Κατανεμημένη Εκτέλεση *SQL Queries* με χρήση *Trino*

Ανάστάσης Αγγλογάλλος
ΑΜ*: 03118641*

Γεώργιος Παπαδούλης
ΑΜ*: 03118003*

Χριστίνα Προεστάκη
ΑΜ*: 03118877*

***Abstract*—**Το *Trino*, επίσης γνωστό ως *PrestoSQL*, είναι ένα *open − source* εργαλείο για κατανεμημένη εκτέλεση *SQL* ερωτημάτων σε ογκώδη σύνολα δεδομένων. Τα δεδομένα αυτά είναι δυνατό να προέρχονται από μία ή παραπάνω πηγές οι οποίες είναι ετερογενείς, χωρίς να υπάρχει ανάγκη να μεταφέρονται σε μια κεντρική τοποθεσία. Στην παρούσα εργασία χρησιμοποιήσαμε το *Trino* ώστε να μετρήσουμε την επίδοσή του εκτελώντας ποικίλα *SQL queries* σε διάφορα δεδομένα που αποθηκευόνται σε διαφορετικές βάσεις δεδομένων.

Index Terms*—Trino*, *MongoDB*, *Cassandra*, *Redis*, *Presto*, *SQL*

GitHubLink: https://github.com/G-Papad/adts.git/

I. Εισαγωγή

Η ραγδαία αύξηση της ποσότητας των δεδομένων και η ανάγκη για αποδοτική διαχείριση τους είναι ιδιαίτερα σημαντική για πληθώρα οργανισμών. Ωστόσο η αύξηση του πλήθους των δεδομένων καθιστά την επεξεργασία τους δύσκολη, ακριβή και ιδιαίτερα αργή. Με την έντονη μετάβαση στην εποχή των *BigData* εγίνε επιτακτική η ανάγκη για την δημιουργία ενός εργαλείου που θα δίνει λύση στο πρόβλημα αυτό. Μία από τις εταιρίες που είχε αυτή την ανάγκη, η *Facebook*, ανέπτυξε το *PrestoSQL* το 2013 - το οποίο μετονομάστηκε στη συνέχεια σε *Trino* - και κατάφερε να επεξεργάζεται αποδοτικά όγκο δεδομένων τάξεως *petabytes*.

Το *Trino* είναι ένα εργαλείο που χρησιμοποεί *SQL* για την εκτέλεση ερωτημάτων (*queries*) σε βάσεις. Συγκεκριμένα είναι ένα κατανεμημένο, επεκτάσιμο, open source εργαλείο που είναι δυνατό να εκτελεί ερωτήματα σε πληθώρα βάσεων. Με τον όρο κατανεμημένο αναφερόμαστε στην ικανότητα του *Trino* να διαιρεί τα queries σε πολλά υπό-ερωτήματα και να τα εκτελεί παράλληλα σε διαφορετικά μηχανήματα. Με τον όρο επεκτάσιμο αναφερόμαστε στο ότι μπορεί εύκολα να ανταποκριθεί σε τεράστιο όγκο δεδομένων εφόσον μπορεί να "αγνοήσει" δεδομένα που πλέον δε χρειάζονται [1] [2].

Ένα από τα σημαντικότερα πλεονεκτήματα του *Trino* είναι η δυνατότητα του να συνδέσει πολλές πηγές δεδομένων, όπως αρχεία *Hadoop*, βάσεις δεδομένων *SQL*, *NoSQL*, και πολλές άλλες πηγές, χωρίς την ανάγκη να μεταφέρονται τα δεδομένα σε μια κεντρική τοποθεσία. Η δυνατότητα αυτή είναι τόσο ωφέλιμη γιατί συχνά οι οργανισμοί χρησιμοποιούν πολλες βάσεις δεδομένων για την αποθήκευση των δεδομένων τους, οι οποίες έχουν ξεχωριστούς τρόπους αποθήκευσης και πρόσβασης σε αυτά. Μέσω του *Trino* γίνεται εύκολη η εκτέλεση *queries* σε ποικίλες και ετερογενείς βάσεις δεδομένων, χρησιμοποιώντας *standardSQL*.

Στόχος αυτής της εργασίας είναι να εξοικειωθούμε με τη χρήση της *Trino*, ώστε να έχουμε τη δυνατότητα να επεξεργαζόμαστε ποικίλες βάσεις δεδομένων απλά με τη χρήση της *standardSQL*. Αυτό θα επιτευχθεί, μετρώντας την απόδοση της *Trino* με διάφορα *queries*, δεδομένα και βάσεις και καταλαβαίνοντας με αυτό τον τρόπο ποια είναι η πιο αποδοτική κατανομή των δεδομένων στις βάσεις. Οι βάσεις που χρησιμοποιήθηκαν για αυτόν το σκοπό είναι οι *MongoDB*, *Cassandra* και *Redis*.

II. Στήσιμο του Συστήματος

Η πρώτη προσπάθεια εγκατάστασης έγινε σε *VM* από τους πόρους του *okeanos − knossos* όπου χρησιμοποιήθηκε λογισμικό *linux* διανομή *Ubuntu*16.04*LTS*. Ύστερα από πολλές προσπάθειες εγκατάστασης των βάσεων δεδομένων και ρύθμιση του συστήματος διαπιστώθηκε ότι η βάση δεδομένων *CassnadraDB* δεν υποστηρίζεται στη συγκεκριμένη διανομή.

Για αυτό το λόγο, οι βάσεις δεδομένων εγκαταστήθηκαν πάνω σε *Oralce VM* με λειτουργικό σύστημα *Linux* σε τοπικό υπολογιστή. Συγκεκριμένα, χρησιμοποιήθηκε η διανομή *Ubuntu* 22.04.2 *LTS* (*Jammy Jellyfish*).

A. MongoDB

Η εγκατάσταση του *mongoDB* έγινε σύμφωνα με τις οδηγίες από το *official site*:
https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/.
Συγκεκριμένα, τα βήματα που ακολουθήθηκαν είναι τα εξής:

- Εισαγωγή του δημόσιου κλειδιού μέσω της παρακάτω εντολής το οποίο θα χρησιμοποιηθεί από το σύστημα διαχείρισης πακέτων:
  wget -qO
  - https://www.mongodb.org/static/pgp/server-6.0.asc — sudo apt-key add -
- Δημιουργία *list file* για την *MongoDB*:
  echo "deb [arch=amd64,arm64] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/6.0 multiverse" — sudo tee /etc/apt/sources.list .d/mongodb-org-6.0.list
- Επαναφόρτωση της τοπικής βάσης δεδομένων πακέτων:
  sudo apt − get update

- Εγκατάσταση των πακέτων
  sudo apt-get install -y mongodb-org=4.4.8 mongodb-org-database=4.4.8 mongodb-org-server=4.4.8 mongodb-org-mongos=4.4.8 mongodb-org-tools=4.4.8
  Εδώ αξίζει να σημειωθεί ότι η πιο πρόσφατη έκδοση του $mongoDB(6.0.4)$ δεν είναι συμβατή με τον KVM $processor$ που χρησιμοποιεί το $Oracle\ VM$ οπότε δε μπορεί να χρησιμοποιηθεί ($service\ mongod\ fail\ with\ error$: '$core - dump$').
- Για την έκδοση 4.4.8 χρειάζεται επίσης και η εγκατάση των βιβλιοθηκών $libssl1.1$ που δεν χρησιμοποιούνται στις νεώτερες εκδόσεις των $ubuntu$. sudo wget http://archive.ubuntu.com/ubuntu/pool/main/o/openssl /libssl1.1_1.1.1f-1ubuntu2_amd64.deb$sudodpkg - ilibssl1.1_1.1.1f - 1ubuntu2_amd64.deb$
- Για την ενεργοποίηση του $service\ mongodb$: $systemctl\ start\ mongod$

Η εισαγωγή δεδομένων στο $mongoDb$ γίνεται μέσω του $mongo - cli$ με την εντολή: mongoimport –db DB_Name –collection Collection_Name –type=csv –file Name-of-file-to-import –headerline

B. Cassandra

Η εγκατάσταση του $Cassandra$ έγινε σύμφωνα με τις οδηγίες από το $official\ site$ https://cassandra.apache.org/doc/latest/cassandra/ getting_ started/ installing.html

Για την εγκατάσταση του $cassandra$ είναι απαραίτητη η εγκατάσταση της $java$ και συγκεκριμένα της version $jdk = jdk - 8$:

$sudoapt - getinstall\ openjdk - 8 - jdk$

Η αλλαγή της έκδοσης $java$ που χρησιμοποιείται γίνεται με την εντολή: $sudoupdate - alternatives$

Εισαγωγή του δημόσιου κλειδίου μέσω της παρακάτω εντολής το οποίο θα χρησιμοποιηθεί από το σύστημα διαχείρισης πακέτων: curl https://downloads.apache.org/cassandra/KEYS — sudo apt-key add -

Δημιουργία $list\ file$:

echo "deb https://debian.cassandra.apache.org 41x main" — sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list

Επαναφόρτωση της τοπικής βάσης δεδομένων πακέτων: sudo apt-get update

Εγκατάσταση των πακέτων: sudo apt-get install cassandra

Για την εκκίνηση του server της $Cassandra$: $service\ cassandra\ start$ Πλέον το $service\ cassandra$ ακούει στο $port$ 9042

Η σύνδεση με τη βάση γίνεται μέσω του cli: $cqlsh$

Η εισαγωγή δεδομένων στο $Cassandra$ γίνεται ως εξής:

C. Redis

Η εγκατάσταση της $Redis$ έγινε σύμφωνα με τις οδηγίες από το $official\ documentation$ που βρίσκονται στον παρακάτω σύνδεσμο : https://redis.io/docs/getting-started/installation/install-redis-on-linux/

Εγκατιστούμε την πιο πρόσφατη σταθερή έκδοση του $Redis$ από το επίσημο αποθετήριο $packages.redis.io\ APT$. Για να το κάνουμε αυτό τα βήματα που ακολουθήθηκαν είναι τα εξής:

- Εγκατηστούμε την $lsb-release$: $sudo\ apt\ install\ lsb-release$
- Προσθέτουμε το αποθετήριο στο $repository\ apt$, το ενημερώνουμε και στη συνέχεια το εγκατηστούμε με τις παρακάτω εντολές:
  $curl - fsSL$ https://packages.redis.io/gpg — sudo gpg –dearmor -o /usr/share/keyrings/redis-archive-keyring.gpg echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://packages.redis.io/deb (lsb_release -cs) main" — sudo tee /etc/apt/sources.list.d/redis.list
  $sudo\ apt - get\ update\ sudo\ apt - get\ install\ redis$

Για να ξεκινήσουμε το server της βάσης πρέπει να χρησιμοποιήσουμε την εντολή : $redis - server$

Έπειτα ο $server$ της $Redis$ Θα μας ακούει στο $port$ $localhost$: 6379

Η επεξεργασία δεδομένων στη $Redis$ γίνεται μέσω του $Redis\ CLI$, το οποίο στην καινούργια έκδοση συμπεριλαμβάνετε στο πακέτο εγκατάστασης της $Redis$. Για την εκκίνησή του χρησιμοποιήθηκε η εντολή :

$redis - cli$

Έπειτα για την εισαγωγή δεδομένων αρχείων τύπου $.csv$ ή $.json$ στην βάση χρησιμποποιήθηκε η εντολή :

cat file-name.csv — awk -F "," '{print $1" "$2" "$3" "$4}' — xargs -n4 sh -c 'redis-cli -p 6370 set 1"2,3,4"' sh

D. Trino

Η εγκατάσταση του $Trino$ είναι η πιο σύνθετη από τις παραπάνω καθώς η επίσημη ιστοσελίδα δεν έχει αρκετές και σαφής πληροφορίες για τα βήματά της. Όμως, υπάρχει ένα άρθρο στο $Medium$ και συγκεκριμένα στο σύνδεσμο https://trino.io/download.html : https://sonusingh-javatech.medium.com/trino-sql-installation-on-ubuntu-a5d563fca22b Το οποίο περιγράφει τα εξής βήματα :

- Αρχικά πρέπει να κατεβάσουμε το $Trino\ Server$ από την επίσημη ιστοσελίδα της $Trino$ στο σύνδεσμο : https://trino.io/download.html Συγκεκριμένα το αρχείο : $trino - server - 407.tar.gz$. Για να το κάνουμε αυτό φτιάχνουμε ένα νέο φάκελο και τον ονομάζουμε $trino$ και έπειτα κατεβάζουμε το αρχείο μέσω της $wget$: path - /home/ : mkdir trino path - /home/trino/ : wget 'https://repo1.maven.org/maven2/io/trino/trino-server/407/trino-server-407.tar.gz'
- Κάνουμε $UN - TAR$ το αρχείο με την εντολή : path - /home/trino/ : tar xzvf trino-server-407.tar.gz
- Αλλάζουμε το όνομα του $trino - server - 407'$ σε '$trino - server'$: path - /home/trino/ : mv 'trino-server-407' 'trino-server'
- Δημιουργούμε έναν $datadirectory$ εκτός του $installation\ directory$, ο οποίος θα χρησιμοποιείται για την αποθήκευση αρχείων καταγραφής, μεταδεδομένων κ.λπ., ώστε να διατηρούνται εύκολα κατά την αναβάθμιση του $Trino$: path - /home/trino/ : mkdir trino-data

- Φτιάχνουμε έναν φάκελο τον οποίο ονομάζουμε "etc" και τον τοποθετούμε στο $/trino - server/$: path - /home/trino/trino-server : mkdir etc

Σε αυτό το σημείο πρέπει να αλλάξουμε το σύνδεσμο από τον οποίο συμβουλευόμαστε τα βήματα εγκατάστασης και να επισκευτούμε το documentation του $Trino$ στο σύνδεσμο : https://trino.io/docs/current/installation/deployment.html
Συνεχίζουμε τα βήματα ρύθμισης του συστήματος με τον εξής τρόπο :

- Πρόσθετουμε ένα αρχείο $node.properties$ στο φάκελο etc και χρησιμοποιούμε το κείμενο που δίνεται στο documentation
- Στο ίδιο directory προσθέτουμε με αντίστοιχο τρόπο το αρχείο "jvm.config" με το κείμενο που υπάρχει
- Αντίστοιχα προσθέτουμε το αρχείο config.properties με το κώδικα που δίνεται αυξάνοντας λίγο τα configurations
- Τέλος προσθέτουμε το αρχείο log.properties στο οποίο θα συγκρατήσουμε το ιστορικό του Trino και των Queris που ρωτάμε

Σε αυτό το στάδιο μπορούμε να ξεκινήσουμε τον Trino server χρησιμοποιώντας την παρακάτω εντολή :
path - /home/trino/trino-server/bin/ : ./launcher start
Έπειτα για να δούμε το γραφιστικό περιβάλλον της Trino δηλαδή το Trino UI επισκευόμαστε το σύνδεσμο URL — http://hostname:7070/ui/
Σύνδεση του Trino με τις βάσεις δεδομένων που διαχειρίζεται
Για να συνδέσουμε το Trino με τις υπόλοιπες βάσεις που διαχειρίζεται φτιάχνουμε ένα νέο αρχείο το οποίο ονομάζουμε catalog και το τοποθετούμε στο directory trino-server/etc/catalog. Έπειτα για κάθε βάση φτιάχνουμε ένα αρχείο db-name.properties στο οποίο βάζουμε τα απαραίτητα δεδομένα για την σύνδεση της κάθε βάση με το Trino

Για την σύνδεση της Cassandra με το Trino φτιάχνουμε το αρχείο cassandra.properties και μέσα στο αρχείο προσθέτουμε το παρακάτω κώδικα :
connector.name=cassandra
cassandra.contact-points=host1,host2
cassandra.load-policy.dc-aware.local-dc=datacenter1

Για την σύνδεση της Redis με το Trino φτιάχνουμε το αρχείο redis.properties και μέσα στο αρχείο προσθέτουμε το παρακάτω κώδικα :
connector.name=redis
redis.table-names=schema1.table1,schema1.table2
redis.nodes=host:port
Χρησιμοποιούμε το default port της Redis άρα στο πεδίο port γράφουμε :6379

Για την σύνδεση το Mongodb με το Trino φτιάχνουμε το αρχείο mongodb.properties και μέσα στο αρχείο προσθέτουμε το παρακάτω κώδικα :
connector.name=mongodb
mongodb.connection-url=mongodb://user:pass@sample.host:27017/
Δεν θα χρησιμοποιήσουμε User και Password οπότε δεν συμπληρώνουμε τα αντίστοιχα πεδία

Τέλος για να μπορούμε να γράφουμε queries στο Trino-server θα εγκαταστήσουμε το Trino CLI το οποίο θα κατεβάσουμε απο τήν επίσημη ιστοσελίδα της Trino :
https://trino.io/download.html

Κατεβάζουμε το trino-cli-408-executable.jar, το μετονομάσου σε trino και το μετατρέπουμε σε executable αρχείο μέσω της εντολής :
chmod +x

Για να τρέξουμε το CLI χρησιμοποιούμε την παρακάτω εντολή :

./trino –server http://trino.example.com:8080

Σε αυτό το σημείο είμαστε έτοιμοι να υποβάλουμε queries στο σύστημά μας.

III. Υποδομή & $Software$

Αξίζει να σημειωθεί ότι αρχικά προσπαθήσαμε να στήσουμε το σύστημα μας σε $Virtual\ Machine$ στο $Okeanos$ σε $Ubuntu$ 16.04. Η εγκατάσταση των βάσεων ήταν ιδιαίτερα πολύπλοκη και παρουσίαζε πολλά προβλήματα όποτε προσπαθήσαμε να δουλέψουμε με το εργαλείο $Docker$. Το $Docker$ είναι μία $software$ πλατφόρμα που επιτρέπει στους χρήστες να στήσουν, να ελέγξουν και να χρησιμοποιήσουν γρήγορα εφαρμογές. Ουσιαστικά επιτρέπει την εγκατάσταση των εφαρμογών στα περισσότερα μηχανήματα. Για κάθε εφαρμογή χρησιμοποιείται ένα $container$ το οποίο λειτουργεί σαν ένας μικρός υπολογιστής -έχει το δικό του λειτουργικό σύστημα, διεργασίες, CPU, μνήμη και πόρους δικτύου. Για κάθε επιμέρους βάση που χρησιμοποιήσαμε $-MongoDB$, $Cassandra$ και $Redis-$ και για το $Trino$ χρησιμοποιήσαμε ένα ξεχωριστό $container$. Ωστόσο κατά τη σύνδεση αυτών των containers παρουσιάστηκαν διάφορα τεχνικά προβλήματα, καθίστοντας εν τέλει τη διαδικασία χρονοβόρα. Επομένως, δοκιμάσαμε να εγκαταστήσουμε τις βάσεις και το $Trino$ σε δίκο VM σε $Ubuntu$ 22.04 με επιτυχία χωρίς τη χρήση του $Docker$.

Οι τρεις βάσεις που χρησιμοποιήσαμε είναι $NoSQL$. Οι βάσεις $NoSQL$ ($Not\ Only\ SQL$) είναι συστήματα βάσεων δεδομένων που δεν χρησιμοποιούν το παραδοσιακό σχεσιακό μοντέλο για την αποθήκευση δεδομένων. Αντ' αυτού, χρησιμοποιούν άλλους τρόπους οργάνωσης και αποθήκευσης των δεδομένων που επιτρέπουν την αντιμετώπιση προβλημάτων κλιμακωσιμότητας και διαθεσιμότητας σε μεγάλες ποσότητες δεδομένων.

Τα συστήματα $NoSQL$ διακρίνονται ανάλογα με τον τρόπο αποθήκευσης των δεδομένων τους. Οι βάσεις $MongoDB$, $Redis$ και $Cassandra$ είναι $Key-Value$ βάσεις δεδομένων.

Οι βάσεις δεδομένων αυτού του τύπου αποθηκεύουν τα δεδομένα σε μορφή κλειδί-τιμή. Τα δεδομένα αντιστοιχούν σε μια τιμή, που είναι συνήθως μια σειρά από bytes, και αναζητούνται με βάση ένα μοναδικό κλειδί.

A. *MongoDB*

Το *MongoDB* είναι ένα ανοιχτού κώδικα σύστημα διαχείρισης βάσεων δεδομένων (*DBMS*) τύπου *NoSQL* (*Not only SQL*), που χρησιμοποιείται για αποθήκευση, ανάκτηση και διαχείριση δεδομένων σε μη δομημένες ή ημι-δομημένες μορφές.

Σε αντίθεση με τις παραδοσιακές σχεσιακές βάσεις δεδομένων, όπου τα δεδομένα οργανώνονται σε πίνακες με σταθερές στήλες και γραμμές, το MongoDB αποθηκεύει τα δεδομένα σε μορφή *JSON* (*JavaScript Object Notation*) και επιτρέπει την αποθήκευση αντικειμένων με διαφορετική δομή στην ίδια συλλογή (*collection*). Αυτό καθιστά το *MongoDB* πολύ ευέλικτο και κατάλληλο για εφαρμογές που χρειάζονται αποθήκευση μεγάλου όγκου δεδομένων με ποικίλη δομή.

Το *MongoDB* διαθέτει πλούσιο σετ εργαλείων για την εύκολη ανάπτυξη, την παρακολούθηση και τη συντήρηση των βάσεων δεδομένων. Επιπλέον, το *MongoDB* είναι κατανεμημένο σε πολλαπλούς κόμβους και υποστηρίζει αυτόματη αντιγραφή (*replication*) και διαμερισμό (*sharding*), καθιστώντας το κατάλληλο για εφαρμογές που απαιτούν υψηλή διαθεσιμότητα και διαθεσιμότητα των δεδομένων.

Η αρχιτεκτονική του *MongoDB* αποτελείται από ένα σύνολο από διακομιστές (*servers*) που συνεργάζονται για να παρέχουν μια επιθυμητή λειτουργικότητα στη βάση δεδομένων. Οι βασικοί τύποι διακομιστών που απαρτίζουν την αρχιτεκτονική του *MongoDB* είναι οι εξής:

Κόμβοι (*Nodes*): Οι κόμβοι είναι οι βασικοί διακομιστές που εκτελούν τη βάση δεδομένων και αποθηκεύουν τα δεδομένα. Κάθε κόμβος περιέχει μια αντίγραφη των δεδομένων της βάσης δεδομένων, και μπορεί να λειτουργήσει ως πρωτεύον (*primary*) ή δευτερεύον (*secondary*) κόμβος.

Ομάδες αντιγράφων (*Replicasets*): Οι ομάδες αντιγράφων αποτελούνται από έναν πρωτεύον και μια ή περισσότερες αντίγραφα των κόμβων (*secondarynodes*). Ο πρωτεύον κόμβος είναι υπεύθυνος για τη διαχείριση των εγγραφών της βάσης δεδομένων, ενώ τα δευτερεύοντα αντίγραφα επικαιροποιούν τα δεδομένα τους από τον πρωτεύοντα κόμβο.

Διαχειριστές συστήματος (*System administrators*): Οι διαχειριστές συστήματος είναι υπεύθυνοι για τη διαχείριση όλων των κόμβων και των ομάδων αντιγράφων, καθώς και για την επίλυση προβλημάτων ασφάλειας και απόδοσης της βάσης δεδομένων.

Διακομιστές δρομολόγησης (*Routingservers*): Οι διακομιστές δρομολόγησης χρησιμοποιούνται για τη δρομολόγηση αιτημάτων στους κατάλληλους κόμβους της βάσης δεδομένων, καθώς και για τη διαχείριση του φορτίου του συστήματος.

Η αρχιτεκτονική του *MongoDB* είναι σχεδιασμένη για να είναι εύκολη στη χρήση και επεκτάσιμη, επιτρέποντας στους χρήστες να προσαρμόζουν τη βάση δεδομένων τους ανάλογα με τις ανάγκες τους. Επιπλέον, η αρχιτεκτονική αυτή παρέχει υψηλή διαθεσιμότητα και αντοχή σε αποτυχίες, καθιστώντας το *MongoDB* κατάλληλο για εφαρμογές που απαιτούν αξιοπιστία και αντοχή σε σφάλματα. Συνολικά, το *MongoDB* αποτελεί μια δημοφιλή επιλογή για την ανάπτυξη εφαρμογών μεγάλης κλίμακας και υψηλών απαιτήσεων σε θέματα διαθεσιμότητας και επεξεργαστικής ισχύος.

B. *Cassandra*

Το *Cassandra* είναι ένα ανοιχτού κώδικα, κατανεμημένο σύστημα βάσεων δεδομένων (*distributed database system*) που χρησιμοποιείται για τη διαχείριση μεγάλων όγκων δεδομένων με υψηλή κλιμακωσιμότητα (*high scalability*). Αναπτύχθηκε από την *Apache Software Foundation* και έχει χρησιμοποιηθεί ευρέως από διάφορες μεγάλες επιχειρήσεις και ιστοσελίδες, όπως η *Netflix*, η *eBay* και η *Twitter* και είναι κατάλληλο για εφαρμογές που απαιτούν υψηλή αντοχή σε αποτυχίες, όπως οι εφαρμογές ανάλυσης δεδομένων.

Η αρχιτεκτονική του *Cassandra* βασίζεται στο μοντέλο κατανεμημένων βάσεων δεδομένων (*distributed database model*), όπου τα δεδομένα αποθηκεύονται σε ένα σύνολο κόμβων (nodes) που βρίσκονται σε διαφορετικούς υπολογιστές και έχουν κυκλική οργάνωση. Επιπλέον, το *Cassandra* χρησιμοποιεί την αντιγραφή δεδομένων (*replication*) για να διατηρήσει αντίγραφα των δεδομένων σε διάφορους κόμβους, προκειμένου να διασφαλίσει τη διαθεσιμότητα των δεδομένων σε περίπτωση αποτυχίας κάποιου κόμβου. Αυτό σημαίνει ότι αν ένας κόμβος αποτύχει, το *Cassandra* μπορεί να αντιστοιχίσει τα δεδομένα σε έναν άλλο κόμβο και να διασφαλίσει την πρόσβαση των χρηστών στα δεδομένα. Κάθε ένας από αυτούς τους κόμβους αποθηκεύει ένα μέρος των δεδομένων σε ένα τοπικό σύστημα αρχείων. Αυτό επιτρέπει στο *Cassandra* να διαχειρίζεται μεγάλα σύνολα δεδομένων και να προσφέρει υψηλές επιδόσεις ανάγνωσης και εγγραφής με μεγαλή ανοχή σε λάθη.

Επιπλέον, η αρχιτεκτονική του *Cassandra* είναι αποκεντρωμένη (*decentralized*), δηλαδή δεν υπάρχει ένα κεντρικό σημείο ελέγχου στο σύστημα. Αντίθετα, οι κόμβοι λειτουργούν ως ανεξάρτητες μονάδες και συνεργάζονται μεταξύ τους για να διαχειρίζονται τα δεδομένα και να παρέχουν υπηρεσίες στους χρήστες. Για την αναζήτηση και την πρόσβαση στα δεδομένα αυτά χρησιμοποιείται ένα μοντέλο κατανεμημένου κλειδιού (*distributed key − value model*). Κάθε κλειδί αντιστοιχίζεται σε ένα κόμβο και τα δεδομένα αποθηκεύονται σε αυτόν τον κόμβο.

Τέλος στο *Cassandra*, το *keyspace* αναφέρεται σε μια λογική ομάδα κολλημένων στοιχείων δεδομένων, παρόμοια με έναν πίνακα βάσης δεδομένων σε άλλα συστήματα βάσεων δεδομένων. Ουσιαστικά, το *keyspace* περιλαμβάνει ένα σύνολο από τους πίνακες, τα δεδομένα και τις ρυθμίσεις που σχετίζονται με ένα συγκεκριμένο θέμα ή εφαρμογή. Κάθε *keyspace* έχει ένα όνομα και μπορεί να έχει ρυθμίσεις που ορίζουν τον αριθμό των αντιγράφων δεδομένων που αποθηκεύονται σε διαφορετικούς κόμβους του δικτύου *Cassandra*, καθώς και άλλες παραμέτρους, όπως η στρατηγική αντιγραφής δεδομένων, οι πολιτικές επανειλημμένων εγγραφών και η πολιτική εκκαθάρισης δεδομένων. Τα *keyspace* μπορούν να χρησιμοποιηθούν για να

οργανώσουν τα δεδομένα στο *Cassandra* σε λογικά σύνολα και να επιτρέψουν την επίλυση διαφορετικών προβλημάτων ανάλογα με τις ανάγκες της εφαρμογής.

C. Redis

Το *Redis* είναι ένας $open - source$, $in - memory$ *data structure store*, χρησιμοποιούμενος κυρίως ως *database*, *cache* και *message broker*. Η βασική λειτουργία του *Redis* είναι να αποθηκεύει δεδομένα σε μνήμη *RAM*, αντί να τα αποθηκεύει σε σκληρό δίσκο. Αυτό του επιτρέπει να παρέχει πολύ γρήγορη ανάγνωση και εγγραφή δεδομένων. Επιπλέον, μπορεί να λειτουργήσει ως *cache* για την αποθήκευση ενδιάμεσων αποτελεσμάτων και για την επιτάχυνση των εφαρμογών. Το *Redis* χρησιμοποιείται ευρέως σε διάφορες εφαρμογές όπως οικονομικές συναλλαγές, *social media*, $real - time$ *analytics*, *gaming*, και *messaging systems*.

Η δομή δεδομένων που υποστηρίζει το *Redis* περιλαμβάνει *strings*, *hashes*, *lists*, *sets*, *sorted sets* και *bitmaps*. Η σταθερή αυτή σειρά δομών δεδομένων του επιτρέπει να υποστηρίζει πολλά είδη εφαρμογών και να παρέχει μια απλή προς χρήση διεπαφή για τους προγραμματιστές.

Η αρχιτεκτονική του *Redis* βασίζεται στον αρχικό σχεδιασμό του ως μια μνήμη *cache* με διαρκή αποθήκευση σε δίσκο. Σήμερα, έχει εξελιχθεί σε ένα πλήρες σύστημα βάσης δεδομένων με πολλαπλές λειτουργίες και δυνατότητες. Αποτελείται από δύο βασικά στοιχεία: τον πυρήνα και τα παρεχόμενα *modules*.

Ο πυρήνας του *Redis* είναι υλοποιημένος σε *C* και αποτελεί τον κύριο μηχανισμό αποθήκευσης κλειδιών-τιμών. Χρησιμοποιεί μια επιλογή αλγορίθμων για τη διαχείριση των δεδομένων, συμπεριλαμβανομένων των αλγορίθμων αναζήτησης, ενημέρωσης και διαγραφής. Οι αλγόριθμοι αυτοί παρέχουν γρήγορη πρόσβαση στα δεδομένα και βελτιστοποιούν την απόδοση του συστήματος.

Τα παρεχόμενα modules του *Redis* επιτρέπουν στους χρήστες να προσθέτουν λειτουργίες και δυνατότητες στο σύστημα. Κάθε *module* περιέχει το δικό του σκοπό και μπορεί να χρησιμοποιηθεί για διαφορετικούς σκοπούς, όπως η επεξεργασία γραφημάτων ή η αποστολή *email*. Τα *modules* είναι επίσης υλοποιημένα σε *C* και μπορούν να προστεθούν στον πυρήνα του *Redis* χρησιμοποιώντας τη δυνατότητα του *Redis* για δυναμική φόρτωση κώδικα.

Όπως αναφέρθηκε παραπάνω, τα δεδομένα αποθηκεύονται σε μνήμη *RAM* αλλά το *Redis* παρέχει επίσης δυνατότητα αποθήκευσης σε δίσκο. Αυτό το χαρακτηριστικό επιτρέπει στους χρήστες να διατηρούν τα δεδομένα τους ακόμα και μετά από επανεκκίνηση του συστήματος.

Επίσης επιτρέπεται η χρήση πολλαπλών επεξεργαστών ($multi - threading$) και η υλοποίησή αυτή βασίζεται στη χρήση ασύγχρονων μηχανισμών I/O. Αυτό σημαίνει ότι οι λειτουργίες του *Redis* εκτελούνται ασύγχρονα, χωρίς να απαιτείται να περιμένουν την ολοκλήρωση μιας άλλης λειτουργίας που εκτελείται ταυτόχρονα. Αυτό καθιστά δυνατή την εκτέλεση πολλών εργασιών ταυτόχρονα και την αύξηση της απόδοσης.

Τέλος, το *Redis* υποστηρίζει πολλές δυνατότητες όπως το $publish/subscribe$ μοντέλο, το οποίο επιτρέπει την επικοινωνία μεταξύ πολλών πελατών και τη συνεχή ροή δεδομένων σε πραγματικό χρόνο. Επίσης, υποστηρίζει τη δημιουργία αντιγράφων επιφανείας καταγραφής ($write - aheadlogging$), τη δυνατότητα εφαρμογής πολλαπλών λειτουργιών σε ένα ατομικό κλειδί, και την εύκολη αλλαγή του συνόλου των δεδομένων που αποθηκεύονται σε έναν κόμβο χωρίς να απαιτείται η διακοπή της λειτουργίας του.

Συνολικά, η αρχιτεκτονική του Redis είναι σχεδιασμένη για την αποθήκευση και διαχείριση μεγάλου όγκου δεδομένων σε πραγματικό χρόνο με υψηλή απόδοση και κλιμακωσιμότητα.

D. Trino

Όπως αναφέρθηκε και προηγουμένως, *Trino* (προηγουμένως γνωστό ως $PrestoSQL$) είναι ένα διανεμημένο σύστημα επερωτήσεων SQL που επιτρέπει στους χρήστες να εκτελούν αποτελεσματικά επερωτήσεις σε μεγάλα συνόλα δεδομένων από διάφορες πηγές δεδομένων.

Η τεχνολογία *Trino* αναπτύχθηκε αρχικά από την εταιρεία *Facebook*, αλλά αργότερα αναπτύχθηκε ως ένα έργο ανοιχτού κώδικα. Το *Trino* έχει σχεδιαστεί για να είναι εξαιρετικά αποδοτικό, με τη δυνατότητα εκτέλεσης επερωτήσεων που επεξεργάζονται δεδομένα σε πολλαπλούς κόμβους, επιτρέποντας την κατανεμημένη επεξεργασία δεδομένων και την αποτελεσματική εκτέλεση επερωτήσεων.

Μερικά από τα χαρακτηριστικά του *Trino* περιλαμβάνουν:

Υποστήριξη για πολλαπλούς τύπους πηγών δεδομένων, όπως *Hadoop*, *Cassandra*, *MongoDB*, *MySQL* και πολλούς άλλους. Υποστήριξη απομακρυσμένης επεξεργασίας δεδομένων μέσω $JDBC/ODBC$ *drivers*. Υποστήριξη προσαρμοστικής επεξεργασίας δεδομένων μέσω επαγωγικών προτύπων. Υψηλή απόδοση και κλιμάκωση δεδομένων.

Η τεχνολογία *Trino* είναι ιδιαίτερα χρήσιμη για επιχειρήσεις και άλλους οργανισμούς που χειρίζονται μεγάλα σύνολα δεδομένων και αναζητούν μια αποτελεσματική λύση για τη διαχείρισή τους. Με την τεχνολογία *Trino*, οι χρήστες μπορούν να εκτελούν επερωτήσεις SQL σε μεγάλα σύνολα δεδομένων και να λαμβάνουν αποτελέσματα σε πραγματικό χρόνο.

Το *Trino* έχει επίσης πλούσια κοινότητα χρηστών και προγραμματιστών, που παρέχουν υποστήριξη και προσθέτουν συνεχώς νέες λειτουργίες και βελτιώσεις στο σύστημα. Η τεχνολογία *Trino* είναι επίσης δωρεάν και ανοιχτού κώδικα, οπότε οι χρήστες μπορούν να την προσαρμόσουν στις ανάγκες τους και να δημιουργήσουν προσαρμοσμένες εφαρμογές.

Συνολικά, το *Trino* είναι μια ισχυρή τεχνολογία για τη διαχείριση μεγάλων συνόλων δεδομένων και την εκτέλεση αποτελεσματικών επερωτήσεων SQL σε αυτά τα σύνολα δεδομένων.

Η αρχιτεκτονική του *Trino* σχεδιάστηκε για να είναι εύκολη στη χρήση, επεκτάσιμη και να παρέχει υψηλή απόδοση σε μεγάλα σύνολα δεδομένων. Στη βάση της, η αρ-

χιτεκτονική του *Trino* αποτελείται από δύο κύρια τμήματα: τον *Trino Coordinator* και τους *Trino Workers*.

Ο *Trino Coordinator* είναι ο κεντρικός κόμβος του συστήματος και διαχειρίζεται τις ερωτήσεις από τους χρήστες. Όταν ένας χρήστης υποβάλλει μια ερώτηση *SQL* στο *Trino*, ο *Trino Coordinator* διαμορφώνει το ερώτημα και το αποστέλλει στους κατάλληλους *Trino Workers* για επεξεργασία.

Οι *Trino Workers* είναι οι κόμβοι που πραγματοποιούν τις πραγματικές επερωτήσεις και ανακτούν τα δεδομένα από το αποθετήριο δεδομένων (όπως ένα σύστημα *Hadoop* ή ένα σύστημα αποθήκευσης αρχείων). Κάθε *Trino Worker* είναι σχεδιασμένος για να εκτελεί αρκετές ερωτήσεις ταυτόχρονα, και οι εργασίες κατανέμονται στους *Trino Workers* με ένα σύστημα διαχείρισης εργασιών.

Το *Trino* υποστηρίζει πολλαπλές πηγές δεδομένων και μπορεί να επεκταθεί εύκολα για να χειριστεί μεγαλ

E. MongoDB vs Cassandra

Ορισμένες από τις σημαντικότερες διαφορές ανάμεσα στη *Cassandra* και τη *MongoDB* είναι οι εξής:

Δομή δεδομένων: Η *Cassandra* χρησιμοποιεί ένα πίνακα επικεφαλίδων-στηλών για την αποθήκευση δεδομένων, ενώ η *MongoDB* χρησιμοποιεί έγγραφα *BSON*. Αυτό σημαίνει ότι η *Cassandra* είναι καλύτερη για περιστατικά που απαιτούν υψηλές ταχύτητες ανάγνωσης και εγγραφής ενώ η *MongoDB* είναι καλύτερη για περιστατικά όπου τα δεδομένα είναι πιο πολύπλοκα.

Συμμετοχή κόμβων: Η *Cassandra* έχει ένα διακομιστή κατανεμημένων κόμβων και κατανεμημένων αντιγράφων δεδομένων, ενώ η *MongoDB* έχει έναν κύριο κόμβο και αντίγραφα δεδομένων. Αυτό σημαίνει ότι η *Cassandra* είναι πιο αξιόπιστη για τη διαχείριση μεγάλων όγκων δεδομένων και χρησιμοποιείται συχνότερα σε μεγάλες επιχειρήσεις, ενώ η *MongoDB* είναι καλύτερη για μικρότερες εφαρμογές που χρειάζονται απλή διαχείριση δεδομένων.

Επεκτασιμότητα: Η *Cassandra* είναι πιο επεκτάσιμη από τη *MongoDB* και μπορεί να αντεπεξέλθει σε μεγάλες εκτάσεις. Η *Cassandra* επιτρέπει στους χρήστες να προσθέτουν περισσότερους κόμβους στο δίκτυό τους, χωρίς να επηρεάζεται η απόδοση. Η *MongoDB* είναι επίσης επεκτάσιμη, αλλά σε μικρότερο βαθμό από τη *Cassandra*.

Απόδοση: Η *Cassandra* είναι σχεδιασμένη για υψηλές απαιτήσεις απόδοσης και μπορεί να χειριστεί μεγάλους όγκους δεδομένων με χαμηλούς χρόνους απόκρισης. Από την άλλη πλευρά, η *MongoDB* είναι πιο κατάλληλη για λιγότερο απαιτητικές εφαρμογές.

Ευελιξία: Η *MongoDB* είναι πιο ευέλικτη από τη *Cassandra* και επιτρέπει στους χρήστες να αλλάζουν τη δομή των δεδομένων όπως επιθυμούν, ενώ η *Cassandra* απαιτεί μια σταθερή δομή δεδομένων.

Από τα παραπάνω χαρακτηριστικά περιμένουμε η *Cassandra* να ανταπεξέλθει καλύτερα στην εκτέλεση των *queries* από τη *MongoDB* εφόσον είναι ταχύτερη. Ο όγκος δεδομένος που θα χρησιμοποιήσουμε δεν είναι τεράστιος - 1.1 *GB* - αλλά επίσης δεν έχουμε πολύπλοκα δεδομένα. Επομένως τα πλεονεκτήματα της MongoDB δεν περιμένουμε να φανούν στη δικιά μας περίπτωση.

F. MongoDB vs Redis

Μερικές συγκρίσεις μεταξύ *MongoDB* και *Redisεναιοιεξς* :

Το MongoDB είναι μια δισκοβάση δεδομένων, ενώ το Redis είναι μια in-memory βάση δεδομένων. Το MongoDB είναι καλύτερο για την αποθήκευση μεγάλων όγκων δεδομένων και την αναζήτηση σε πολλά πεδία, ενώ το Redis είναι καλύτερο για την αντιμετώπιση μικρών δεδομένων με υψηλή απόκριση. Το MongoDB είναι πιο ευέλικτο στη σχεδίαση και μπορεί να προσαρμοστεί σε διαφορετικές ανάγκες εφαρμογών, ενώ το Redis έχει ένα πιο περιορισμένο αριθμό λειτουργιών. Το MongoDB παρέχει υψηλή ανθεκτικότητα και δυνατότητα κλιμάκωσης, ενώ το Redis είναι σχεδιασμένο για υψηλές επιδόσεις σε περιβάλλοντα με μικρά δεδομένα. Συνολικά, η επιλογή μεταξύ του MongoDB και του Redis εξαρτάται από τις ανάγκες της εφαρμογής. Αν για παράδειγμα η εφαρμογή απαιτεί την αποθήκευση μεγάλων όγκων δεδομένων, τότε το MongoDB θα ήταν μια καλή επιλογή. Αντίθετα, αν η εφαρμογή σας απαιτεί ταχύτητα και χαμηλή καθυστέρηση, τότε το Redis θα ήταν μια καλή επιλογή.

Επιπλέον, οι δυο αυτές βάσεις δεδομένων μπορούν να χρησιμοποιηθούν μαζί, συμπληρώνοντας τις ανάγκες της εφαρμογής σας. Για παράδειγμα, μπορείτε να χρησιμοποιήσετε το MongoDB για την αποθήκευση μεγάλων όγκων δεδομένων και το Redis για την παρακολούθηση και ανανέωση μικρών στοιχείων σε πραγματικό χρόνο.

Στην προκειμένη περίπτωση κατά τη σύγκριση μεταξύ των δύο βάσεων περιμένουμε η Redis να έχει καλύτερα αποτελέσματα εφόσον έχουμε σχετικά μικρά δεδομένα. Ωστόσο μέσω του Trino θα μπορέσουμε να εκμεταλλευτούμε και τα πλεονκεκτήματα των δύο βάσεων.

G. Redis vs Cassandra

Μερικές συγκρίσεις μεταξύ της *Redis* και *Cassandra* είναι οι εξής:

Η *Redis* είναι μια $in-memory$ βάση δεδομένων, ενώ η *Cassandra* υποστηρίζει την αποθήκευση σε δίσκο και την ανάγνωση από τη μνήμη. Αυτό σημαίνει ότι η *Redis* είναι πολύ πιο γρήγορη για αναγνώσεις και εγγραφές, αλλά η *Cassandra* είναι καλύτερη για μεγάλα σετ δεδομένων και για εφαρμογές που απαιτούν υψηλή διαθεσιμότητα και αντοχή σε σφάλματα. Η *Redis* υποστηρίζει διάφορα *data structures*, όπως *strings*, *hashes*, *lists*, *sets* και *sorted sets* και άρα η *Redis* είναι πιο κατάλληλη για εφαρμογές που απαιτούν ποικιλία συναρτήσεων και δεδομένων, ενώ η *Cassandra* είναι καλύτερη για εφαρμογές που απαιτούν υψηλή αντοχή και αναζήτηση μεγάλων σετ δεδομένων.

Η *Redis* υποστηρίζει μόνο μία μορφή αντιγραφής ασφαλείας, ενώ η *Cassandra* υποστηρίζει διάφορες μορφές αντιγράφων ασφαλείας, όπω

IV. Δεδομένα & $Queries$

A. Περιγραφή των Δεδομένων

Για την παραγωγή των δεδομένων αναπτύξαμε κώδικα σε $Python$ ο οποίος είναι διαθέσιμος στο $github\ link$. Δημιουργήσαμε τρεία διαφορετικά $tables$ ώστε να μπορέσουμε να εκτέλουσουμε $queries$ με $joins$ και $aggregations$. Συγκεκριμένα τα τρεία $tables$ είναι οι πίνακες $WORKER_i$, $BONUS_i$, $TITLE_i$, όπου i = 1,2,3 αντίστοιχα για τις 3 διαφορετικές βάσεις. Συγκεκριμένα τα $WORKER_1$, $BONUS_1, TITLES_1$ είναι τα δεδομένα του $Cassandra$, τα $WORKER_2$, $BONUS_2$, $TITLE_2$ τα δεδομένα του $Redis$ και αντίστοιχα τα $WORKER_3$, $BONUS_3$, $TITLE_3$ τα δεδομένα του $MongoDB$. Ο πίνακας $WORKER_i$ εμπεριέχει όλους τους υπάλληλους μίας εταιρίας, ο πίνακας $BONUS_i$ τους υπάλληλους που πήραν bonus και ο $TITLE_i$ τη θέση των υπαλλήλων. Οι $WORKER_i$ και $TITLE_i$ έχουν ένα row για κάθε υπάλληλο ενώ το Bonus λιγότερα. Επομένως, παράξαμε 15.000.000 σειρές για τους πίνακες $WORKER_i$ και $TITLE_i$ και 1.500.000 σειρές για τον πίνακα $BONUS_i$.

$WORKER_i$

$Keys$	Data Types
$WORKER\_REF\_ID$	int
$BONUS\_AMOUNT$	char
$BONUS\_DATE$	char

$BONUS_i$

$Keys$	Data Types
$WORKER\_REF\_ID$	int
$BONUS_AMOUNT$	char
$BONUS\_DATE$	char

$TITLE_i$

$Keys$	Data Types
$WORKER\_REF\_ID$	int
$WORKER\_TITLE$	char
$AFFECTED\_FROM$	char

B. Queries

Εφόσον η δημιουργία πολλών διαφορετικών queries είναι ιδιαίτερα χρονοβόρα αποφασίσαμε να βρούμε έτοιμα. Βρήκαμε στο github [link gia to github] ένα project με SQL queries πάνω σε αντίστοιχους πίνακες με τους δικούς μας. Συνολικά, χρησιμοποιήθηκαν 50 queries. Κάποια από αυτά ήταν ιδιαίτερα απλά και αλλά πιο σύνθετα, με πράξεις join και aggregate όπως ζητήθηκε στην εκφώνηση.

V. Αποτελέσματα & $Συμπερσματα$

Στη συνέχεια παρουσιάζονται τα αποτελέσματα από την πειραματική διαδικασία. Αρχικά θα εκτελέσουμε ξεχωριστά queries στους πίνακες της κάθε βάσης ώστε να μπορέσουμε να συγκρίνουμε τους χρόνους με τα queries που τρέχουν σε όλες τις βάσεις συνδυαστικά.

A. Εκτέλεση *query1* στην *MongoDB*

select * from mongogb.final.worker
where mongogb.final.worker.department='IT' and mongogb.final.worker.last_name='Diwan' and mongogb.final.worker.first_name='Eve';

Η εκτέλεση του query1 κράτησε 5.21 λεπτά στη MongoDB.

B. Εκτέλεση *query1* στην *Redis*

Στη συνέχεια εκτελέσαμε το ίδιο query1 και στην Redis. Η εκτέλεση του query1 κράτησε 9 δευτερόλεπτα στη Redis.

C. Εκτέλεση *query1* στην *MongoDB* και στο *Redis*

Έπειτα εκτελούμε το ίδιο query1 τόσο στην MongoDB όσο και στην Redis.

Η εκτέλεση του query1 κράτησε 6 λεπτά.

D. Συμπεράσματα

Στη Redis τα δεδομένα αποθηκεύονται σε ένα hash table με τη μορφή (key, value), κατά τη διάρκεια της αναζήτησης, η Redis κάνει hash στο κλειδί και παράγει μία τιμή. Για αυτό το λόγο είναι τόσο γρήγορη.

References

[1] R. Sethi et al., "Presto: SQL on Everything," 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 2019, pp. 1802-1813, doi: 10.1109/ICDE.2019.00196.

[2] Matt Fuller, Manfred Moser, Martin Traverso, Trino: The Definitive Guide[Online]. Available: https://www.wisdominterface.com/wp-content/uploads/2021/07/Trino-Oreilly-Guide.pdf

Redis

https://chartio.com/resources/tutorials/how-to-get-all-keys-in-redis/

https://stackoverflow.com/questions/50840707/how-to-import-a-csv-data-file-into-the-redis-database https://livebook.manning.com/book/redis-in-action/chapter-1/95 https://stackoverflow.com/questions/53315481/what-is-keyspace-in-redis

MongoDB

https://www.mongodb.com/docs/v4.4/tutorial/query-documents/

https://www.mongodb.com/docs/manual/reference/method/db.collection.insert/

https://hevodata.com/learn/mongoimport/

https://www.tutorialsteacher.com/mongodb/mongodb-shell-commands

Cassandra

https://cassandra.apache.org/doc/latest/cassandra/getting_started/installing.html/

https://www.tutorialspoint.com/cassandra/cassandra$_c reate_k eyspace.html/$