PROJECT REVIEW DOCUMENTATION

DATABASE: NEO4J

17BCB0013 17BCB0082

17BCB0112 17BCI0058

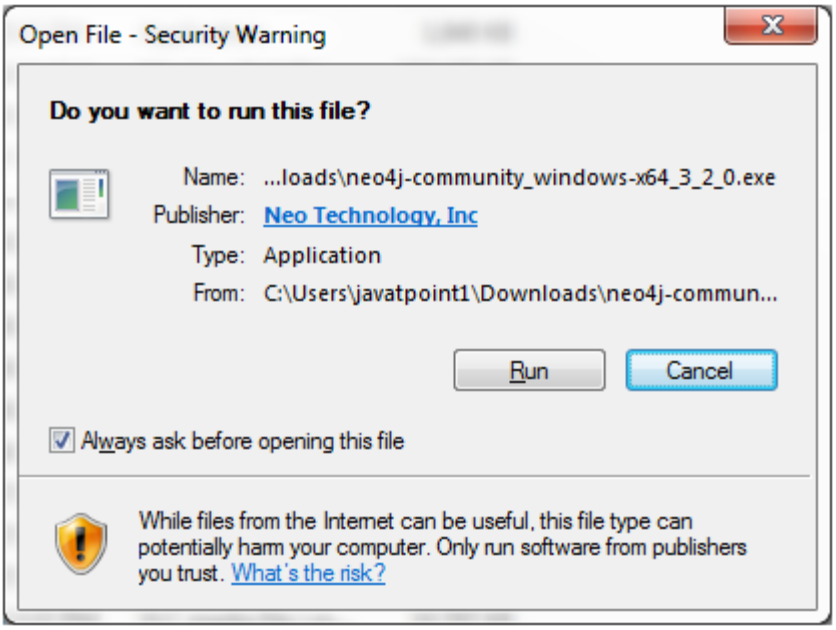
INSTALLATION GUIDE:

Download the correct package from https://neo4j.com/download-center/

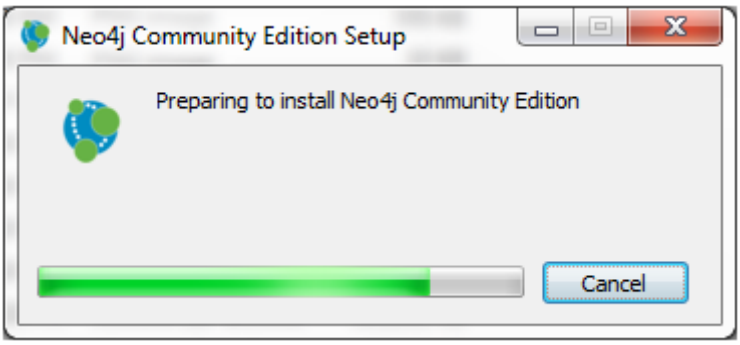
Download Neo4j

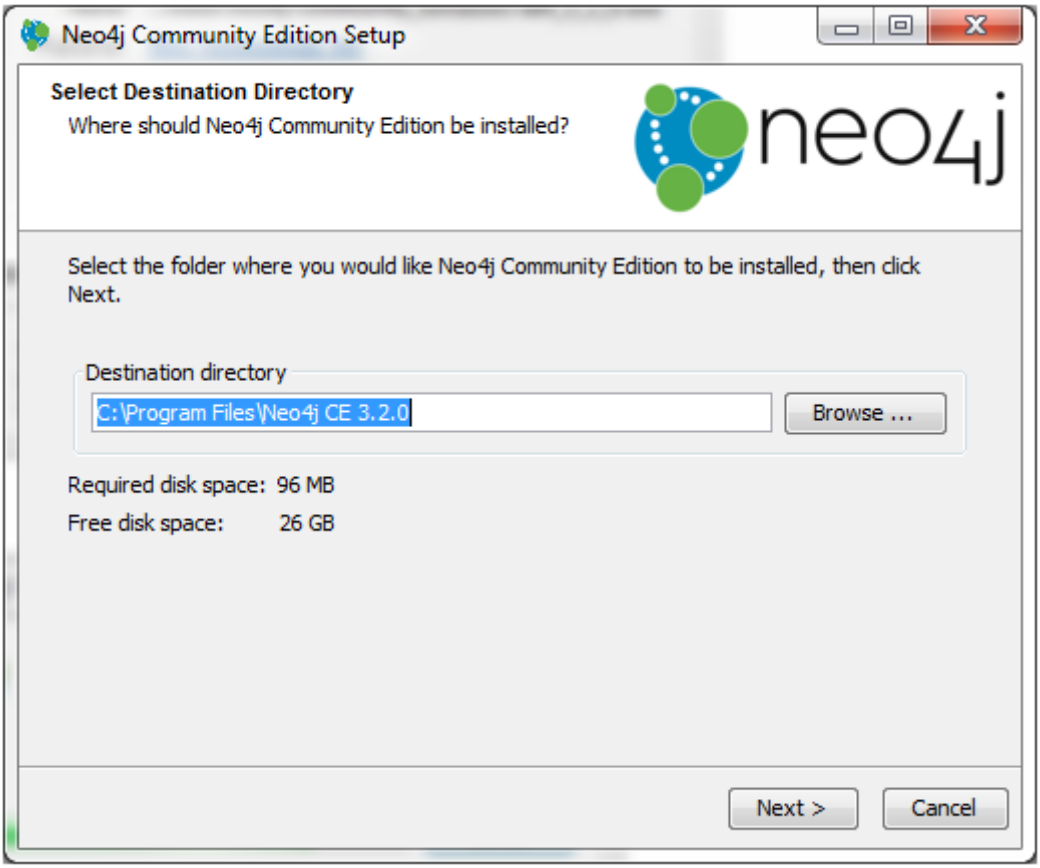
First download Neo4j from its official website: https://neo4j.com/download/

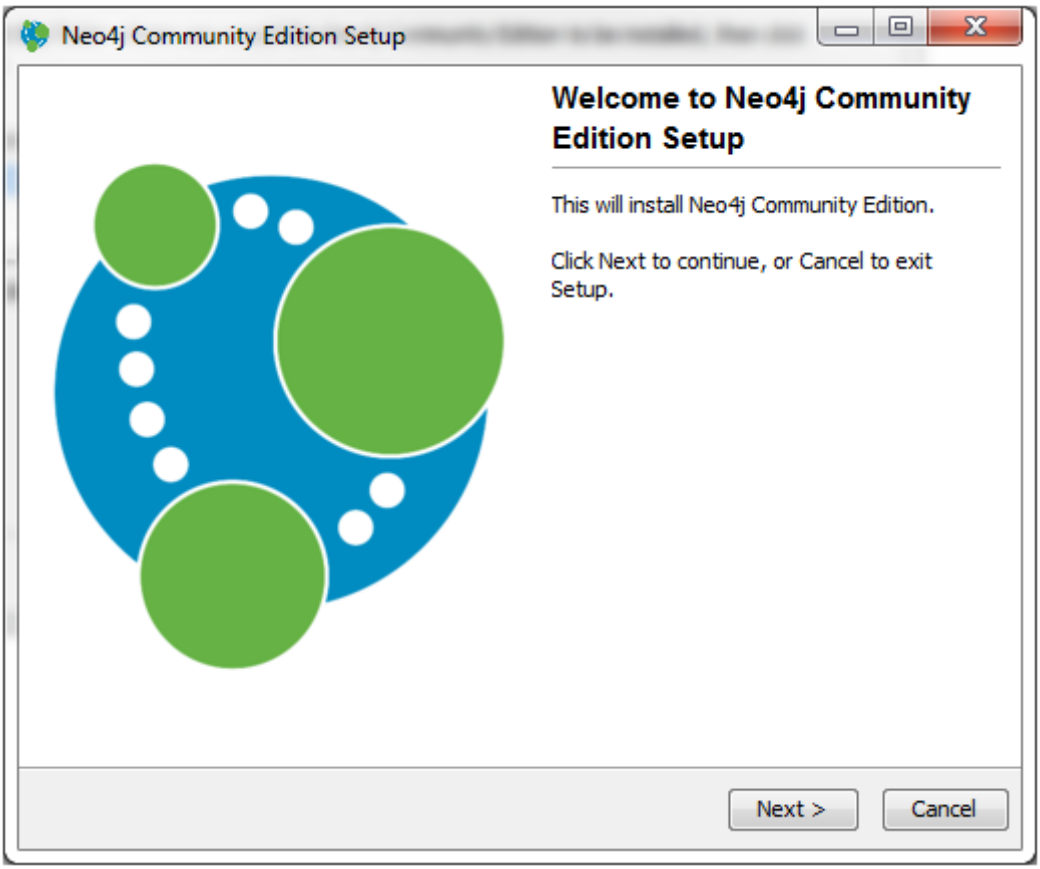
You can choose from either a free Enterprise Trial, or the free Community Edition. Here, we are using the Community Edition.

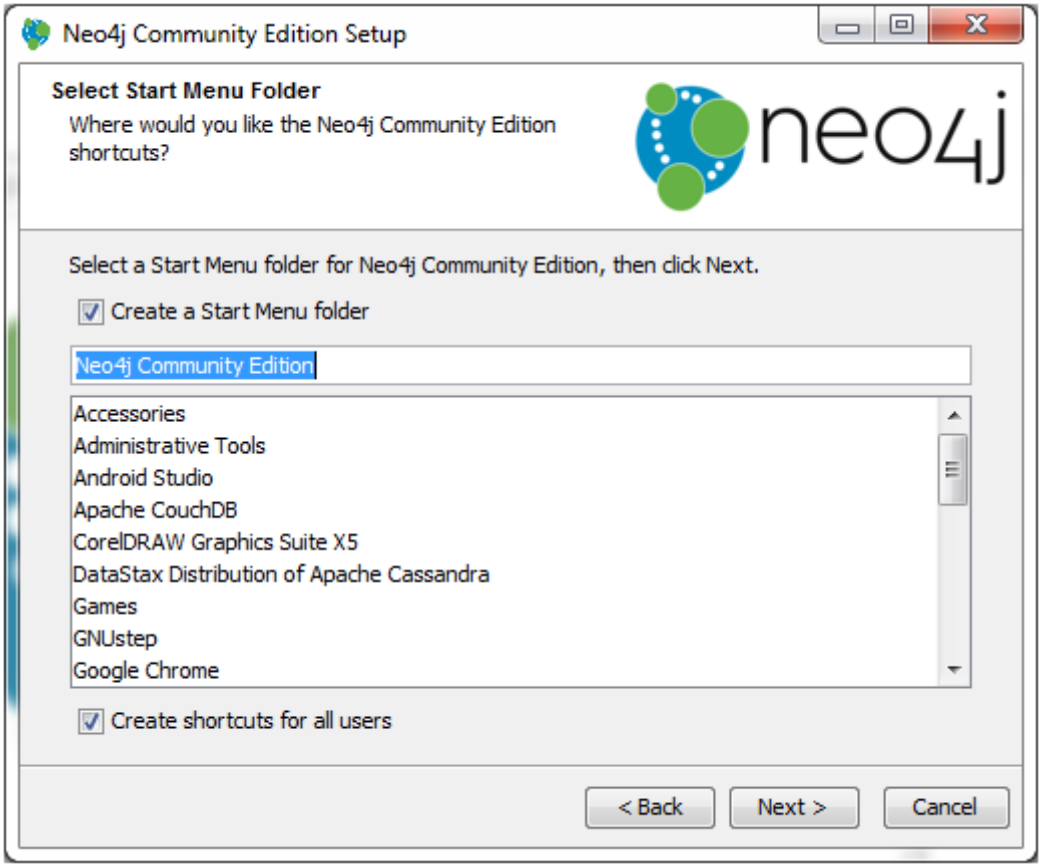


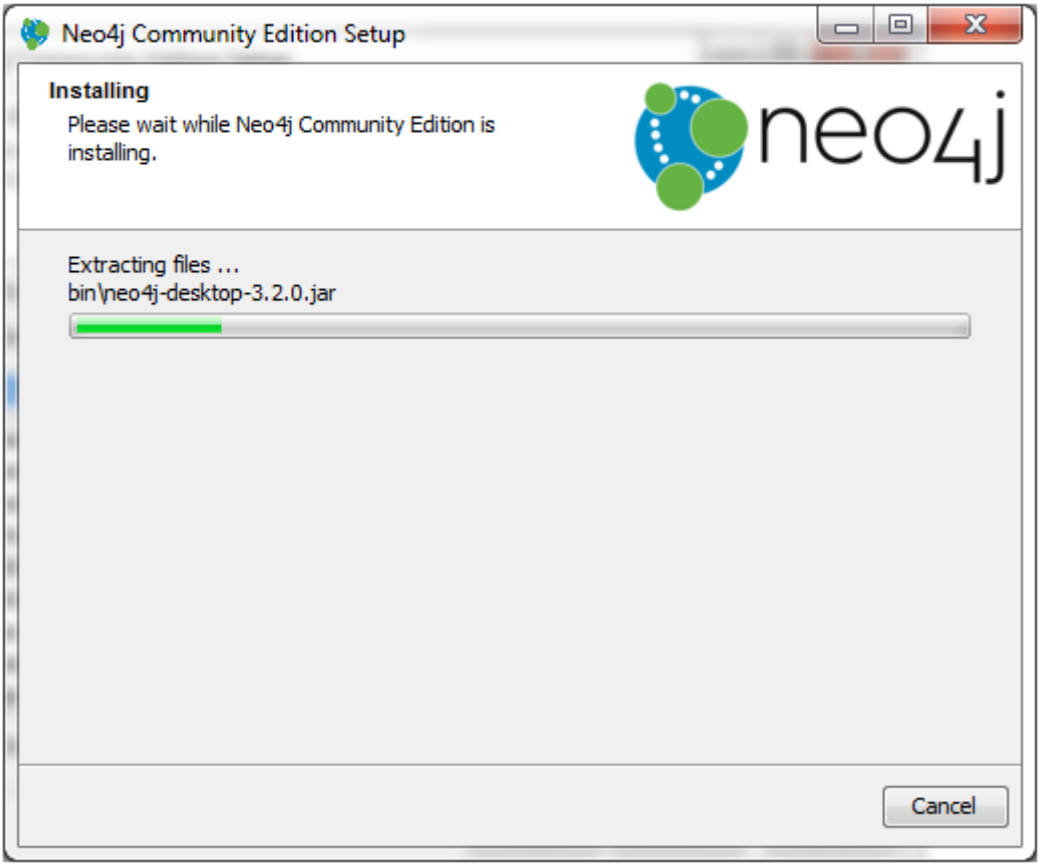
Run the downloaded file and follow the instructions given below:

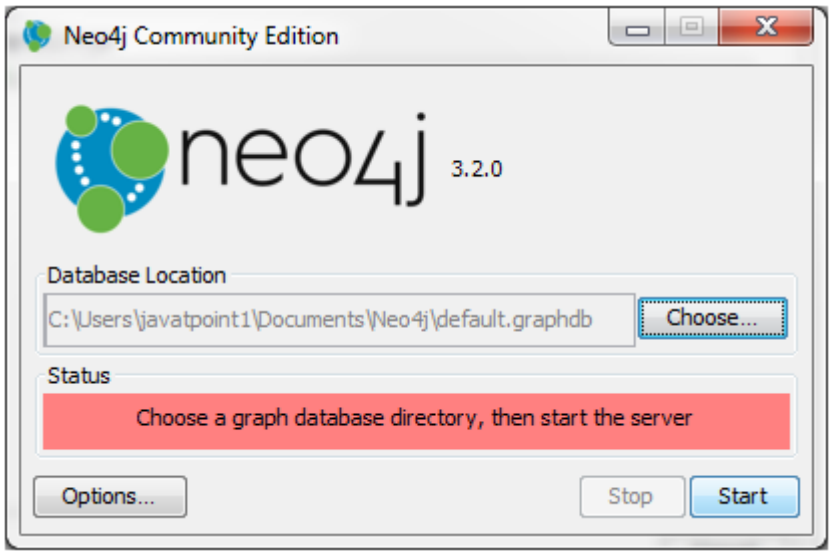

Neo4j Community Edition Setup
Preparing to install Neo4j Community Edition
Cancel


Neo4j Community Edition Setup
Select Destination Directory
Where should Neo4j Community Edition be installed?
neo4j
Select the folder where you would like Neo4j Community Edition to be installed, then click Next.
Destination directory
C:\Program Files\Neo4j CE 3.2.0
Browse ...
Required disk space: 96 MB
Free disk space: 26 GB
Next >
Cancel


Neo4j Community Edition Setup
Welcome to Neo4j Community Edition Setup
This will install Neo4j Community Edition.
Click Next to continue, or Cancel to exit Setup.
Next >
Cancel


Neo4j Community Edition Setup
Select Start Menu Folder
Where would you like the Neo4j Community Edition shortcuts?
neo4j
Select a Start Menu folder for Neo4j Community Edition, then click Next.
Create a Start Menu folder
Neo4j Community Edition
Accessories
Administrative Tools
Android Studio
Apache CouchDB
CorelDRAW Graphics Suite X5
DataStax Distribution of Apache Cassandra
Games
GNUstep
Google Chrome
Create shortcuts for all users
< Back
Next >
Cancel

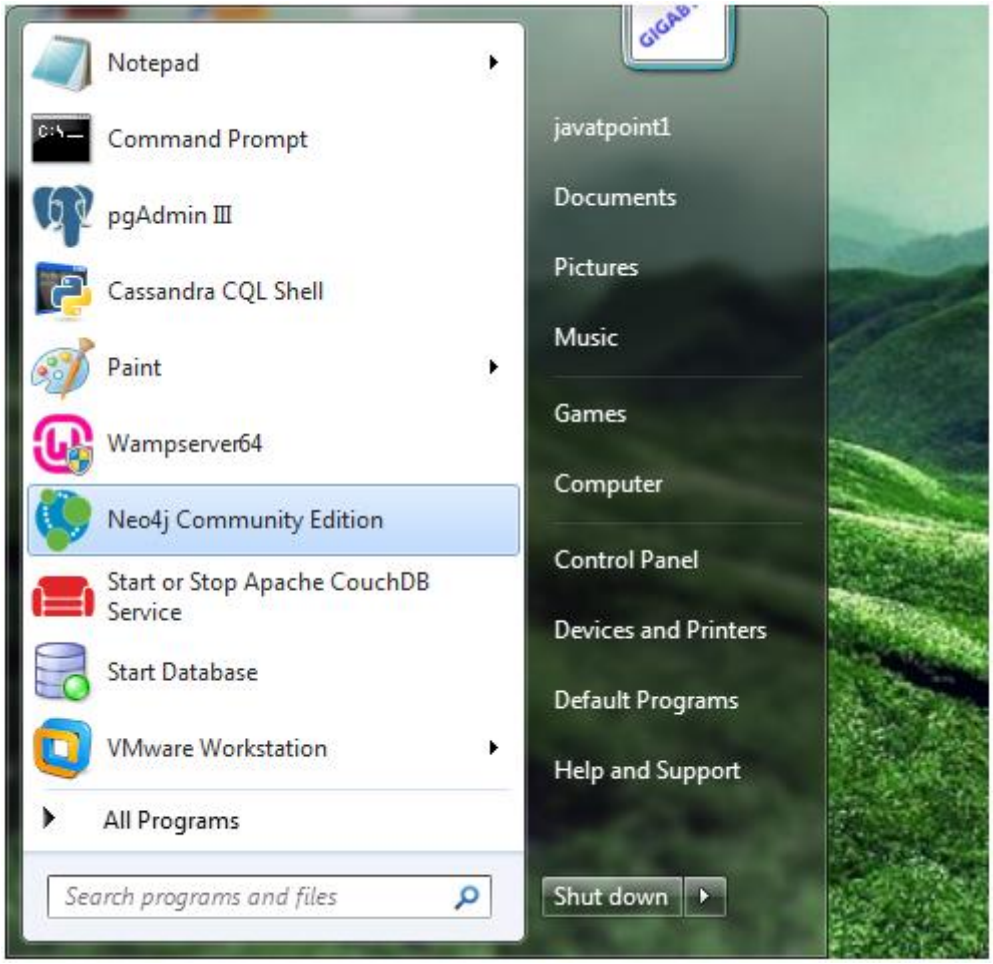

Neo4j Community Edition Setup
Installing
Please wait while Neo4j Community Edition is installing.
neo4j
Extracting files ...
bin\neo4j-desktop-3.2.0.jar
Cancel



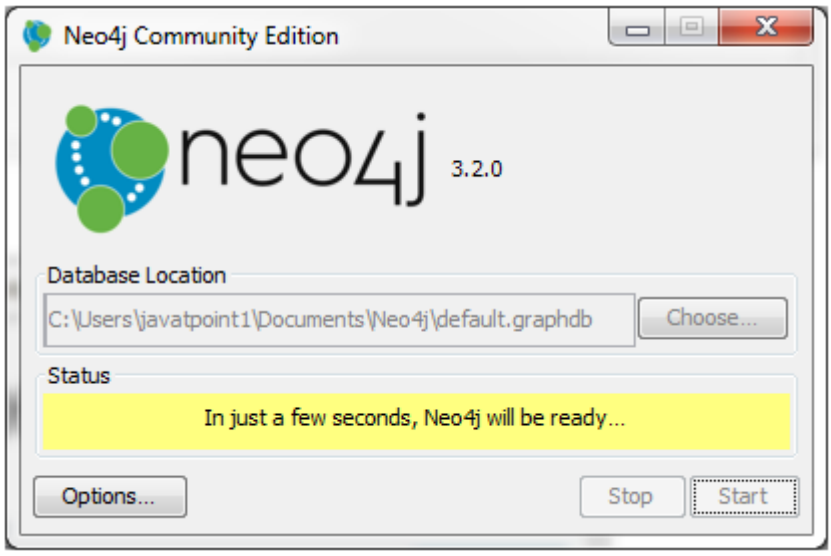
Start Neo4j:

Start the Server

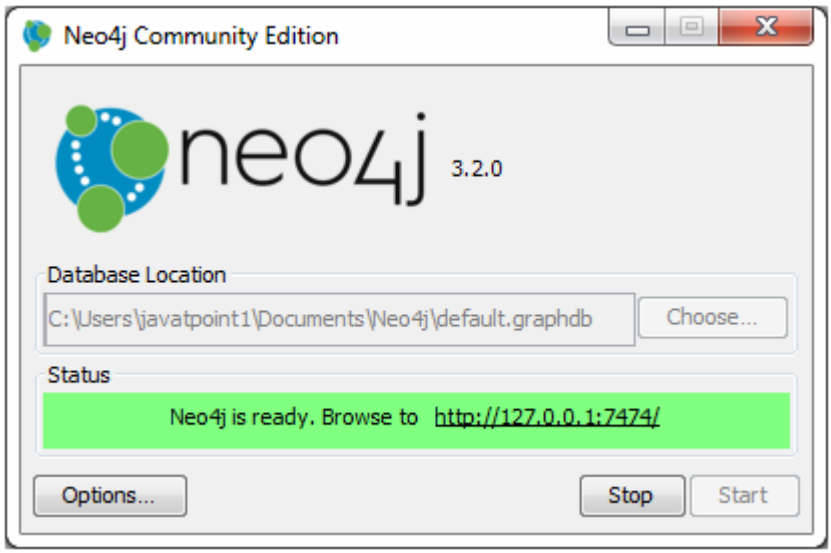
Click on the installed Neo4j Community Edition.



Initialization started:



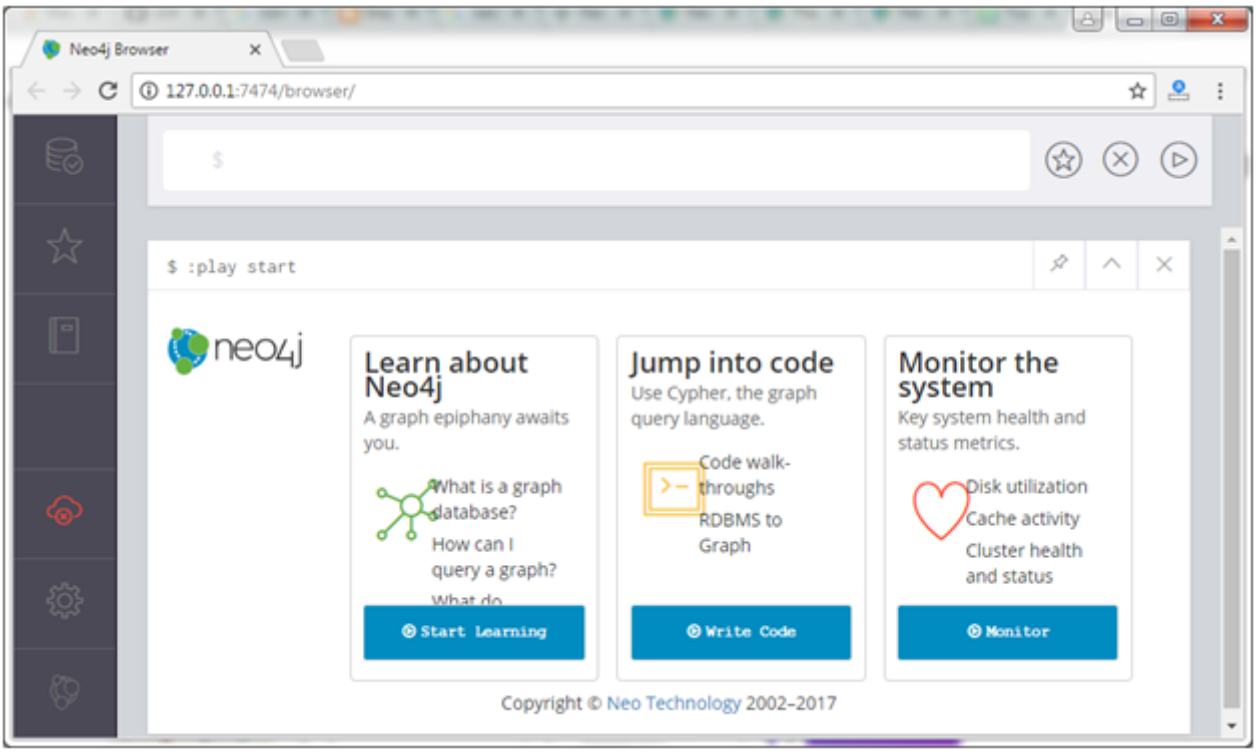
Neo4j is started. It is ready to use.



Open browser and go to local host : **http://localhost:7474/browser/**

Or

http://127.0.0.1:7474/browser/



Start a new project. Add a graph, remember the login credentials assigned to it. Execute necessary commands to create the graph (database) and add data to it.

ABOUT THE DATABASE:

Neo4j is a graph database management system developed by Neo4j, Inc. Described by its developers as an ACID-compliant transactional database with native graph storage and processing, Neo4j is the most popular graph database according to DB-Engines ranking, and the 22nd most popular database overall.

Neo4j is available in a GPL3-licensed open-source "community edition", with online backup and high availability extensions licensed under the terms of the Affero General Public License. Neo also licenses Neo4j with these extensions under closed-source commercial terms.

Neo4j is implemented in Java and accessible from software written in other languages using the Cypher Query Language through a transactional HTTP endpoint, or through the binary "bolt" protocol

DATABASE COMMANDS USED:

CREATE

(sh1:sh_Seller{Customer_id:'101',Book_id:'98745',Phone_no:'987456321',Current_year:'2018'}),

(sh2:sh_Seller{Customer_id:'102',Book_id:'98746',Phone_no:'987456322',Current_year:'2016'}),

(sh3:sh_Seller{Customer_id:'103',Book_id:'98747',Phone_no:'987456323',Current_year:'2015'}),

(sh4:sh_Seller{Customer_id:'104',Book_id:'98748',Phone_no:'987456324',Current_year:'2014'}),

(sh5:sh_Seller{Customer_id:'105',Book_id:'98749',Phone_no:'987456325',Current_year:'2012'}),

(sh6:sh_Seller{Customer_id:'106',Book_id:'98750',Phone_no:'987456326',Current_year:'2015'}),

(sh7:sh_Seller{Customer_id:'107',Book_id:'98751',Phone_no:'987456327',Current_year:'2014'}),

(sh8:sh_Seller{Customer_id:'108',Book_id:'98752',Phone_no:'987456328',Current_year:'2017'}),

(sh9:sh_Seller{Customer_id:'109',Book_id:'98753',Phone_no:'987456329',Current_year:'2018'}),

(sh10:sh_Seller{Customer_id:'110',Book_id:'98754',Phone_no:'987456330',Current_year:'2011'})

RETURN *

CREATE CONSTRAINT ON (sh:sh_Seller) ASSERT sh.Customer_id IS UNIQUE

CREATE CONSTRAINT ON (sh:sh_Seller) ASSERT exists(sh.Customer_id)

CREATE

(b1:Books{book_id:'98745',book_name:'bs_grewal',no_of_resale:'2',shopowner_price:'200',current_price:'300'}),

(b2:Books
{book_id:'98746',book_name:'Discrete_Mathematics_and_its_Applications',no_of_resale:'3',shopowner_price:'150',current_price:'300'}),

(b3:Books{book_id:'98747',book_name:'Introduction_to_Algorithms',no_of_resale:'4',shopowner_price:'100',current_price:'299'}),

(b4:Books {book_id:'98748',book_name:'Let_Us_C',no_of_resale:'6',shopowner_price:'100',current_price:'150'}),

(b5:Books {book_id:'98749',book_name:'Let_Us_C++',no_of_resale:'7',shopowner_price:'100',current_price:'200'}),

(b6:Books
{book_id:'98750',book_name:'Introduction_to_the_Theory_of_Computation',no_of_resale:'1',shopowner_price:'300',current_price:'400'}),

(b7:Books {book_id:'98751',book_name:'Hacking:_The_Art_of_Exploitation',no_of_resale:'5',shopowner_price:'299',current_price:'400'}),

(b8:Books {book_id:'98752',book_name:'Code_Complete',no_of_resale:'5',shopowner_price:'299',current_price:'500'}),

(b9:Books {book_id:'98753',book_name:'Introduction_to_Python',no_of_resale:'10',shopowner_price:'100',current_price:'199'}),

(b10:Books {book_id:'98754',book_name:'Introduction_to_Linear_Algebra',no_of_resale:'9',shopowner_price:'100',current_price:'199'}),

(b11:Books{book_id:'18745',book_name:'bs_grewal',no_of_resale:'0',current_price:'500',shopowner_price:'400'}),

(b12:Books{book_id:'18746',book_name:'Discrete_Mathematics_and_its_Applications',no_of_resale:'0',current_price:'600',shopowner_price:'450'
}),

(b13:Books{book_id:'18747',book_name:'Introduction_to_Algorithms',no_of_resale:'0',current_price:'400',shopowner_price:'299'}),

(b14:Books{book_id:'18748',book_name:'Let_Us_C',no_of_resale:'0',current_price:'300',shopowner_price:'299'}),

(b15:Books{book_id:'18749',book_name:'Let_Us_C++',no_of_resale:'0',current_price:'300',shopowner_price:'200'}),

(b16:Books{book_id:'18750',book_name:'Introduction_to_the_Theory_of_Computation',no_of_resale:'0',current_price:'700',shopowner_price:'60
0'}),

(b17:Books{book_id:'18751',book_name:'Hacking:_The_Art_of_Exploitation',no_of_resale:'0',current_price:'999',shopowner_price:'799'}),

(b18:Books{book_id:'18752',book_name:'Code_Complete',no_of_resale:'0',current_price:'999',shopowner_price:'599'}),

(b19:Books{book_id:'18753',book_name:'Introduction_to_Python',no_of_resale:'0',current_price:'700',shopowner_price:'399'}),

(b20:Books{book_id:'18754',book_name:'Introduction_to_Linear_Algebra',no_of_resale:'0',current_price:'700',shopowner_price:'499'})

RETURN *

CREATE CONSTRAINT ON (b:Books) ASSERT b.book_id IS UNIQUE

CREATE CONSTRAINT ON (b:Books) ASSERT exists(b.book_id)

create

(`t` :Transaction {Transaction_id: '' , Book_price: ''})

```
create

(c1:Customers{Customer_id:'4001',Phone_no:'7845128956',Book_id:'98745',Current_year:''}),

(c2:Customers{Customer_id:'4002',Phone_no:'4512788956',Book_id:'98746',Current_year:''}),

(c3:Customers{Customer_id:'4003',Phone_no:'1245788956',Book_id:'98747',Current_year:''}),

(c4:Customers{Customer_id:'4004',Phone_no:'8956237845',Book_id:'98748',Current_year:''}),

(c5:Customers{Customer_id:'4005',Phone_no:'5623897845',Book_id:'98749',Current_year:''}),

(c6:Customers{Customer_id:'4006',Phone_no:'2356897845',Book_id:'98750',Current_year:''}),

(c7:Customers{Customer_id:'4007',Phone_no:'2312564578',Book_id:'98751',Current_year:''}),

(c8:Customers{Customer_id:'4008',Phone_no:'1223455689',Book_id:'98752',Current_year:''}),

(c9:Customers{Customer_id:'4009',Phone_no:'4556122378',Book_id:'98753',Current_year:''}),

(c10:Customers{Customer_id:'4010',Phone_no:'5645122389',Book_id:'98754',Current_year:''}),

(c11:Customers{Customer_id:'4011',Phone_no:'7889455612',Book_id:'18745',Current_year:''}),

(c12:Customers{Customer_id:'4012',Phone_no:'8978455612',Book_id:'18746',Current_year:''}),

(c13:Customers{Customer_id:'4013',Phone_no:'1599517535',Book_id:'18747',Current_year:''}),

(c14:Customers{Customer_id:'4014',Phone_no:'7894561237',Book_id:'18748',Current_year:''}),

(c15:Customers{Customer_id:'4015',Phone_no:'8945612378',Book_id:'18749',Current_year:''}),

(c16:Customers{Customer_id:'4016',Phone_no:'9456123789',Book_id:'18750',Current_year:''}),

(c17:Customers{Customer_id:'4017',Phone_no:'4561237891',Book_id:'18751',Current_year:''}),

(c18:Customers{Customer_id:'4018',Phone_no:'5612378945',Book_id:'18752',Current_year:''}),

(c19:Customers{Customer_id:'4019',Phone_no:'6123789452',Book_id:'18753',Current_year:''}),

(c20:Customers{Customer_id:'4020',Phone_no:'1237894563',Book_id:'18754',Current_year:''})


RETURN *



CREATE CONSTRAINT ON (c:Customers) ASSERT c.Customer_id IS UNIQUE

CREATE CONSTRAINT ON (c:Customers) ASSERT exists(c.Customer_id)


CREATE

(so1:Shop_owner{book_id:'18745',book_name:'bs_grewal',source:'first_hand',selling_price:'500',cost_price:'400'}),

(so2:Shop_owner{book_id:'18746',book_name:'Discrete_Mathematics_and_its_Applications',source:'first_hand',selling_price:'600',cost_price:'450'}),

(so3:Shop_owner{book_id:'18747',book_name:'Introduction_to_Algorithms',source:'first_hand',selling_price:'400',cost_price:'299'}),

(so4:Shop_owner{book_id:'18748',book_name:'Let_Us_C',source:'first_hand',selling_price:'300',cost_price:'299'}),

(so5:Shop_owner{book_id:'18749',book_name:'Let_Us_C++',source:'first_hand',selling_price:'300',cost_price:'200'}),

(so6:Shop_owner{book_id:'18750',book_name:'Introduction_to_the_Theory_of_Computation',source:'first_hand',selling_price:'700',cost_price:'600'}),

(so7:Shop_owner{book_id:'18751',book_name:'Hacking:_The_Art_of_Exploitation',source:'first_hand',selling_price:'999',cost_price:'799'}),

(so8:Shop_owner{book_id:'18752',book_name:'Code_Complete',source:'first_hand',selling_price:'999',cost_price:'599'}),
```

(so9:Shop_owner{book_id:'18753',book_name:'Introduction_to_Python',source:'first_hand',selling_price:'700',cost_price:'499'}),

(so10:Shop_owner{book_id:'18754',book_name:'Introduction_to_Linear_Algebra',source:'first_hand',selling_price:'700',cost_price:'299'}),

(so11:Shop_owner{book_id:'98745',book_name:'bs_grewal',source:'second_hand',selling_price:'300',cost_price:'200'}),

(so12:Shop_owner{book_id:'98746',book_name:'Discrete_Mathematics_and_its_Applications',source:'second_hand',selling_price:'300',cost_price:'150'}),

(so13:Shop_owner{book_id:'98747',book_name:'Introduction_to_Algorithms',source:'second_hand',selling_price:'299',cost_price:'100'}),

(so14:Shop_owner{book_id:'98748',book_name:'Let_Us_C',source:'second_hand',selling_price:'150',cost_price:'100'}),

(so15:Shop_owner{book_id:'98749',book_name:'Let_Us_C++',source:'second_hand',selling_price:'200',cost_price:'100'}),

(so16:Shop_owner{book_id:'98750',book_name:'Introduction_to_the_Theory_of_Computation',source:'second_hand',selling_price:'400',cost_price:'300'}),

(so17:Shop_owner{book_id:'98751',book_name:'Hacking:_The_Art_of_Exploitation',source:'second_hand',selling_price:'400',cost_price:'299'}),

(so18:Shop_owner{book_id:'98752',book_name:'Code_Complete',source:'second_hand',selling_price:'500',cost_price:'299'}),

(so19:Shop_owner{book_id:'98753',book_name:'Introduction_to_Python',source:'second_hand',selling_price:'199',cost_price:'100'}),

(so20:Shop_owner{book_id:'98754',book_name:'Introduction_to_Linear_Algebra',source:'second_hand',selling_price:'199',cost_price:'100'})

RETURN *

CREATE CONSTRAINT ON (so:Shop_owner) ASSERT so.book_id IS UNIQUE

CREATE CONSTRAINT ON (so:Shop_owner) ASSERT exists(so.book_id)

add no of quantity in the relation.

CREATE

(ws1:Whole_seller{Selling_id:'201', Book_id: '18745' , Publication : 'Khanna' , Manufacture_price : '200' , Selling_price : '400'}),

(ws2:Whole_seller{Selling_id:'202', Book_id: '18746' , Publication : 'Kenneth_Rosen' , Manufacture_price : '200' , Selling_price : '450'}),

(ws3:Whole_seller{Selling_id:'203', Book_id: '18747' , Publication : 'MIT_press' , Manufacture_price : '150' , Selling_price : '299'}),

(ws4:Whole_seller{Selling_id:'204', Book_id: '18748' , Publication : 'GPB' , Manufacture_price : '99' , Selling_price : '299'}),

(ws5:Whole_seller{Selling_id:'205', Book_id: '18749' , Publication : 'GPB' , Manufacture_price : '99' , Selling_price : '200'}),

(ws6:Whole_seller{Selling_id:'206', Book_id: '18750' , Publication : 'Michael' , Manufacture_price : '299' , Selling_price : '600'}),

(ws7:Whole_seller{Selling_id:'207', Book_id: '18751' , Publication : 'ABC' , Manufacture_price : '450' , Selling_price : '799'}),

(ws8:Whole_seller{Selling_id:'208', Book_id: '18752' , Publication : 'Microsoft_press' , Manufacture_price : '300' , Selling_price : '599'}),

(ws9:Whole_seller{Selling_id:'209', Book_id: '18753' , Publication : 'EFG' , Manufacture_price : '150' , Selling_price : '399'}),

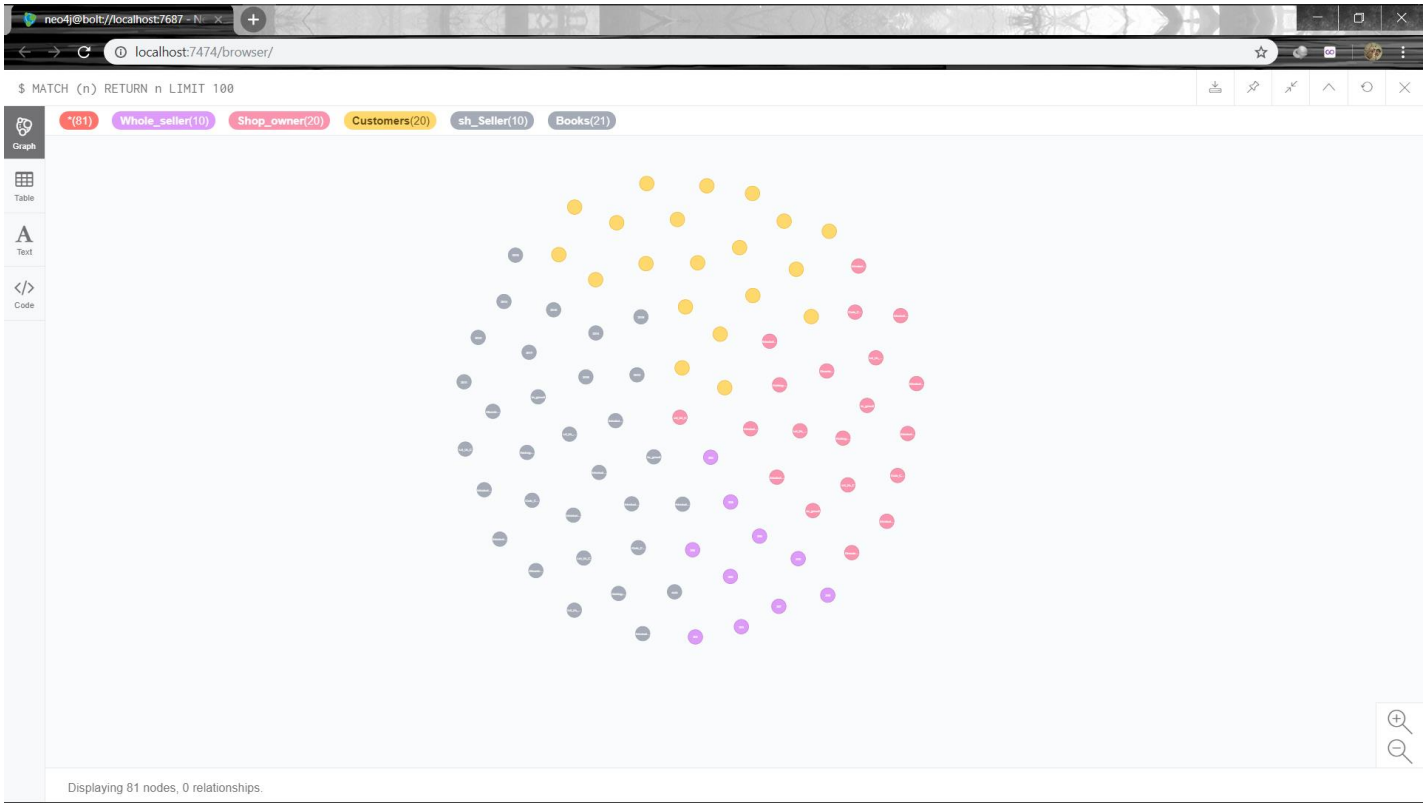
(ws10:Whole_seller{Selling_id:'210', Book_id: '18754' , Publication : 'Gilbert' , Manufacture_price : '200' , Selling_price : '499'})

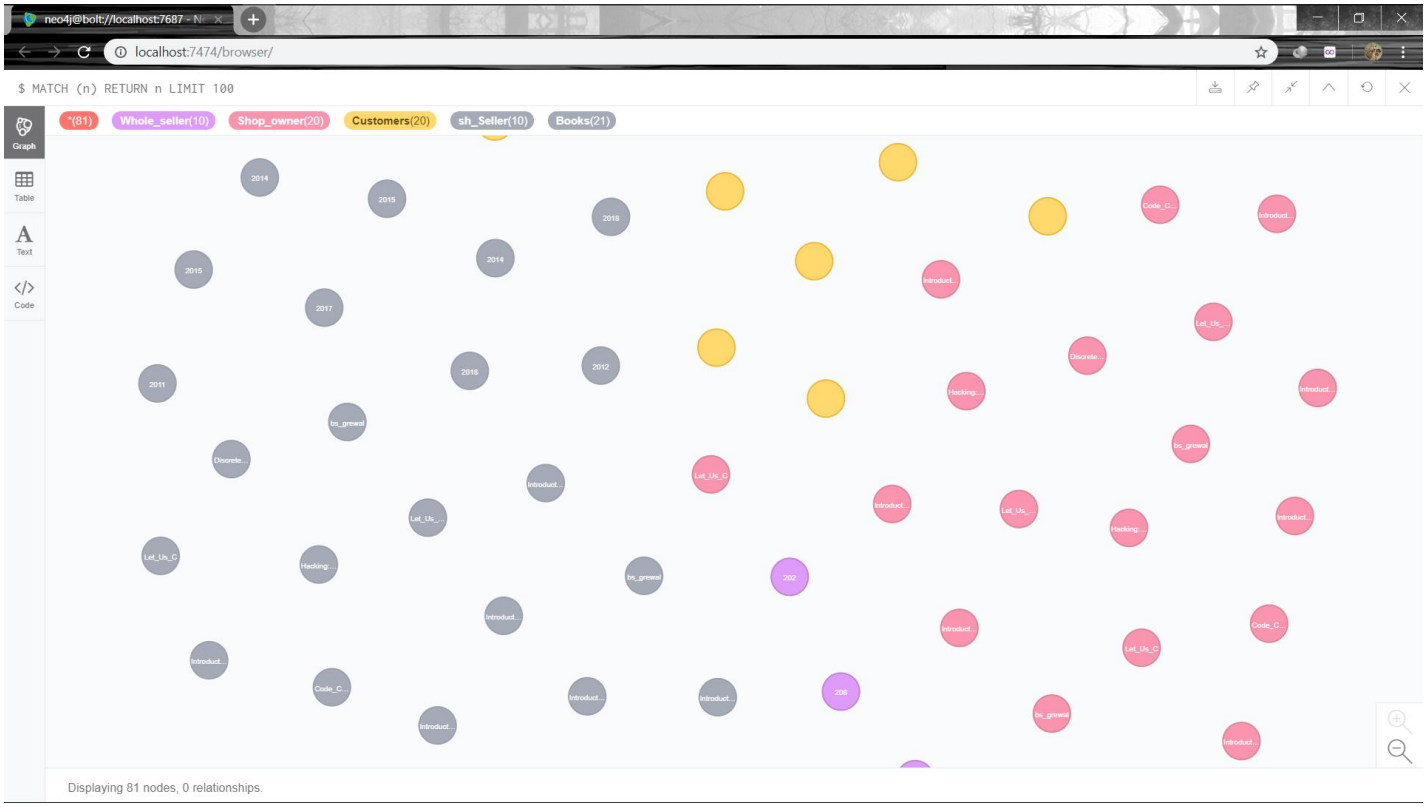
RETURN *

CREATE CONSTRAINT ON (ws:Whole_seller) ASSERT ws.Selling_id IS UNIQUE

CREATE CONSTRAINT ON (ws:Whole_seller) ASSERT exists(ws.Selling_id)

DATABASE SCREENSHOTS:





PYTHON APP CODE:

```
from py2neo import *
import json
from pandas import *

graph = Graph("http://neo4j:12345@localhost:7474")

def books():
        print("\n 1.Add \n2.Delete \n3.Search \n4.Show All \n5.Go Back")
        chc = int(input())
        if(chc == 1):
                cpr = (input("Enter Current Price: "))
                bid = int(input("Enter Book ID: "))
                bnm = input("Enter Book Name: ")
                nrs = int(input("Enter No Of Resales: "))
                spr = int(input("Enter Shopkeeper Price: "))
                bk = Node("Books", book_id = bid, book_name = bnm, no_of_resale = nrs, shopowner_price = spr, current_price =
cpr)
                graph.create(bk)
                books()
        if(chc == 2):
                dch = int(input("\n1. Book Name \n2.Book ID \nSelect criteria to delete: "))
```

```
		if(dch == 1):
				bnm = input("Enter Book Name")
				matcher = NodeMatcher(graph)
				res = matcher.match("Books",book_name = bnm).first()
				graph.delete(res)
		if(dch == 2):
				bid = input("Enter Book ID")
				matcher = NodeMatcher(graph)
				res = matcher.match("Books",book_id = bid).first()
				graph.delete(res)
		books()
	if(chc == 3):
		dch = int(input("\n1. Book Name \n2.Book ID \nSelect criteria to seaarch: "))
		if(dch == 1):
				bnm = input("Enter Book Name")
				matcher = NodeMatcher(graph)
				res = list(matcher.match("Books",book_name = bnm))
				print(DataFrame(res))
		if(dch == 2):
				bid = input("Enter Book ID")
				matcher = NodeMatcher(graph)
				res = list(matcher.match("Books",book_id = bid))
				print(DataFrame(res))
		books()
	if(chc == 4):
		gdata = graph.run("MATCH (n:Books) RETURN n.current_price AS CURRENT_PRICE,n.book_id AS
BOOK_ID,n.book_name AS BOOK_NAME,n.no_of_resale AS NO_OF_RESALES,n.shopowner_price AS SHOPOWNER_PRICE LIMIT
100").to_table()
		print(gdata)
		books()
	if(chc == 5):
		main();



def cust():
	print("\n 1.Add \n2.Delete \n3.Search \n4.Show All \n5.Go Back")
	chc = int(input())
	if(chc == 1):
		cyr = (input("Enter Current Year: "))
		bid = int(input("Enter Book ID: "))
		cid = input("Enter Customer ID: ")
```

```
            pno = int(input("Enter Phone Number: "))
            bk = Node("Customers", Customer_id = cid,Phone_no=pno,Book_id=bid,Current_year=cyr)
            graph.create(bk)
            cust()
        if(chc == 2):
            dch = int(input("\n1. Customer ID \n2.Phone Number  \nSelect criteria to delete: "))
            if(dch == 1):
                        cid = input("Enter Customer ID")
                        matcher = NodeMatcher(graph)
                        res = matcher.match("Customers",Customer_id = cid).first()
                        graph.delete(res)
            if(dch == 2):
                        pno = input("Enter Phone number")
                        matcher = NodeMatcher(graph)
                        res = matcher.match("Customers",Phone_no = pno).first()
                        graph.delete(res)
            books()
        if(chc == 3):
            dch = int(input("\n1. Customer ID \n2.Phone Number \nSelect criteria to seaarch: "))
            if(dch == 1):
                        cid = input("Enter Customer ID")
                        matcher = NodeMatcher(graph)
                        res = list(matcher.match("Customers",Customer_id = cid))
                        print(DataFrame(res))
            if(dch == 2):
                        pno = input("Enter Book ID")
                        matcher = NodeMatcher(graph)
                        res = list(matcher.match("Customers",Phone_no = pno))
                        print(DataFrame(res))
            cust()
        if(chc == 4):
            gdata = graph.run("MATCH (n:Customers) RETURN n.Customer_id AS CUSTOMER_ID,n.Book_id AS
BOOK_ID,n.Current_year AS CURRENT_YEAR,n.Phone_no AS PHONE_NO LIMIT 100").to_table()
            print(gdata)
            books()
        if(chc == 5):
            main();



def shpo():
```

```
print("\n 1.Add \n2.Delete \n3.Search \n4.Show All \n5.Go Back")
chc = int(input())
if(chc == 1):
        bsp = (input("Enter Selling Price: "))
        fsh = int(input("Enter Source: "))
        bid = input("Enter Book ID: ")
        bnm = int(input("Enter Book Name: "))
        bcp = int(input("Enter Cost Price: "))
        bk = Node("Shop_owner", selling_price=bsp,source=fsh,book_id=bid,book_name = bnm,cost_price=bcp)
        graph.create(bk)
        shpo()
if(chc == 2):
        dch = int(input("\n1. Book Name \n2.Book ID \nSelect criteria to delete: "))
        if(dch == 1):
                bnm = input("Enter Book Name")
                matcher = NodeMatcher(graph)
                res = matcher.match("Shop_owner",book_name = bnm).first()
                graph.delete(res)
        if(dch == 2):
                bid = input("Enter Book ID")
                matcher = NodeMatcher(graph)
                res = matcher.match("Shop_owner",book_id = bid).first()
                graph.delete(res)
        shpo()
if(chc == 3):
        dch = int(input("\n1. Book Name \n2.Book ID \nSelect criteria to seaarch: "))
        if(dch == 1):
                bnm = input("Enter Book Name")
                matcher = NodeMatcher(graph)
                res = list(matcher.match("Shop_owner",book_name = bnm))
                print(DataFrame(res))
        if(dch == 2):
                bid = input("Enter Book ID")
                matcher = NodeMatcher(graph)
                res = list(matcher.match("Shop_owner",book_id = bid))
                print(DataFrame(res))
        shpo()
if(chc == 4):
        gdata = graph.run("MATCH (n:Shop_owner) RETURN n.selling_price AS Selling_Price,n.book_id AS
BOOK_ID,n.book_name AS BOOK_NAME,n.source AS Source,n.cost_price AS Cost_Price LIMIT 100").to_table()
```

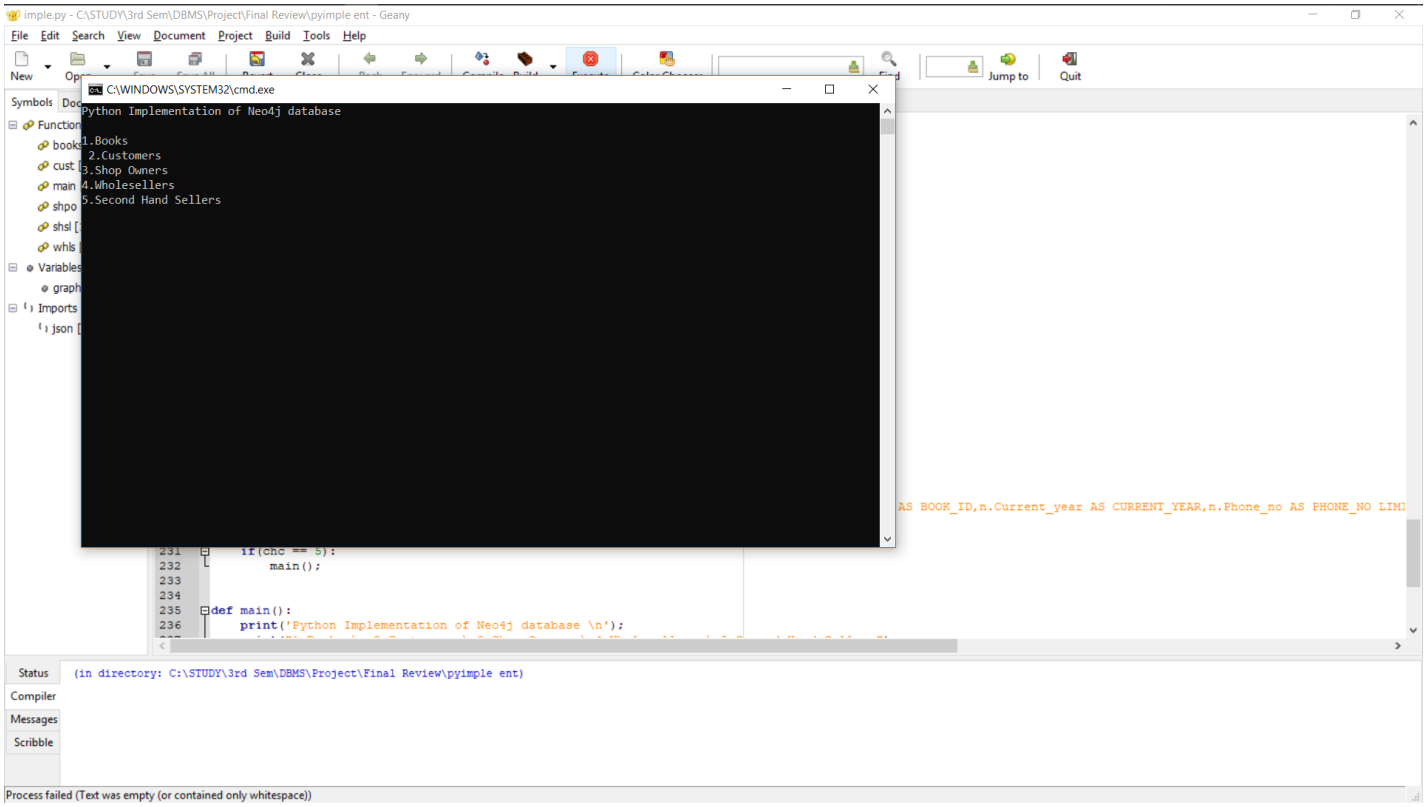
```
                print(gdata)
                shpo()
        if(chc == 5):
                main();



def whls():
        print("\n 1.Add \n2.Delete \n3.Search \n4.Show All \n5.Go Back")
        chc = int(input())
        if(chc == 1):
                sid = (input("Enter Selling ID: "))
                pub = int(input("Enter Publication: "))
                bid = input("Enter Book ID: ")
                bsp = int(input("Enter Selling Price: "))
                bmp = int(input("Enter Manufacture Price: "))
                bk = Node("Whole_seller", selling_price=bsp,Publication=pub,book_id=bid,Selling_id = sid, Manufacture_price =
bmp)
                graph.create(bk)
                whls()
        if(chc == 2):
                dch = int(input("\n1. Publication \n2.Book ID \nSelect criteria to delete: "))
                if(dch == 1):
                                pub = input("Enter Publication: ")
                                matcher = NodeMatcher(graph)
                                res = matcher.match("Whole_seller",Publication  = pub).first()
                                graph.delete(res)
                if(dch == 2):
                                bid = input("Enter Book ID: ")
                                matcher = NodeMatcher(graph)
                                res = matcher.match("Whole_seller",Book_id = bid).first()
                                graph.delete(res)
                whls()
        if(chc == 3):
                dch = int(input("\n1. Publisher  \n2.Book ID \nSelect criteria to search: "))
                if(dch == 1):
                                pub = input("Enter Publisher: ")
                                matcher = NodeMatcher(graph)
                                res = list(matcher.match("Whole_seller",Publication = pub))
                                print(DataFrame(res))
                if(dch == 2):
```

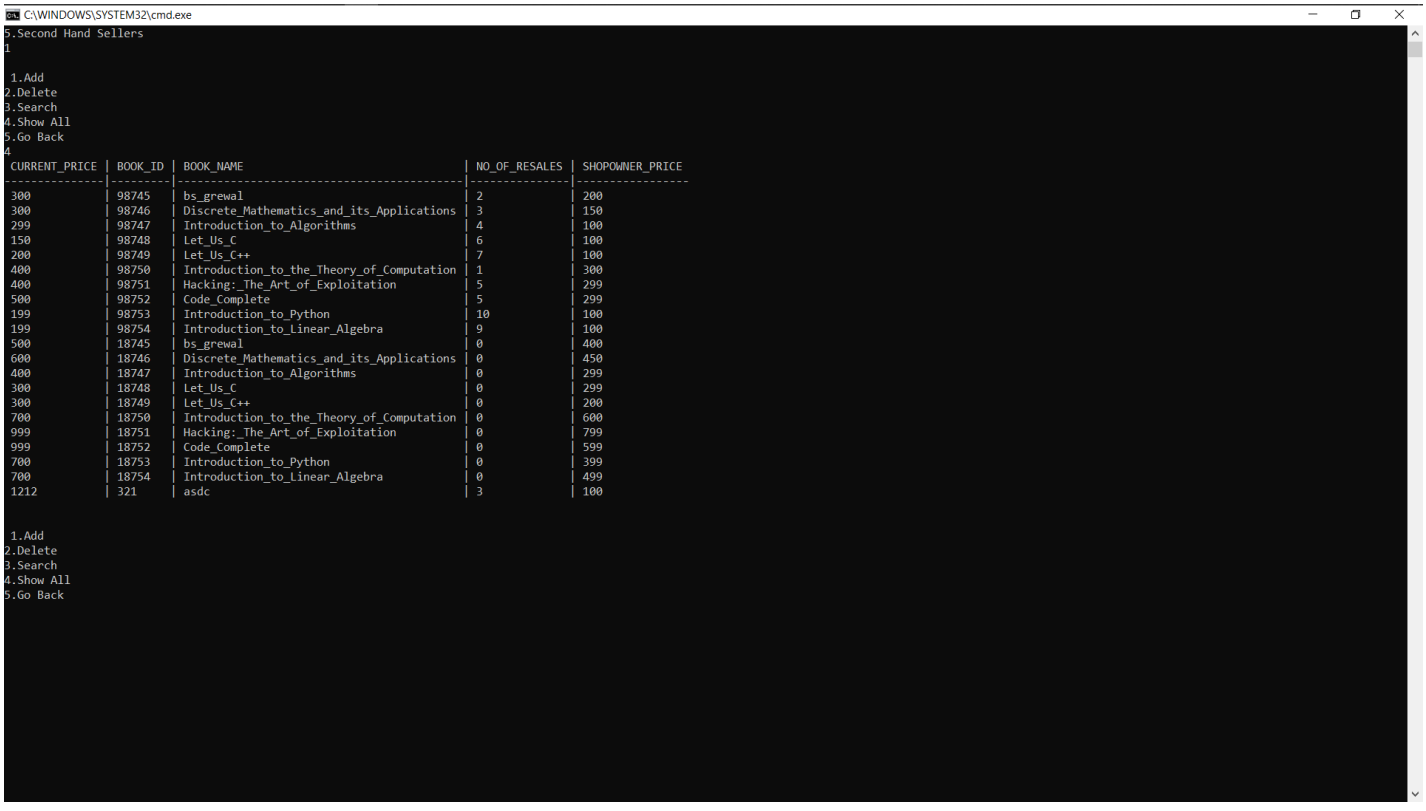
```
                bid = input("Enter Book ID")
                matcher = NodeMatcher(graph)
                res = list(matcher.match("Whole_seller",Book_id = bid))
                print(DataFrame(res))
        whls();
    if(chc == 4):
        gdata = graph.run("MATCH (n:Whole_seller) RETURN n.selling_price AS Selling_Price,n.Book_id AS
BOOK_ID,n.Selling_id AS SELLING_ID,n.Publication AS PUBLICATION,n.Manufacture_price AS MANUFACTURE_PRICE LIMIT
100").to_table()
        print(gdata)
        whls()
    if(chc == 5):
        main();



def shsl():
    print("\n 1.Add \n2.Delete \n3.Search \n4.Show All \n5.Go Back")
    chc = int(input())
    if(chc == 1):
        cyr = (input("Enter Current Year: "))
        bid = int(input("Enter Book ID: "))
        cid = input("Enter Customer ID: ")
        pno = int(input("Enter Phone Number: "))
        bk = Node("sh_Seller", Customer_id = cid,Phone_no=pno,Book_id=bid,Current_year=cyr)
        graph.create(bk)
        shsl()
    if(chc == 2):
        dch = int(input("\n1. Customer ID \n2.Phone Number  \nSelect criteria to delete: "))
        if(dch == 1):
                cid = input("Enter Customer ID")
                matcher = NodeMatcher(graph)
                res = matcher.match("sh_Seller",Customer_id = cid).first()
                graph.delete(res)
        if(dch == 2):
                pno = input("Enter Phone number")
                matcher = NodeMatcher(graph)
                res = matcher.match("sh_Seller",Phone_no = pno).first()
                graph.delete(res)
        shsl()
    if(chc == 3):
        dch = int(input("\n1.  Customer ID \n2.Phone Number \nSelect criteria to seaarch: "))
```

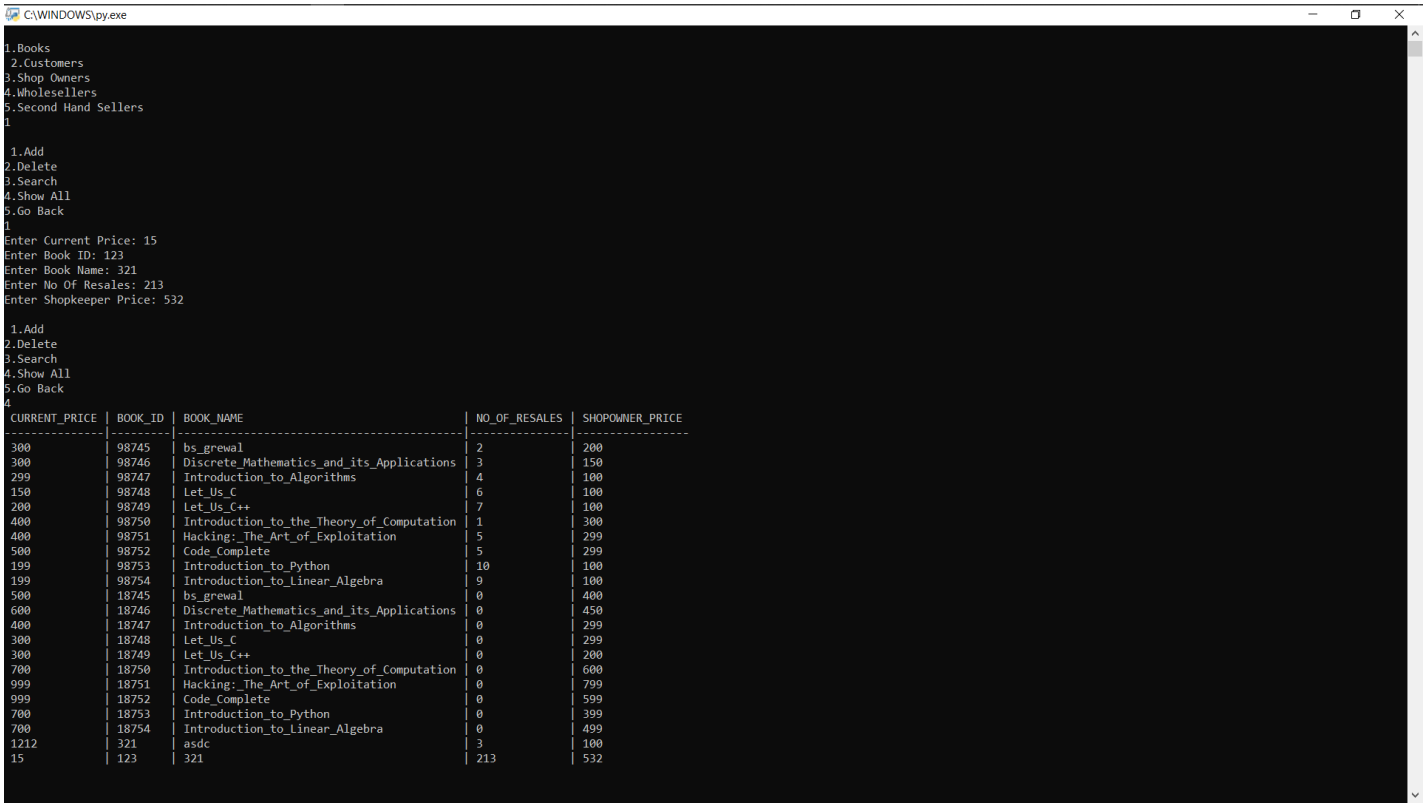
```
                if(dch == 1):
                                cid = input("Enter Customer ID")
                                matcher = NodeMatcher(graph)
                                res = list(matcher.match("sh_Seller",Customer_id = cid))
                                print(DataFrame(res))
                if(dch == 2):
                                pno = input("Enter Phone Number")
                                matcher = NodeMatcher(graph)
                                res = list(matcher.match("sh_Seller",Phone_no = pno))
                                print(DataFrame(res))
                shsl()
        if(chc == 4):
                gdata = graph.run("MATCH (n:sh_Seller) RETURN n.Customer_id AS CUSTOMER_ID,n.Book_id AS
BOOK_ID,n.Current_year AS CURRENT_YEAR,n.Phone_no AS PHONE_NO LIMIT 100").to_table()
                print(gdata)
                shsl()
        if(chc == 5):
                main();



def main():
        print('Python Implementation of Neo4j database \n');
        print("1.Books \n 2.Customers \n3.Shop Owners \n4.Wholesellers \n5.Second Hand Sellers")
        x = int(input())

        if(x==1):
                books()
        if(x==2):
                cust()
        if(x==3):
                shpo()
        if(x==4):
                whls()
        if(x==5):
                shsl()




main()
```

PYTHON APP SCREENSHOTS:

IDLE USED: GEANY







Advantage of NoSQL :

LESS NEED FOR ETL

NoSQL databases support storing data “as is.” Key value stores give you the ability to store simple data structures, whereas document NoSQL databases provide you with the ability to handle a range of flat or nested structures.

Most of the data flying between systems does so as a message. Typically, the data takes one of these formats:

A binary object to be passed through a set of layers

An XML document

A JSON document

Being able to handle these formats natively in a range of NoSQL databases lessens the amount of code you have to convert from the source data format to the format that needs storing. This is called extract, transform, and load (ETL).

Using this approach, you greatly reduce the amount of code required to start using a NoSQL database. Moreover, because you don’t have to pay for updates to this “plumbing” code, ongoing maintenance costs are significantly decreased.

SUPPORT FOR UNSTRUCTURED TEXT

The vast majority of data in enterprise systems is unstructured. Many NoSQL databases can handle indexing of unstructured text either as a native feature (MarkLogic Server) or an integrated set of services including Solr or Elasticsearch.

Being able to manage unstructured text greatly increases information and can help organizations make better decisions. For example, advanced uses include support for multiple languages with facetted search, snippet functionality, and word-stemming support. Advanced features also include support for dictionaries and thesauri.

Furthermore, using search alert actions on data ingest, you can extract named entities from directories such as those listing people, places, and organizations, which allows text data to be better categorized, tagged, and searched.

Entity enrichment services such as SmartLogic, OpenCalais, NetOwl, and TEMIS Luxid that combine extracted information with other information provide a rich interleaved information web and enhance efficient analysis and use.

ABILITY TO HANDLE CHANGE OVER TIME

Because of the schema agnostic nature of NoSQL databases, they’re very capable of managing change — you don’t have to rewrite ETL routines if the XML message structure between systems changes.

Some NoSQL databases take this a step further and provide a universal index for the structure, values, and text found in information. Microsoft DocumentDB and MarkLogic Server both provide this capability.

If a document structure changes, these indexes allow organizations to use the information immediately, rather than having to wait for several months before you can test and rewrite systems.

NO RELIANCE ON SQL MAGIC

Structured Query Language (SQL) is the predominant language used to query relational database management systems. Being able to structure queries so that they perform well has over the years become a thorny art. Complex multitable joins are not easy to write from memory.

Although several NoSQL databases support SQL access, they do so for compatibility with existing applications such as business intelligence (BI) tools. NoSQL databases support their own access languages that can interpret the data being stored, rather than require a relational model within the underlying database.

This more developer-centric mentality to the design of databases and their access application programming interfaces (API) are the reason NoSQL databases have become very popular among application developers.

Application developers don't need to know the inner workings and vagaries of databases before using them. NoSQL databases empower developers to work on what is required in the applications instead of trying to force relational databases to do what is required.

ABILITY TO SCALE HORIZONTALLY ON COMMODITY HARDWARE

NoSQL databases handle partitioning (sharding) of a database across several servers. So, if your data storage requirements grow too much, you can continue to add inexpensive servers and connect them to your database cluster (horizontal scaling) making them work as a single data service.

Contrast this to the relational database world where you need to buy new, more powerful and thus more expensive hardware to scale up (vertical scaling). If you were to double the amount of data you store, you would easily quadruple the cost of the hardware you need.

Providing durability and high availability of a NoSQL database by using inexpensive hardware and storage is one of NoSQL's major assets. Being able to do so while providing generous scalability for many uses also doesn't hurt!

BREADTH OF FUNCTIONALITY

Most relational databases support the same features but in a slightly different way, so they are all similar.

NoSQL databases, in contrast, come in four core types: key-value, columnar, document, and triple stores. Within these types, you can find a database to suit your particular (and peculiar!) needs. With so much choice, you're bound to find a NoSQL database that will solve your application woes.

SUPPORT FOR MULTIPLE DATA STRUCTURES

Many applications need simple object storage, whereas others require highly complex and interrelated structure storage. NoSQL databases provide support for a range of data structures.

Simple binary values, lists, maps, and strings can be handled at high speed in key-value stores.

Related information values can be grouped in column families within Bigtable clones.

Highly complex parent-child hierarchal structures can be managed within document databases.

A web of interrelated information can be described flexibly and related in triple and graph stores.

VENDOR CHOICE

The NoSQL industry is awash with databases, though many have been around for less than ten years. For example, IBM, Microsoft, and Oracle only recently dipped their toes into this market. Consequently, many vendors are targeting particular audiences with their own brew of innovation.

Open-source variants are available for most NoSQL databases, which enables companies to explore and start using NoSQL databases at minimal risk. These companies can then take their new methods to a production platform by using enterprise offerings.

NO LEGACY CODE

Because they are so new, NoSQL databases don't have legacy code, which means they don't need to provide support for old hardware platforms or keep strange and infrequently used functionality updated.

NoSQL databases enjoy a quick pace in terms of development and maturation. New features are released all the time, and new and existing features are updated frequently (so NoSQL vendors don't need to maintain a very large code base). In fact, new major releases occur annually rather than every three to five years.

EXECUTING CODE NEXT TO THE DATA

NoSQL databases were created in the era of Hadoop. Hadoop's highly distributed filesystem (HDFS) and batch-processing environment (Map/Reduce) signaled changes in the way data is stored, queried, and processed.

Queries and processing work now pass to several servers, which provides high levels of parallelization for both ingest and query workloads. Being able to calculate aggregations next to the data has also become the norm.

You no longer need a separate data warehouse system that is updated overnight. With fast aggregations and query handling, analysis is passed to the database for execution next to the data, which means you don't have to ship a lot of data around a network to achieve locally combined analysis.

Advantage of Neo4j :

Highly scalable: Neo4j is highly scalable. It provides a simple, powerful and flexible data model which can be changed according to applications and uses. It provides:

Higher vertical scaling.

Improved operational characteristics at scale.

Higher concurrency.

Simplified tuning.

Schema-free: Neo4j is schema-free like other NoSQL databases.

High availability: Neo4j provides high availability for large enterprise real-time applications with transactional guarantees.

Real-time data analysis: Neo4j provides results based on real-time data.

Easy representation: Neo4j provides a very easy way to represent connected and semi-structured data.

Fast Execution: Neo4j is fast because more connected data is very easy to retrieve and navigate.

Easy retrieval: Neo4j facilitates you not only represent but also easily retrieve (traverse/navigate) connected data faster other databases comparatively.

Cypher Query language: Neo4j provides CQL (Cypher Query Language) a declarative query language to represent the graph visually, using ASCII-art syntax. The commands of this language is very easy to learn and human readable.

No Join: Neo4j doesn't require complex Joins to retrieve connected/related data as it is very easy to retrieve its adjacent node or relationship details without Joins or Indexes because it is a graph database and all nodes are already connected.