν υπάρχει η μεταβλητή clerk. Εάν δεν υπάρχει θα την δημιουργήσει και θα της βάλει την ιδιότητα joined, που θα πάρει την τιμή 'today'.***



merge(c:clerk) on create set c.joined='yesterday' return c ***; τρέχουμε την εντολή ξανά, όμως τώρα αλλάζουμε το today με το yesterday. Αυτό που θα γίνει, είναι το πρόγραμμα θα ελέγξει εάν υπάρχει ο κόμβος clerk και αν δεν υπάρχει θα του βάλει την Ιδιότητα yesterday. Ωστόσο, έχουμε ήδη δημιουργήσει τον κόμβο, άρα υπάρχει, άρα η ιδιότητα yesterday δεν θα αντικαταστήσει το today.***

- ## Η ΕΝΤΟΛΉ MERGE ON MATCH

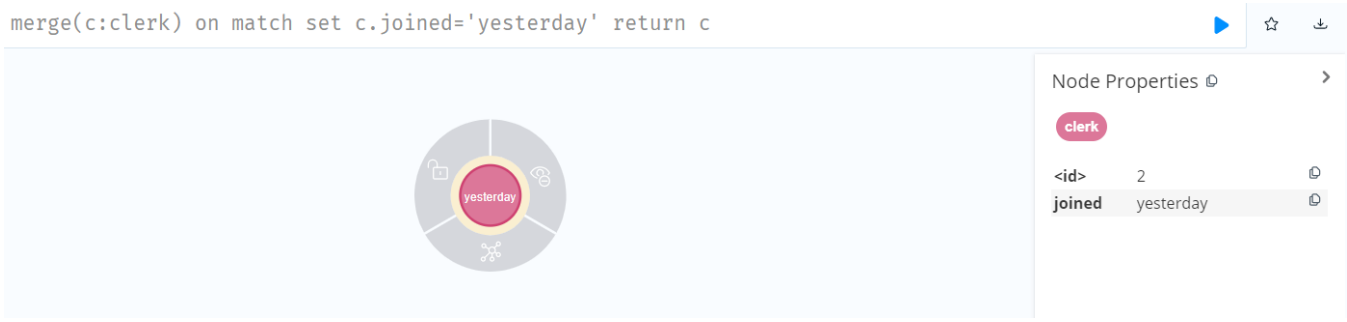
Χρησιμοποιείται για την συγχώνευση κόμβων και την ρύθμιση ιδιοτήτων σε κόμβους που βρέθηκαν.

ΆΣΚΗΣΗ

Να ορίσετε μια ιδιότητα για έναν κόμβο μόνο εάν υπάρχει ήδη χρησιμοποιώντας τη εντολή MERGE ON MATCH.

ΛΎΣΗ

merge(c:clerk) on match set c.joined='yesterday' return c ***; με την εντολή on match όταν το πρόγραμμα εντοπίσει ότι υπάρχει ο κόμβος διαθέσιμος τότε μόνο θα του βάλει την δοσμένη ιδιότητα. Εφόσον ο κόμβος clerk υπάρχει, διότι τον είχαμε χρησιμοποιήσει για την προηγούμενη άσκηση, τότε ναι, το πρόγραμμα θα πάει και θα του βάλει την ιδιότητα yesterday.***

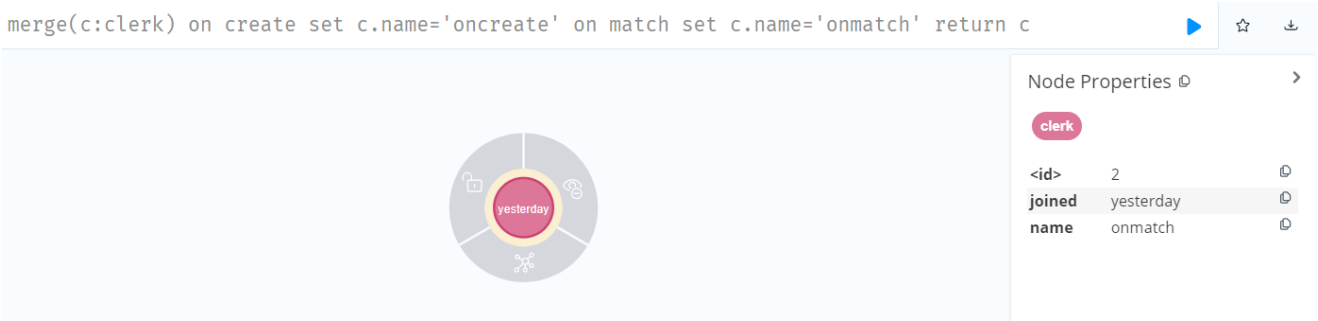


- ## MERGE ON MATCH ΚΑΙ MERGE ON CREATE

Εάν ο κόμβος είναι νέος τότε εκτελείται η εντολή ON CREATE διαφορετικά εάν ο κόμβος υπάρχει ήδη, τότε εκτελείται η εντολή ON MATCH

Παράδειγμα:

merge(c:clerk) on create set c.name='oncreate' on match set c.name='onmatch' return c ***; εδώ το πρόγραμμα θα ελέγξει εάν ο κόμβος clerk υπάρχει. Εάν δεν υπάρχει, τότε θα δημιουργήσει τον κόμβο και μια ιδιότητα name, που θα παίρνει τιμή oncreate. Εάν υπάρχει, τότε θα τον δημιουργήσει εκείνος και θα του βάλει την ιδιότητα name, που θα παίρνει τιμή 'onmatch'. Επειδή υπάρχει ήδη ο κόμβος μας, τότε θα λειτουργήσει η εντολή on match.***

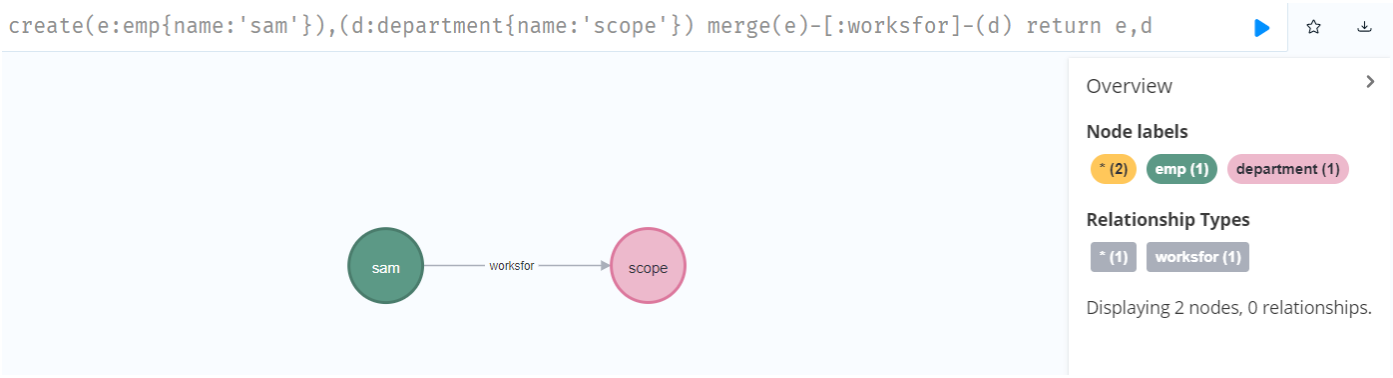


- ## MERGE ΣΕ ΜΙΑ ΣΧΕΣΗ

Το MERGE μπορεί να χρησιμοποιηθεί αντίστοιχα για την δημιουργία μιας σχέσης.

Παράδειγμα:

create(e:emp{name:'sam'}),(d:department{name:'scope'}) merge(e)-[:worksfor]-(d) return e,d



- ## AGGREGATE FUNCTIONS

NAME	SALARY	GENDER	DEPARTMENT
SAM	20000	MALE	SCOPE
ROMEO	60000	MALE	BBD
JANE	30000	FEMALE	SITE
THOMAS	40000	MALE	SCOPE

NAME	LOCATION
SCOPE	GREECE
BBD	FRANCE
SITE	GERMANY

ΆΣΚΗΣΗ

Με βάση τους πίνακες παραπάνω:

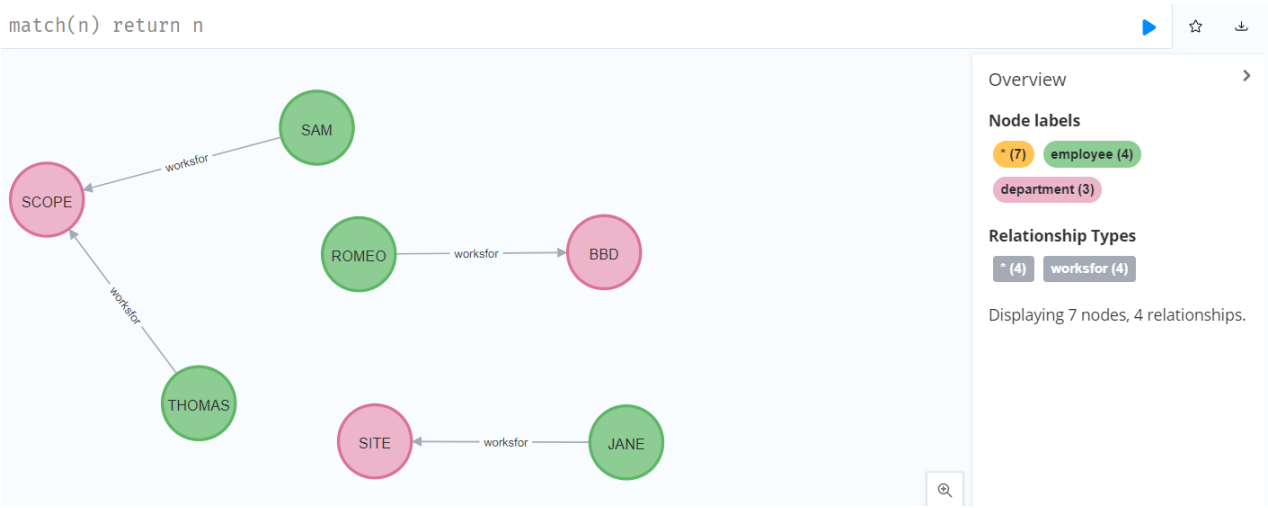
1) Μετρήστε τον αριθμό τον employees στον οργανισμό
2) Βρείτε τους employees με τον ελάχιστο και μέγιστο μισθό
3) Επιστρέψτε στο πρόγραμμα το SUM του μισθού όλων των employees
4) Επιστρέψτε στο πρόγραμμα τον τελικό αριθμό των άντρων employees στον οργανισμό
5) Επιστρέψτε στο πρόγραμμα τον τελικό αριθμό των άντρων employees για κάθε department
6) Επιστρέψτε στο πρόγραμμα τον τελικό αριθμό των άντρων employees για κάθε department με αύξουσα σειρά

ΛΥΣΗ

create(e:employee{name:'SAM',gender:'MALE',salary:20000})-[w:worksfor]->(d:department{name:'SCOPE',location:'GREECE'})<-[w1:worksfor]-(e1:employee{name:'THOMAS',gender:'MALE',salary:40000}) return e,w,d,w1,e1

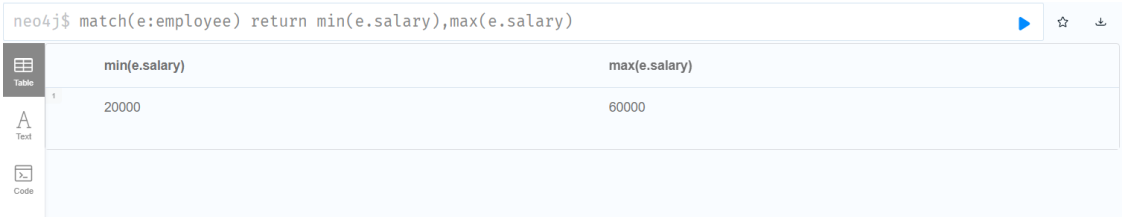
merge(e:employee{name:'ROMEO',gender:'MALE',salary:60000})-[w:worksfor]->(d:department{name:'BBD',location:'FRANCE'})

merge(e:employee{name:'JANE',gender:'FEMALE',salary:30000})-[w:worksfor]->(d:department{name:'SITE',location:'GERMANY'})

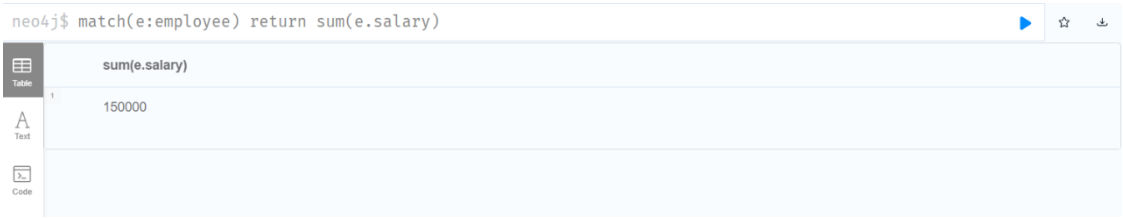


1) match(e:emp) return count(*) as totalemp ***; με την χρήση του count(*) το πρόγραμμα θα μετρήσει όλους τους employee. Έτσι, μας επιστρέφεται ένας πίνακας, που το αποτέλεσμα είναι 4.***

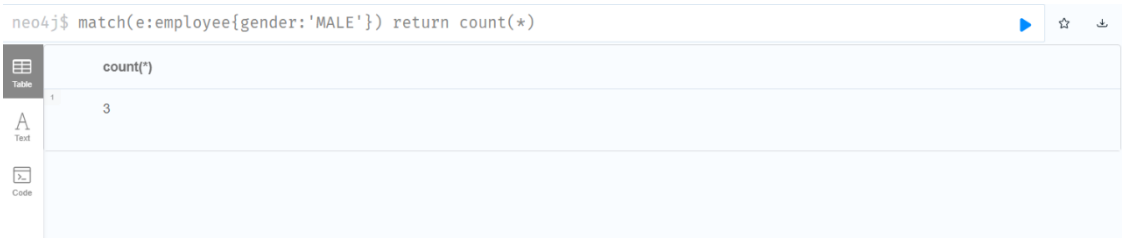
2) match(e:emp) return min(e.salary),max(e.salary) ***; υπολογίζουμε τον ελάχιστο και τον μέγιστο μισθό.***



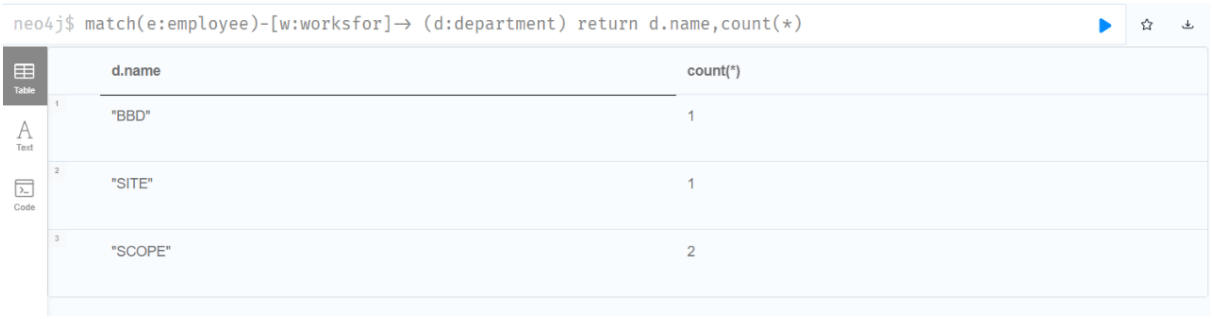
3) match(e:emp) return sum(e.salary) ***; παίρνει τους μισθούς όλων των κόμβων που είναι αποθηκευμένοι στο employee, τους προσθέτει και υπολογίζει το άθροισμα.***



4) match(e:emp{gender:'male'}) return count(*) ***; για να υπολογίσουμε τους άνδρες υπαλλήλους θα αντιστοιχίσουμε την ιδιότητα gender που υπάρχει στον πίνακα έτσι ώστε να βρούμε το πόσοι άνδρες υπάρχουν.***

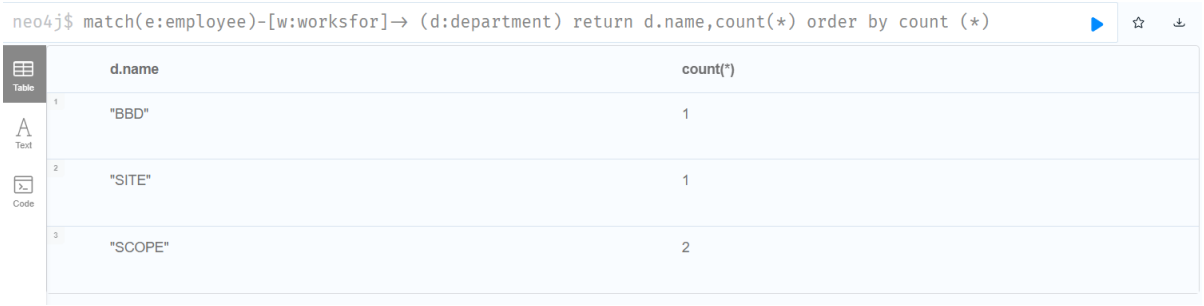


5) match(e:emp)-[w:worksfor]-> (d:dept) return d.name,count(*) ***; για να βρούμε τον αριθμό σε κάθε department θα πρέπει να ενώσουμε δυο πίνακες, και αυτό το κάνουμε με την χρήση της σχέσης worksfor.***

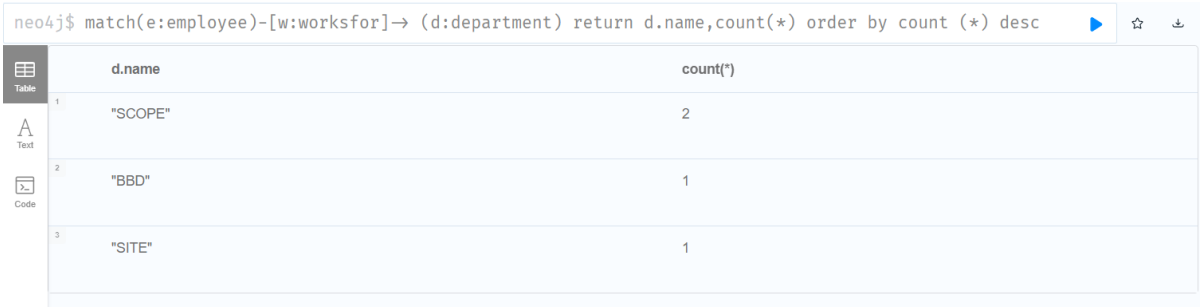


6) match(e:emp)-[w:worksfor]-> (d:dept) return d.name,count(*) order by count (*)

; για να βρούμε τον αριθμό σε κάθε department και να παρουσιάσουμε το αποτέλεσμα με αύξουσα σειρά, τότε θα χρησιμοποιήσουμε την παραπάνω εντολή με την πρόσθεση του order by count(*) που είναι η αύξηση.



7) match(e:emp)-[w:worksfor]-> (d:dept) return d.name,count(*) order by count (*) desc ***; Αν θέλουμε το αποτέλεσμα σε φθίνουσα σειρά, τότε θα προσθέσουμε στο order by count(*) το desc, που είναι η μείωση.***



3.2.9 CONSTRAINTS / ΠΕΡΙΟΡΙΣΜΟΊ

Τα Constraints είναι σαν έναν κανόνα τον οποίο εδραιώνουμε στο database μας. Ένα Constraint μπορεί είτε να είναι αριθμός πχ 123, είτε null, αλλά σε καμία περίπτωση δεν μπορεί να εμπεριέχει χαρακτήρα, πχ 123x και έχει την δυνατότητα να είναι και ιδιότητα σε έναν κόμβο, Ένα Constraint πρέπει επίσης να είναι μοναδικό, δηλαδή να μην επαναλαμβάνεται σε έναν κόμβο.

- **ΔΗΜΙΟΥΡΓΊΑ CONSTRAINTS**

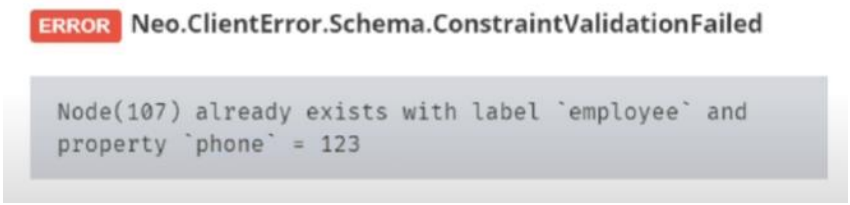
  ΜΟΝΑΔΙΚΆ CONSTRAINTS

  create(e:employee{phone:123}) return e ***; με βάση τους κόμβους που έχουμε από το προηγούμενο παράδειγμα, δημιουργούμε ένα label employee και του δίνουμε την ιδιότητα phone, που είναι 123.***

  create constraint on (e:employee) assert e.phone is unique ***; δημιουργούμε ένα constraint στο label employee, το οποίο θέτουμε ως την ιδιότητα phone και λέμε ότι είναι μοναδικό για κάθε employee.***

  call db.constraints ***; χρησιμοποιούμε αυτή την εντολή για να δούμε όλα τα constraints που έχουμε στο database μας.***

  create(e:employee{phone:123}) ***; εφόσον υπάρχει το constraint phone, αν προσπαθήσουμε να φτιάξουμε άλλον έναν employee και να του προσθέσουμε το τηλέφωνο 123 τότε θα οδηγηθούμε σε σφάλμα, καθώς, όπως ένα constraint είναι μοναδικό.***

  create(e:employee{phone:null}) ***; εάν από την άλλη θέλουμε να φτιάξουμε έναν employee που να μην έχει καθόλου τηλέφωνο, δηλαδή το e.phone να είναι null, τότε δεν θα βγει σφάλμα, διότι ένα constraint μπορεί να πάρει τιμή null.***

  create(e:employee{phone:123}) ***; επίσης δεν θα οδηγηθούμε σε σφάλμα καθώς ο employee με τηλέφωνο 124 δεν υπάρχει. Άρα το τηλέφωνο θα είναι μοναδικό.***

  match(e:employee) delete e

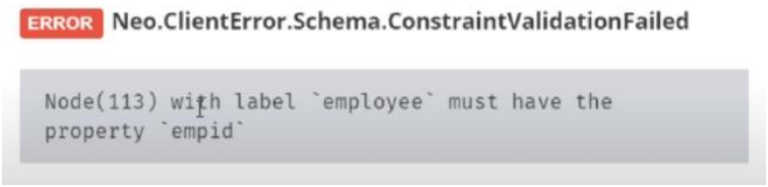
PROPERTY EXISTENCE CONSTRAINTS / ΠΕΡΙΟΡΙΣΜΟΊ ΤΩΝ ΉΔΗ ΥΠΑΡΧΌΝΤΩΝ ΙΔΙΟΤΉΤΩΝ

Διασφαλίζει ότι υπάρχει μια ιδιότητα για όλους τους κόμβους με μια συγκεκριμένη ετικέτα ή για όλες τις σχέσεις με έναν συγκεκριμένο τύπο. Όλοι οι employees πρέπει να έχουν μια ιδιότητα empid.

create(e:employee{empid:1}) ***; δημιουργούμε έναν employee ο οποίος θα έχει empid 1.***

create constraint on (e:employee) assert exists(e.empid) ***; δημιουργούμε έναν constraint και διασφαλίζουμε ότι το empid υπάρχει για κάθε employee.***

create(e:employee{age:23}) ***; αν προσπαθήσουμε να δημιουργήσουμε έναν καινούργιο employee που θα έχει την ηλικία 23, το πρόγραμμα θα μας πετάξει σφάλμα, διότι δεν θα έχει την ιδιότητα empid η οποία έχει οριστεί ως constraint και καθαυτού πρέπει να υπάρχει σε κάθε employee.***



drop constraint on (e:employee) assert (e.phone) is unique ***; με το drop μπορούμε να αφαιρέσουμε ένα constraint. Σε αυτή την περίπτωση αφαιρούμε τον αριθμό τηλεφώνου που είχαμε ορίσει στο προηγούμενο παράδειγμα.***

drop constraint on (e:employee) assert exists(e.empid) ***; με τον ίδιο τρόπο αφαιρούμε και το empid, το constraint που έχουμε ορίσει σε αυτό το παράδειγμα.***

match(e:employee) delete e

NODE KEYS

Διασφαλίζουν ότι όλοι οι κόμβοι με μια συγκεκριμένη ετικέτα έχουν ένα σύνολο καθορισμένων ιδιοτήτων των οποίων η τιμή είναι μοναδική και όπου υπάρχουν όλες οι ιδιότητες στο σύνολο – Το empid είναι μοναδικό και όχι null.

Παράδειγμα:
1) δημιουργήστε ένα node key (single και composite)
2) κάντε drop το node key.

1) create constraint on (e:employee) assert (e.empid) is node key
create(e:employee{empid:1}) ***; single***
create constraint on (e:employee) assert (e.empid,e.phone) is node key ***;***

composite; δηλαδή δεν υπάρχει μόνο μια ιδιότητα η οποία θα μπει ως constraint και μπαίνουν στην ίδια παρένθεση.

2) drop constraint on (e:employee) assert (e.empid) is node key

3.2.10 STRING MATCHING EXPRESSIONS

Τα String Matching Expressions διακρίνονται σε 3 είδη:

- Starts With
- Ends With
- Contains

ΆΣΚΗΣΗ

Επιστρέψτε στο πρόγραμμα όλους τους employees που δουλεύουν για ένα department που το όνομα του αρχίζει με Σ.

ΛΎΣΗ

match p=(e:emp) return p ***; το πρόγραμμα θα μας επιστρέψει όλους τους κόμβους οι οποίοι έχουν label emp.***

match p=(e:emp)-[:worksfor]->(d:dept) where e.name starts with 's' return p ***; το πρόγραμμα θα μας επιστρέψει όλους τους employees που έχουν σχέση worksfor με το department τους και που η ιδιότητα name τους, δηλαδή το όνομα τους, ξεκινάει με το 'Σ'.***

match p=(e:emp)-[:worksfor]->(d:dept) where e.name ends with 'm' return p ***; το πρόγραμμα θα μας επιστρέψει όλους τους employees που έχουν σχέση worksfor με το department τους και που η ιδιότητα name τους, δηλαδή το όνομα τους, τελειώνει με το 'Μ'.***

match p=(e:emp)-[:worksfor]->(d:dept) where e.name contains 'sam' return p ***; το πρόγραμμα θα μας επιστρέψει όλους τους employees που έχουν σχέση worksfor με το department τους και που η ιδιότητα name τους, δηλαδή το όνομα τους, περιέχει το 'sam'***

3.2.11 CASE EXPRESSION

Χρησιμοποιείται για την σύγκριση μιας έκφρασης με πολλαπλές τιμές.

Η σύνταξή της CASE ακολουθεί την παρακάτω μορφή:

```
CASE test
        WHEN value  THEN result
        [ WHEN. . . ]
        [ ELSE default ]
END
```

ΆΣΚΗΣΗ

Εάν το όνομα είναι Sam, το πρόγραμμα να επιστρέψει ‘Hi Sam’. Εάν το όνομα είναι Thomas, τότε το πρόγραμμα να επιστρέψει ‘Hi Thomas’. Εάν το όνομα είναι οτιδήποτε άλλο, τότε το πρόγραμμα να επιστρέψει ‘Hello’.

ΛΎΣΗ

```
match(e:emp)
return
case e.name
        when 'sam'
                then 'hi sam'
        when 'thomas'
                then 'hi thomas'
        else 'hello'
end
```

match(e:emp) return case e.name when 'sam' then 'hi sam' when 'thomas' then 'hi thomas' else 'bye' end ***; το παραπάνω σχεδιάγραμμα έχει αυτήν την μορφή κώδικα στο neo4j.***

3.2.12 FOREACH CLAUSE

Το FOREACH είναι το αντίστοιχο ‘for’ σε γλώσσες όπως Jana και Python. Η FOREACH χρησιμοποιείται για να ενημερώνονται τα δεδομένα όπως η εκτέλεση εντολών σε στοιχεία σε μια διαδρομή ή σε μια λίστα. Ότι έχουμε μέσα στην FOREACH, δεν θα μπορεί να χρησιμοποιηθεί από το έξω περιβάλλον: δηλαδή αν έχουμε μια εντολή CREATE ή MERGE μέσα στην FOREACH δεν θα μπορούμε να την χρησιμοποιήσουμε εκτός της FOREACH. Η FOREACH μπορεί να χρησιμοποιήσει διάφορες εντολές όπως: CREATE (δημιουργία), MERGE (συγκέντρωση) , DELETE (διαγραφή), SET (θέτω), REMOVE (αφαίρεση) και την ίδια την FOREACH.

Παράδειγμα:

match p=(e:emp) return nodes(p) ***; το πρόγραμμα μας επιστρέφει όλους τους κόμβους που έχουμε θέσει ως p, που σε αυτήν την περίπτωση είναι, όπως βλέπουμε, όλοι οι employees μας.***

match p=(e:emp) foreach(employee in nodes(p) | set e.joined='today') ***; για κάθε employee, το πρόγραμμα θα πάρει και θα τους βάλει στο p, και θα τους προσθέσει μια έξτρα ιδιότητα εκτός από τις άλλες που έχουν. Αυτή θα είναι το joined και θα έχει τιμή today.***

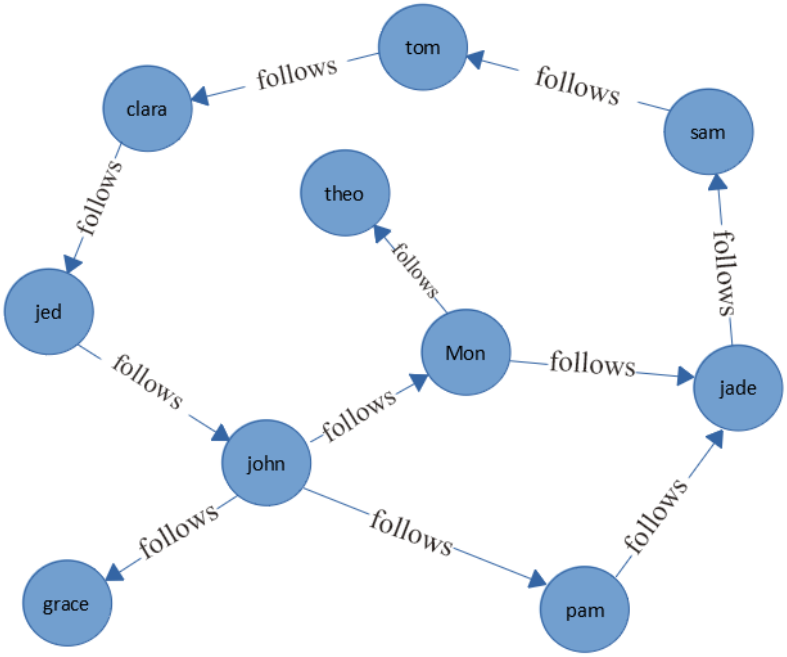
match(e:emp) return e

match p=(e:emp) foreach(employee in nodes(p) | remove e.joined) ***; ομοίως, το πρόγραμμα θα κάνει ακριβώς την ίδια διαδικασία, όμως αντί να προσθέσει την ιδιότητα joined, θα την αφαιρέσει από όλους τους κόμβους.***

3.2.13 VARIABLE LENGTH RELATIONSHIPS

Οι κόμβοι που αποτελούν μεταβλητό αριθμό σχέσεων -> Node hops away μπορούν να βρεθούν χρησιμοποιώντας την ακόλουθη σύνταξη: -[:TYPE*minHops..maxHops]→. Τα minHops και maxHops είναι προαιρετικά και από προεπιλογή είναι 1 και άπειρο αντίστοιχα. Όταν δεν δίνονται όρια, οι τελείες μπορεί να παραλειφθούν. Οι κουκκίδες μπορούν επίσης να παραλειφθούν όταν ορίζεται μόνο ένα όριο και αυτό συνεπάγεται ένα μοτίβο σταθερού μήκους

Θεωρήστε τον παρακάτω γράφο:



match(u:user) return u

match(u:user{name:'john'})-[*1]-(u1:user) return u.u1

match(u:user{name:'john'})-[:follows*1]->(u1:user) return u,u1

match(u:user{name:'john'})-[*2]-(u1:user) return u,u1

match(u:user{name:'john'})-[:follows*3]->(u1:user) return u,u1

match(u:user{name:'john'})-[*1..2]-(u1:user) return u,u1

match(u:user{name:'john'}), (u1:user{name:'jade'}), p=shortestpath((u)-[:follows*1..2]->(u1)) return p

match(u:user{name:'john'}), (u1:user{name:'jade'}), p=allshortestpaths((u)-[:follows*1..2]->(u1)) return p

3.2.14 SINGLE PROPERTY INDEX / ΕΝΙΑΊΟ ΕΥΡΕΤΗΡΊΟ ΙΔΙΟΚΤΗΣΊΑΣ

Το Cypher επιτρέπει τη δημιουργία ευρετηρίων σε μία ή περισσότερες ιδιότητες για όλους τους κόμβους που έχουν μια δεδομένη ετικέτα. Ένα ευρετήριο που δημιουργείται σε μια μεμονωμένη ιδιότητα για οποιαδήποτε δεδομένη ετικέτα ονομάζεται ευρετήριο μιας ιδιότητας. Ένα ευρετήριο που δημιουργείται σε περισσότερες από μία ιδιότητες για οποιαδήποτε δεδομένη ετικέτα ονομάζεται σύνθετο ευρετήριο.

- **ΔΗΜΙΟΥΡΓΊΑ ΕΝΙΑΊΟΥ ΕΥΡΕΤΗΡΊΟΥ ΙΔΙΟΚΤΗΣΊΑΣ**

  create index on :user(name) ***; δημιουργεί ένα index στον χρήστη και προσθέσει την ιδιότητα name.***

  call db.indexes() ***; βλέπουμε όλα τα Index που έχουμε.***

  drop index on :user(name) ***; χρησιμοποιούμε το drop για να αφαιρέσουμε ένα index***

3.2.15 COMPOSITE / ΣΎΝΘΕΤΟ ΣΤΟΙΧΕΊΟ

Ένα ευρετήριο σε πολλαπλές ιδιότητες για όλους τους κόμβους που έχουν μια συγκεκριμένη ετικέτα. π.χ. ένα σύνθετο ευρετήριο.

- **ΔΗΜΙΟΥΡΓΊΑ ΕΝΌΣ ΣΎΝΘΕΤΟΥ ΣΤΟΙΧΕΊΟΥ**

  create index on :user(name,salary) ***; συνδυασμός των ιδιοτήτων name και salary.***

  drop index on :user(name,salary) ***; αφαιρούμε ένα composite.***

Σε αντίθεση με τα ευρετήρια μεμονωμένων ιδιοτήτων, τα σύνθετα ευρετήρια υποστηρίζουν προς το παρόν μόνο έλεγχο ισότητας: n.prop — value και έλεγχος μέλους λίστας: n.prop IN list. Δεν υποστηρίζονται ερωτήματα που περιέχουν τους ακόλουθους τύπους ιδιοτήτων που βασίζονται σε ιδιότητες στο ευρετήριο:

- existence check: check(n.prop)
- range search: n.prop > value
- prefix search: STARTS WITH
- suffix search: ENDS WITH
- substring search: CONTAINS

3.2.16 STRING HANDLING FUNCTIONS IN NEO4J

return left('Johnathan',3) ***; θα επιστρέψει τους 3 αριστερούς χαρακτήρες 'Joh'***

return right('Johnathan',3) ***; θα επιστρέψει τους 3 δεξιά χαρακτήρες 'han'***

return ltrim(' Johnathan') ***; θα διαγράψει τον πρώτο χαρακτήρα από αριστερά (σε αυτήν την περίπτωση το κενό). Άρα θα γίνει 'Johnathan'***

return rtrim(' Johnathan ') ***; θα διαγράψει τον πρώτο χαρακτήρα από δεξιά (σε αυτήν την περίπτωση το κενό). Άρα θα γίνει ' Johnathan'***

return trim(' Johnathan ') ***; θα διαγράψει έναν χαρακτήρα και από τις δύο πλευρές, αριστερά και δεξιά. Άρα θα γίνει 'Johnathan'.***

return replace('Johnathan','n','r') ***; βρίσκει μέσα στην λέξη τον χαρακτήρα 'n' και τον αντικαθιστά με τον χαρακτήρα 'r'. Άρα θα γίνει 'Johrathar'***

return reverse('Johnathan') ***; αντιστρέφει όλους τους χαρακτήρες. Άρα θα γίνει nahtanhoJ***

return split(' Johnathan mvp',' ') ***; χωρίζει τις λέξεις με κόμμα, όταν μεσολαβεί κάποιο κενό. Άρα θα γίνει Johnathan,myp.***

return substring('Johnathan',0,2) ***; θα πάρει τους 3 πρώτους χαρακτήρες (ξεκινάμε να μετράμε από το 0). Άρα θα γίνει Joh.***

return tostring(123) ***; μετατρέπει έναν αριθμό σε String.***

ToLower() ***; μετατρέπει όλους τους χαρακτήρες σε πεζούς***

ToUpper() ***; μετατρέπει όλους τους χαρακτήρες σε κεφαλαία***

3.2.17 ΟΙ ΕΝΤΟΛΈΣ CALL ΚΑΙ YIELD

Οι διαδικασίες καλούνται χρησιμοποιώντας την ρήτρα CALL. Η δευτερεύουσα πρόταση YIELD χρησιμοποιείται για να επιλέξει ρητά σε ποια από τα διαθέσιμα πεδία αποτελέσματος επιστρέφονται μεταβλητές πρόσφατα δεσμευμένες από την κλήση διαδικασίας στον χρήστη ή για περαιτέρω επεξεργασία από το υπόλοιπο ερώτημα.

call dbms.procedures() ***; καλεί τα procedures***

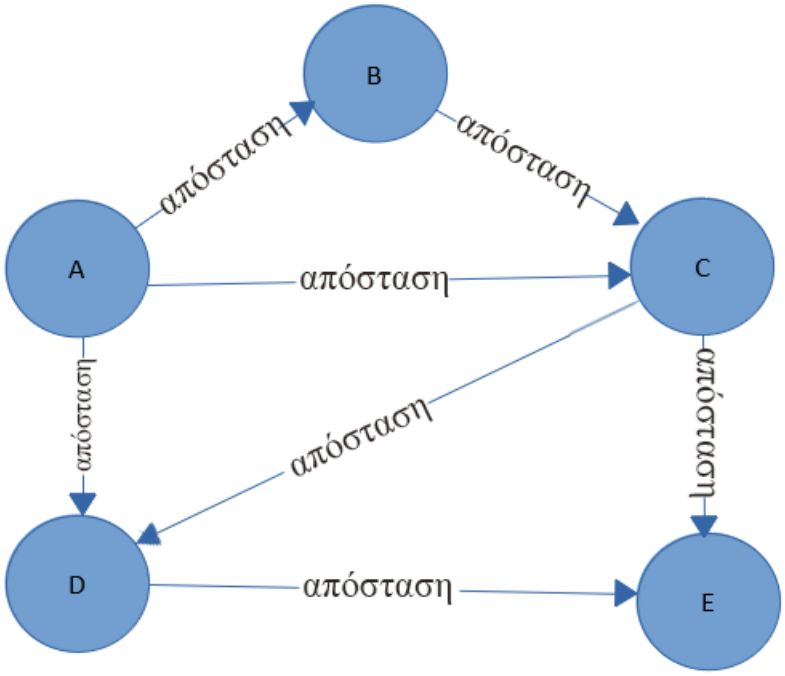
call dbms.procedures() yield name,signature ***; με την χρήση του call . . . yield καλούμε συγκεκριμένους παραμέτρους των procedures***

3.2.18 Η ΣΥΝΆΡΤΗΣΗ GET NODE BY ID

Καλούμε έναν κόμβο με βάση την ιδιότητα / αναγνωριστικό ID που έχει.

```
return algo.getNodeById(231)
return algo.getNodeById(231).name
match(l:location) return l
```

3.2.19 SHORTEST PATH ΧΡΗΣΙΜΟΠΟΙΏΝΤΑΣ HOPS



ΆΣΚΗΣΗ

Χρησιμοποιώντας τον παραπάνω γράφο, να βρείτε τη συντομότερη διαδρομή μεταξύ κόμβων χρησιμοποιώντας hops. Αγνοήστε την κατεύθυνση των βέλων.

ΛΎΣΗ

```
match(l:location{name:'D'})-[d:distance]-(l1:location{name:'C'}),p=shortestpath((l)-[:distance]-(l1)) return p
```

3.2.20 WEIGHTED SHORTEST PATH

ΆΣΚΗΣΗ

Χρησιμοποιώντας τον γράφο της προηγούμενης άσκησης καθορίστε την σταθμισμένη συντομότερη διαδρομή μεταξύ δύο κόμβων Α και Ε.

ΛΎΣΗ

```
match(l:location{name:'A'}),(l1:location{name:'E'})
call algo.shortestPath.stream(l,l1,"cost")
yield nodeId,cost
return algo.getNodeById(nodeId).name,cost
```

3.2.21 ALL PAIRS SHORTEST PATH

ΆΣΚΗΣΗ

Χρησιμοποιώντας τον προηγούμενο γράφο καθορίστε την σταθμισμένη συντομότερη διαδρομή από όλους τους κόμβους.

ΛΎΣΗ

```
WITH [
      // start, end, distance
    [ "A","B", 10 ]
    [ "A","C", 33 ]
    [ "A","D", 35 ]
    [ "C","D", 28 ]
    [ "B","C", 20 ]
    [ "C","E", 6 ]
    [ "D","E", 40 ]
] AS nested
UNWIND nested AS row
MERGE (n: City {name: row[0]})
MERGE (m: City {name: row[1]})
MERGE (n)-[r:ROAD {distance: row[2]}] -> (m)


call algo.allShortestPaths.stream("cost")
```

yield sourceNodeId,targetNodeId,distance

where sourceNodeId<targetNodeId

return

algo.getNodeById(sourceNodeId).name,

algo.getNodeById(targetNodeId).name,distance

order by distance

3.2.22 MINIMUM SPANNING TREE / MST QUERY

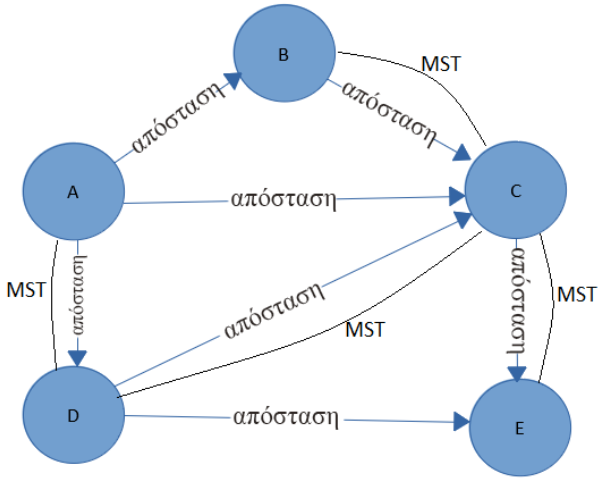
ΆΣΚΗΣΗ

Για να προσδιορίσουμε την καλύτερη διαδρομή για να επισκεφθούμε όλους τους κόμβους, το δέντρο δεν θα πρέπει να περιέχει κύκλους. Καθορίστε το Minimum Spanning Tree για το δοσμένο μονοπάτι. Αρχίστε με τον κόμβο Α.

ΛΎΣΗ

```
match(l:location{name:'A'})
call algo.spanningTree.minimum("location","distance","cost",id(l))
yield loadMillis,computeMillis,effectiveNodeCount
return loadMillis,computeMillis,effectiveNodeCount
match(n) return n
```

NAME	TYPE	DESCRIPTION
effectiveNodeCount	int	The number of visited nodes
loadMillis	int	Milliseconds for loading data
computeMillis	int	Milliseconds for running the algorithm
writeMillis	int	Milliseconds for writing result data back



3.2.23 ΕΙΣΑΓΩΓΉ ΔΕΔΟΜΈΝΩΝ ΑΠΌ CSV ΑΡΧΕΊΟ

Το CSV είναι ένα αρχείο που χωρίζεται με κόμματα, το οποίο συναντάμε στο EXCEL ή σε παρεμφερές εργαλεία. Ουσιαστικά, το CSV είναι μια διαδεδομένη μορφή αρχείων με φιλικό περιβάλλον προς τον χρήστη και είναι ήδη ενσωματωμένο σε συστήματα και στην καθημερινότητά μας. Ανάλογα με τις ανάγκες μας και το μέγεθος του αρχείου, το NEO4J διαθέτει τους ακόλουθους τρόπους ενσωμάτωσης ενός αρχείου CSV:

Η LOAD CSV είναι απλή στην χρήση και χρησιμοποιείται από πολλούς χρήστες. Συνδυάζει κομμάτια όπως:

- Υποστηρίζει τη φόρτωση / απορρόφηση δεδομένων CSV από ένα URI
- Αντιστοιχίζει απευθείας τα δεδομένα εισόδου σε σύνθετη δομή γραφήματος / τομέα
- Χειρίζεται τη μετατροπή δεδομένων
- Υποστηρίζει πολύπλοκους υπολογισμούς
- Δημιουργεί ή συγχωνεύει οντότητες, σχέσεις και δομή

Το αρχείο CSV για χρήση LOAD CSV πρέπει να έχει τα ακόλουθα χαρακτηριστικά:

- η κωδικοποίηση χαρακτήρων είναι UTF-8.
- ο τερματισμός της τελικής γραμμής εξαρτάται από το σύστημα, π.χ. είναι \nστο unix ή \r\n στα windows.
- ο προεπιλεγμένος τερματιστής πεδίου είναι , ;
- Ο χαρακτήρας τερματισμού πεδίου μπορεί να αλλάξει χρησιμοποιώντας την επιλογή FIELDTERMINATOR που είναι διαθέσιμη στην LOAD CSV εντολή.

- Οι συμβολοσειρές σε εισαγωγικά επιτρέπονται στο αρχείο CSV και τα εισαγωγικά απορρίπτονται κατά την ανάγνωση των δεδομένων.
- ο χαρακτήρας για την εισαγωγική συμβολοσειράς είναι διπλό εισαγωγικό ".
- εάν dbms.import.csv.legacy_quote_escaping έχει οριστεί στην προεπιλεγμένη τιμή του true, χρησιμοποιείται ως χαρακτήρας διαφυγής.
- ένα διπλό εισαγωγικό πρέπει να είναι σε μια συμβολοσειρά σε εισαγωγικά και να έχει διαφύγει, είτε με τον χαρακτήρα διαφυγής είτε με ένα δεύτερο διπλό εισαγωγικό.

Εισαγωγή δεδομένων από ένα αρχείο CSV

Για να εισαγάγετε δεδομένα από ένα αρχείο CSV στο Neo4j, μπορείτε να τα χρησιμοποιήσετε LOAD CSV για να λάβετε τα δεδομένα στο ερώτημά σας. Στη συνέχεια, το γράφετε στη βάση δεδομένων σας χρησιμοποιώντας τις κανονικές ρήτρες ενημέρωσης του Cypher.

Η διεύθυνση URL του αρχείου CSV καθορίζεται χρησιμοποιώντας FROM ακολουθούμενη από μια αυθαίρετη έκφραση που αξιολογεί την εν λόγω διεύθυνση URL.

Απαιτείται να καθορίσετε μια μεταβλητή για τα δεδομένα CSV χρησιμοποιώντας AS.

Τα αρχεία CSV μπορούν να αποθηκευτούν στον διακομιστή βάσης δεδομένων και στη συνέχεια είναι προσβάσιμα χρησιμοποιώντας μια file:///διεύθυνση URL. Εναλλακτικά, LOAD CSVυποστηρίζει επίσης την πρόσβαση σε αρχεία CSV μέσω HTTPS , HTTP και FTP .

LOAD CSVυποστηρίζει πόρους συμπιεσμένους με gzip και Deflate . Επιπλέον LOAD CSVυποστηρίζει τοπικά αποθηκευμένα αρχεία CSV συμπιεσμένα με ZIP .

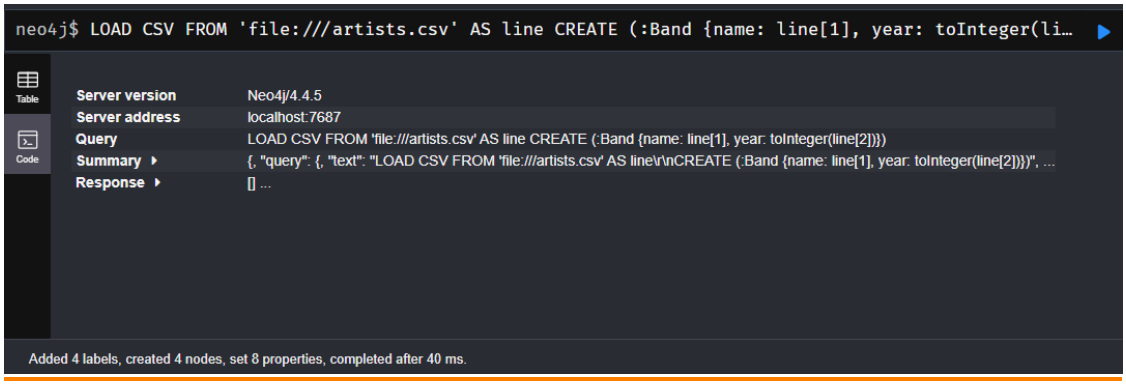
LOAD CSVθα ακολουθήσει ανακατευθύνσεις HTTP , αλλά για λόγους ασφαλείας δεν θα ακολουθήσει ανακατευθύνσεις που αλλάζουν το πρωτόκολλο, για παράδειγμα εάν η ανακατεύθυνση γίνεται από HTTPS σε HTTP .

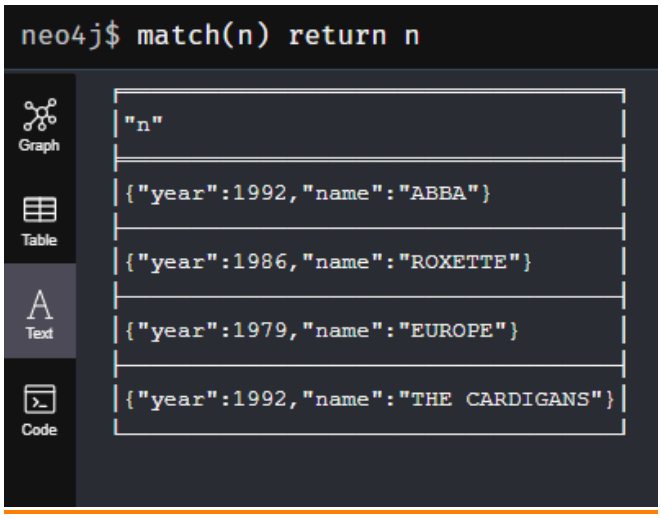
LOAD CSV χρησιμοποιείται συχνά σε συνδυασμό με την υπόδειξη ερωτήματος PERIODIC COMMIT.

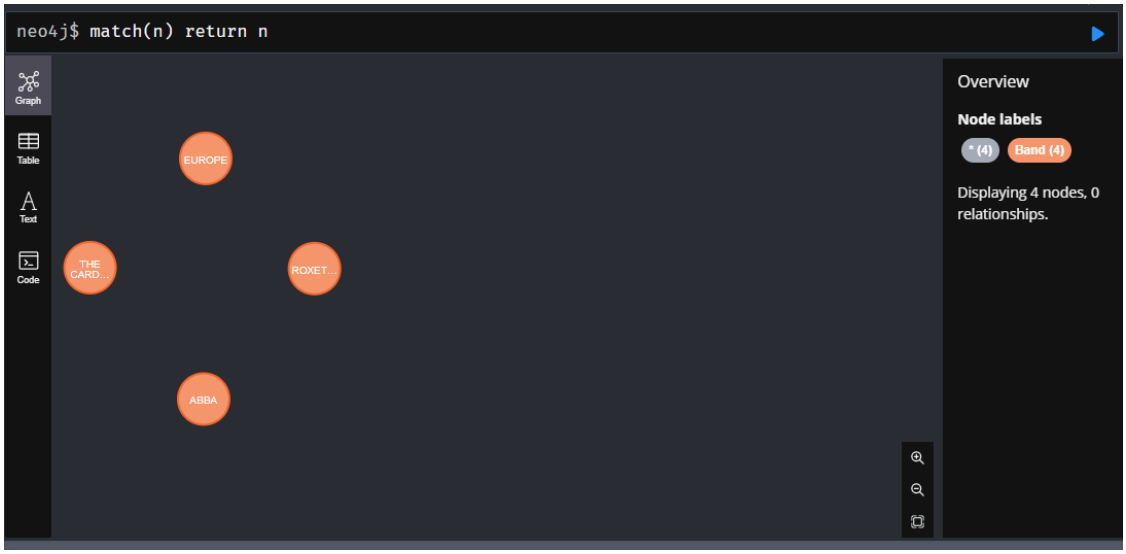
bandes.csv
1 , Led Zeppelin, 1968 2 , Scorpions, 1965 3 , Muse, 1994 4 , The Cranberries, 1990

LOAD CSV FROM 'file:///bandes.csv' AS line
CREATE (:Band {name: line[1], year: toInteger(line[2])})

Ένας νέος κόμβος με την Band ετικέτα δημιουργείται για κάθε σειρά στο αρχείο CSV. Επιπλέον, δύο στήλες από το αρχείο CSV ορίζονται ως ιδιότητες στους κόμβους.





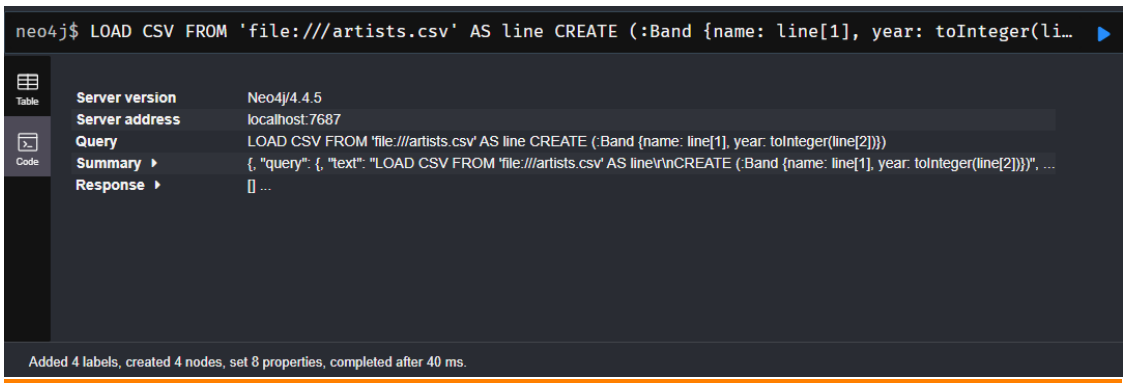


Εισαγάγετε δεδομένα από ένα απομακρυσμένο αρχείο CSV

Αντίστοιχα, μπορείτε να εισάγετε δεδομένα από ένα αρχείο CSV σε απομακρυσμένη τοποθεσία στο Neo4j. Λάβετε υπόψη ότι αυτό ισχύει για όλες τις παραλλαγές αρχείων CSV.

data.neo4j.com/bands/bandes.csv
1 , Led Zeppelin, 1968 2 , Scorpions, 1965 3 , Muse, 1994 4 , The Cranberries, 1990

```
LOAD CSV FROM 'https://data.neo4j.com/bands/bandes.csv' AS line
CREATE (:Band {name: line[1], year: toInteger(line[2])})
```

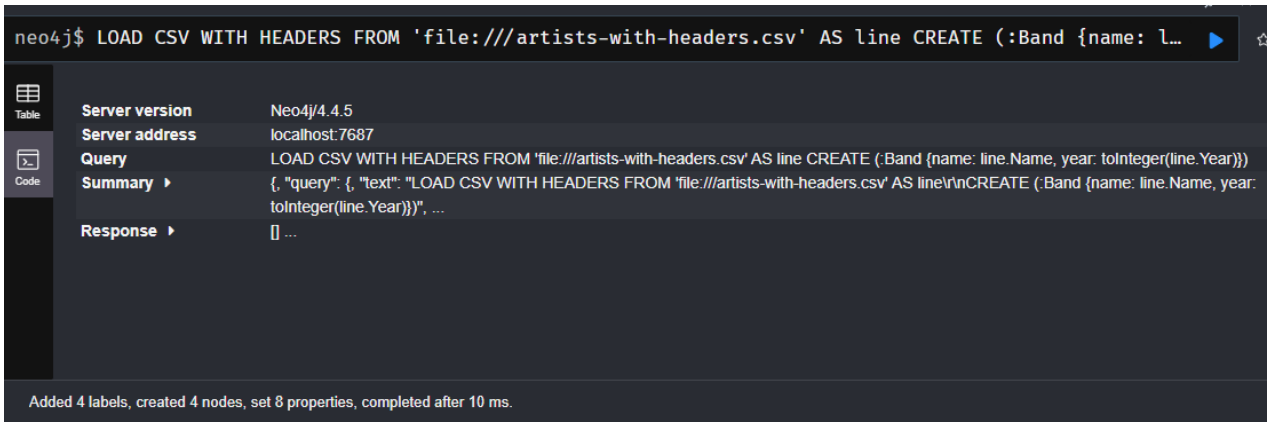


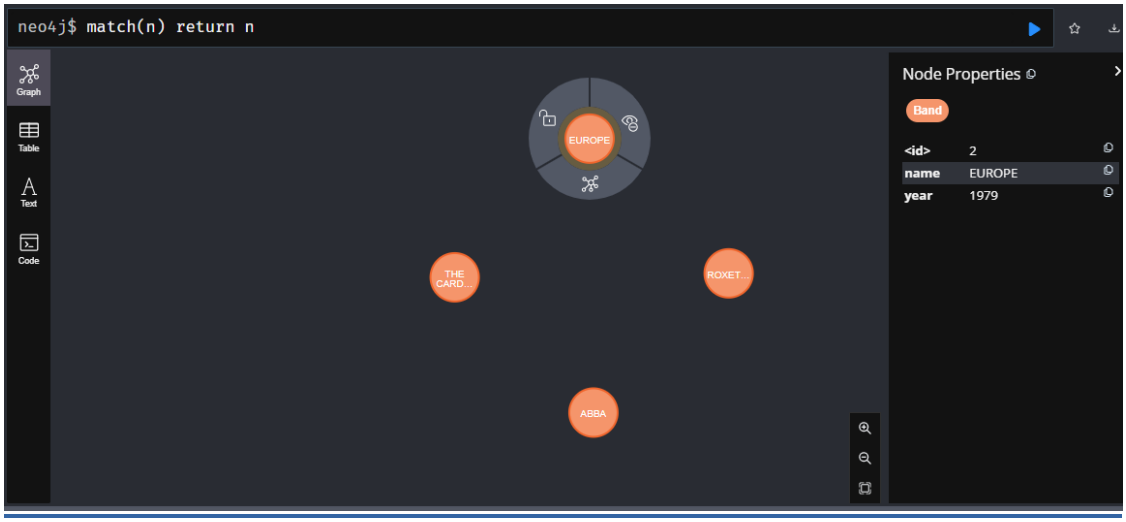
Εισαγωγή δεδομένων από ένα αρχείο CSV που περιέχει κεφαλίδες

Όταν το αρχείο CSV έχει κεφαλίδες, μπορείτε να προβάλετε κάθε σειρά του αρχείου ως χάρτη αντί ως μια σειρά από συμβολοσειρές.

bandes-with-headers.csv
1 , Led Zeppelin, 1968 2 , Scorpions, 1965 3 , Muse, 1994 4 , The Cranberries, 1990

```
LOAD CSV WITH HEADERS FROM 'file:///bandes-with-headers.csv' AS line
CREATE (:Band {name: line.Name, year: toInteger(line.Year)})
```


neo4j$ LOAD CSV WITH HEADERS FROM 'file:///artists-with-headers.csv' AS line CREATE (:Band {name: l…
Table
Code
Server version
Neo4j/4.4.5
Server address
localhost:7687
Query
LOAD CSV WITH HEADERS FROM 'file:///artists-with-headers.csv' AS line CREATE (:Band {name: line.Name, year: toInteger(line.Year)})
Summary
{, "query": {, "text": "LOAD CSV WITH HEADERS FROM 'file:///artists-with-headers.csv' AS line\r\nCREATE (:Band {name: line.Name, year: toInteger(line.Year)})", ...
Response
[] ...
Added 4 labels, created 4 nodes, set 8 properties, completed after 10 ms.


neo4j$ match(n) return n
Graph
Table
Text
Code
EUROPE
THE CARD...
ROXET...
ABBA
Node Properties
Band
<id> 2
name EUROPE
year 1979

ΚΕΦΑΛΑΙΟ 4

4.1 ΥΛΟΠΟΊΗΣΗ ΕΦΑΡΜΟΓΉΣ

Η βασική μας ιδέα είναι η δημιουργία μίας βάσης δεδομένων ενός αεροδρομίου όπου θα περιέχει πτήσεις από έναν προορισμό σε κάποιον άλλο. Κάθε πτήση θα έχει μερικές μοναδικές ιδιότητες όπως Flight_Num, Flight_Name, Flight_Date και Flight_Time οι οποίες μπορούν να αναζητηθούν μεμονωμένα με βάση τα προηγούμενα παραδείγματα. Ο χρήστης θα μπορεί να ενημερώνεται για τις πτήσεις, το πρόγραμμά τους και τους προορισμούς.

Οι τρόποι με τους οποίους μπορεί να συνταχθεί κώδικας είναι δύο, είτε προσθέτοντας χειροκίνητα τους κόμβους με τις ιδιότητές τους και τις σχέσεις τους μία-μία εντολή, είτε από ένα αρχείο csv όπως το παρακάτω:

A1 | Origin

	A	B	C	D	E	F
1	Origin	Destination	Flight_No	Flight_Name	Flight_Date	Flight_Time
2	Athens	Eindhoven	6542	AEI6542	19022022	9:30
3	Athens	Alexandria	4189	AAL4189	17112022	10:00
4	Amsterdam	Addis Ababa	5458	AAD5458	19022022	8:15
5	Hanover	Valencia	4378	HVA4378	25092022	17:20
6	Brussels	Amman	1852	BAM1852	18122022	21:00
7	Vienna	Athens	2692	VAT2692	30072022	6:00
8	Warsaw	Beirut	8178	WBE8178	14092022	13:35
9	Vilnius	Zurich	9687	VZU9687	15062022	23:10
10	Amman	Athens	6817	AAT6817	31032022	21:00
11	Addis Ababa	Amsterdam	4865	AAM4865	19082022	7:15
12	Eindhoven	Germany	6871	EGE6871	16042022	7:45
13	Valencia	Amsterdam	3581	VAM3581	12042022	2:20
14	Beirut	Hanover	1084	BHA1084	11112022	1:35
15	Alexandria	Germany	3285	AGE3285	29012022	8:00

Α) Εισαγωγή πληροφοριών μέσω csv

```
LOAD CSV WITH HEADERS
FROM 'file:///flights.csv' AS line
MERGE (Origin:Airport{code:line.Origin})
MERGE (Destination:Airport{code:line.Destination})
MERGE (Origin)-[:FLY_TO]->(Destination)
```

```
1 LOAD CSV WITH HEADERS
2 FROM 'file:///flights.csv' AS line
3 MERGE (Origin:Airport{code:line.Origin})
4 MERGE (Destination:Airport{code:line.Destination})
5 MERGE (Origin)-[:FLY_TO]→(Destination)
```

Table

Code

Added 15 labels, created 15 nodes, set 15 properties, created 14 relationships, completed after 207 ms.

Β) Εισαγωγή πληροφοριών χωρίς εξωτερικών αρχείων

create(o:Origin{name:'AMMAN',Flight_No:6817,Flight_Name:'AAT6817',Flight_Date:31032022,Flight_Time:'21:00'})-[w:FLY_TO]->(d:Destination{name:'ATHENS'})<-[w1:FLY_TO]-(o1:Origin{name:'ALEXANDRIA',Flight_No:4189,Flight_Name:'AAL4189',Flight_Date:17112022,Flight_Time:'10:00'}) return o,w,d,w1,o1

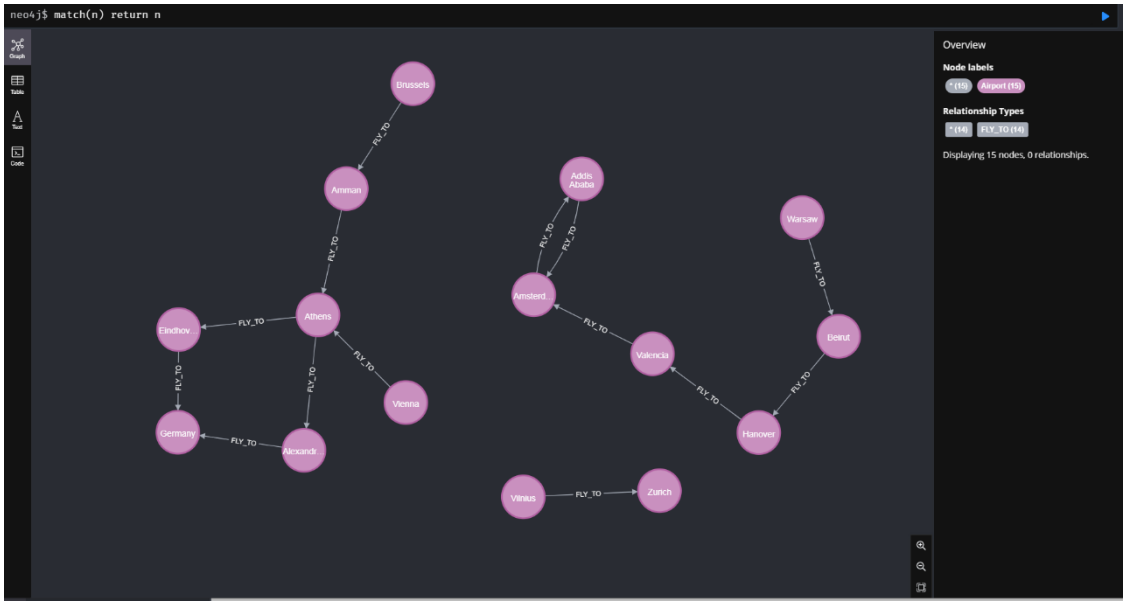
create(o:Origin{name:'BEIRUT',Flight_No:1084,Flight_Name:'BHA1084',Flight_Date:11112022,Flight_Time:'1:35'})-[w:FLY_TO]->(d:Destination{name:'HANOVER'})<-[w1:FLY_TO]-(o1:Origin{name:'VALENCIA',Flight_No:4378,Flight_Name:'HVA4378',Flight_Date:25092022,Flight_Time:'17:20'}) return o,w,d,w1,o1

.

.

.

Match(n) return n ;για εμφάνιση συνολικού γραφήματος



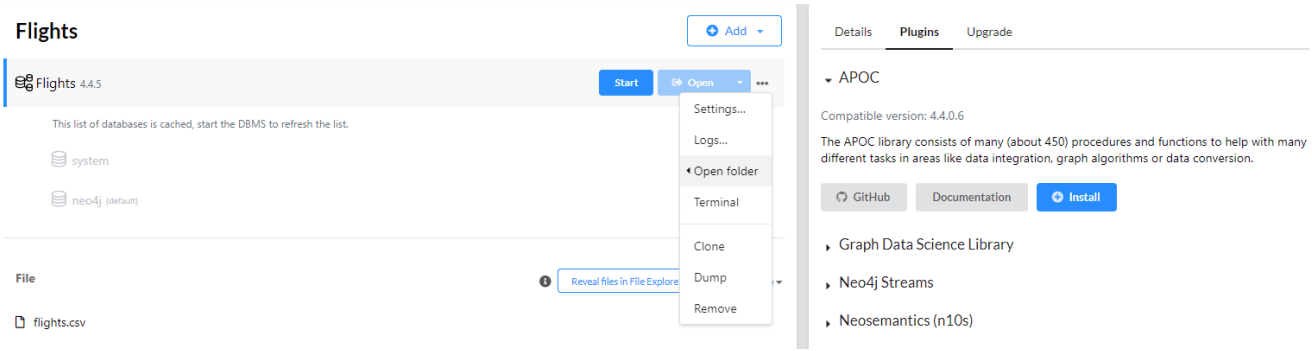
4.2 ΠΡΟΒΛΗΜΑΤΑ ΠΟΥ ΠΡΟΕΚΥΨΑΝ Q&A

Παρακάτω ακολουθεί μία λίστα από μικρο-προβλήματα που συναντήσαμε μαζί με τους τρόπους επίλυσής τους:

- Κατά τη συγγραφή του κώδικα θέλαμε να βρούμε έναν γρήγορο και αξιόπιστο τρόπο να δημιουργηθούν σχέσεις μεταξύ των κόμβων μας οπότε και χρησιμοποιήσαμε την εξής εντολή: **“apoc.create.relationship”**
  Παρόλα αυτά μας εμφάνιζε το σφάλμα: **“there's no procedure with the name 'apoc.create.relationship' registered for this data”**

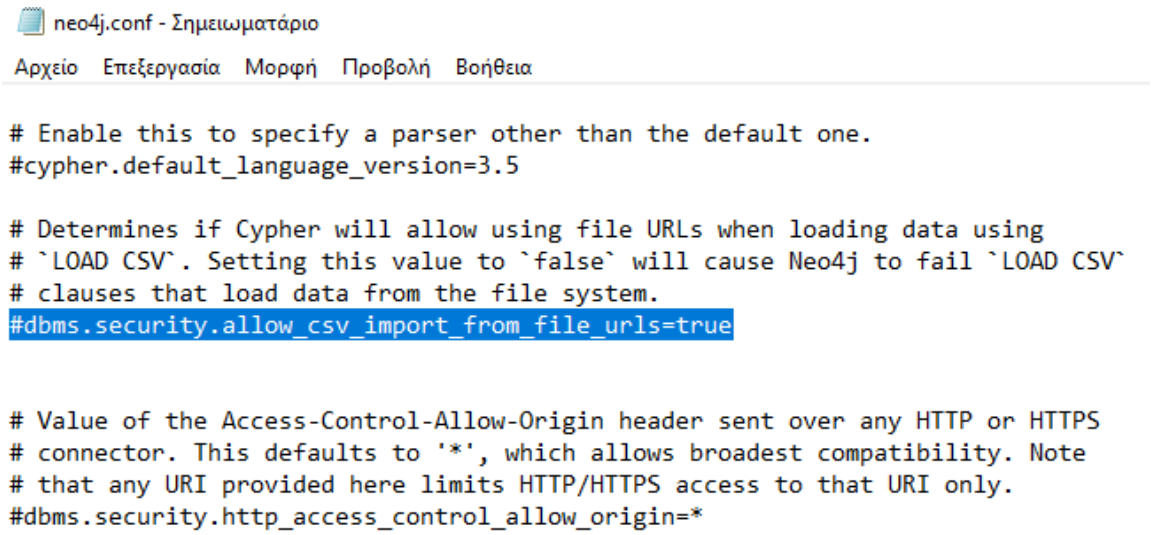
  Επίλυση:
  Χρειάστηκε να προσθέσουμε χειροκίνητα αυτήν τη γραμμή στο αρχείο **.neo4j.conf**:
  dbms.directories.plugins=/Applications/Neo4j\Community\Edition.app/Contents/Resources/app/plugins ,υποθέτοντας ότι εκεί βάλαμε το APOC jar executable και στη συνέχεια επανεκκινούμε τον διακομιστή.

  Εναλλακτικά, πηγαίνουμε στο ήδη υπάρχον DBMS (στο παράθυρο της εφαρμογής) πχ. Flights , κάνουμε κλικ πάνω του και δεξιά εμφανίζεται ένα πάνελ με τις επιλογές Details, Plugins, Upgrade. Επιλέγουμε το Plugins και κάνουμε install το APOC_library.



- Στην συγκεκριμένη περίπτωση θέλαμε να εισάγουμε δεδομένα από ένα αρχείο .csv ,αλλά δεν αναγνώριζε το path του αρχείου που είχαμε δηλώσει.
  Εμφάνιζε σφάλμα τύπου: **Couldn't load the external resource at: file:/home/user/Documents/neo4j-community 4.0.1/import/home/user/Documents/flights.csv**

  Επίλυση:
  Αυτή είναι μια ασφάλεια που έχει ενσωματωθεί στο neo4j 3.0 προκειμένου να αποτραπεί η φόρτωση scripts πηγών από έναν ανεπιθύμητο κατάλογο (όπως το /etc/password για παράδειγμα.
  Οπότε, πηγαίνουμε και προσθέτουμε στο αρχείο conf/neo4j.conf την εξής εντολή για να το παρακάμψουμε:
  dbms.security.allow_csv_import_from_file_urls=true



  Εναλλακτικά κάνουμε import file από το παράθυρο πλοήγησης και στη συνέχεια παίρνουμε το αρχείο και το αντιγράφουμε στο path με το αντίστοιχο project μας (πχ. C:\Users\User\.Neo4jDesktop**relate-data\dbmss\dbms-36b792fa-6bc6-47bf-8be2-f52d362e619a\import**\flights.csv).

❖ Ένα τελευταίο πρόβλημα που αντιμετωπίσαμε ήταν κατά τη δημιουργία κόμβων, όπου στην τελική εικόνα εμφανίζονταν διπλή και τριπλή φορά, έτσι ώστε να συνδυαστούν τα διάφορα δεδομένα και για να γίνει χρήση όλων αυτών αλλά και των μεταξύ τους σχέσεων.

Επίλυση:

Από τη στιγμή που έχουμε έναν κοινό κόμβο και απλά έχει διαφορετικές σχέσεις μπορούμε να κάνουμε τα εξής:

1) Βρίσκουμε όλους τους εξτρά κόμβους και τροποποιούμε τις παρακάτω γραμμές με βάση τις ανάγκες μας (ονομασίες, σχέσεις και λοιπά).

```
MATCH (n:Tag)
WITH n.name AS name, COLLECT(n) AS nodelist, COUNT(*) AS count
WHERE count > 1
RETURN name, nodelist, count
```

2) Αντιγράφουμε όλες τις σχέσεις από τους διπλότυπους κόμβους στον πρώτο.

3) Διαγράφουμε όλους τους αχρείαστους κόμβους.

ΒΙΒΛΙΟΓΡΑΦΊΑ

- https://www.oracle.com/autonomous-database/what-is-graph-database
- https://neo4j.com/developer/guide-import-csv/
- What is a Graph Database? {Definition, Use Cases & Benefits} (phoenixnap.com)
- Fast Graph Database: Neo4j Performance Experiment
- Native vs. Non-Native Graph Database Architecture & Technology (neo4j.com)
- Graph Databases & OrientDB (slideshare.net)
- https://neo4j.com/docs/java-reference/current/transaction-management/
- https://neo4j.com/blog/native-vs-non-native-graph-technology/
- https://neo4j.com/top-ten-reasons/
- https://www.infoq.com/articles/graph-nosql-neo4j/
- https://www.packetlabs.net/posts/bloodhound-tool-active-directory-assessment/
- https://github.com/BloodHoundAD/BloodHound
- https://bloodhound.readthedocs.io/en/latest/data-collection/sharphound.html
- https://el.wikipedia.org/wiki/ACID
- https://neo4j.com/developer/graph-database/
- https://en.wikipedia.org/wiki/Neo4j
- List functions - Neo4j Cypher Manual
- Neo4j System Properties (db-engines.com)
- Five Common GraphQL Problems and How Neo4j-GraphQL Aims To Solve Them | by William Lyon | GRANDstack - GraphQL, React, Apollo, Neo4j Database
- Clauses - Neo4j Cypher Manual
- Graph database (slideshare.net)
- (3) Exploring Graph Algorithms with Neo4j: The Shortest Path Algorithm | packtpub.com - YouTube
- https://subscription.packtpub.com/book/hardware-and-creative/9781783287758/app/ch05lvl1sec32/operators
- https://graphaware.com/graphaware/2015/05/19/neo4j-cypher-variable-length-relationships-by-example.html
- https://neo4j.com/docs/cypher-manual/current/syntax/
- https://stackoverflow.com/questions/42800137/neo4j-cypher-merge-duplicate-nodes
- Install a plugin - Neo4j Desktop
- neo4j - There is no procedure with the name `apoc.help` registered for this database instance - Stack Overflow