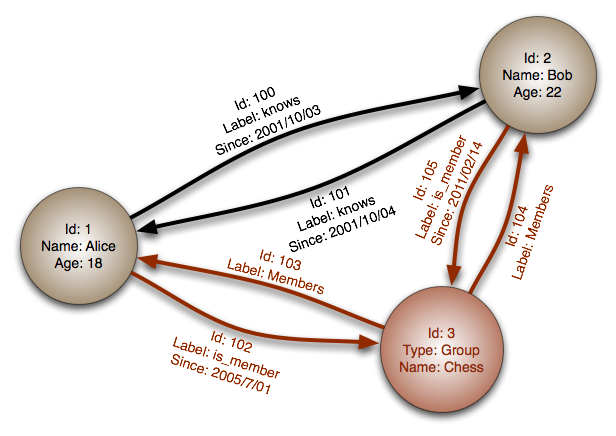
GRAPH DATABASE

Graph database, matematikte [graph](http://en.wikipedia.org/wiki/Graph_(mathematics)) [theory](http://en.wikipedia.org/wiki/Graph_(mathematics))olarak geçen yapıyı birebir model alan veri tabanıdır. Nesneler tablolarda değilde graph’larda tutuluyor. Geleneksel tablo yapısı kullanılmaz, bunun yerine nesne tabanlı bir ağ yapısı kullanılır. Yüksek transaction ihtiyacını kolaylıkla karşılamaktadır. Grafik veritabanı kullanan grafik yapıları anlamsal sorgu düğümleri,kenarlar ve özellikler bulundurabilir. Bir grafik veritabanı endeksi ücretsiz komşuluklarını sağlayan herhangi bir depolama sistemidir. Bir kayıt node yada edge olabiliyor. Edge’ler nodelar arasında yönleri (birectional) nesneleri tutuyor. Öğrenci sınıf ilişkisinde 2 node(öğrenci, sınıf), 2 edge (“X sınıfına gidiyor”, “Y benim öğrencim”) var. Graph veritabanı nesneler arasındaki bağlantıların (relation) en az nesneler kadar önem taşıdığı modeller için tasarlanmıştır.

**YAPI:**Grafik veritabanlarının dayalı olduğu grafik teorisi



**KULLANIM ALANLARI:**

 Grafik veritabanları genellikle hızlı ilişkisel veri setleri içindir. Facebook, Linkedin gibi milyon kaydın birbiriyle ilişkisini en hızlı sorgulamaya çalışan sistemleri aday sistemler olarak düşünebiliriz.

**GRAFİK VERİ TABANI PROJELERİ:**

Aşağıdaki birçok tanınmış grafik veritabanı projelerinin bir listesi:

| **Isim** | **Versiyon** | **Lisans** | **Dil** | **Tanım** |
| --- | --- | --- | --- | --- |
| AllegroGraph | 4.14.1 (Eylül 2014) | Tescilli , v1 Eclipse | C# , C ,Common Lisp ,Java ,Python | Bir RDF ve grafik veri tabanı. |
| ArangoDB | 2.4.4 (Şubat 2015) | Apache 2 | C, C++ ve JavaScript | Bir dağıtılmış çoklu-model belge deposu ve grafik veri tabanı.Son derece ölçeklenebilir. |
| Blazegraph | 1.5.1 (Mart 2015) | GPLv2 , ticari lisans | Java | Kümelenmiş dağıtım yapabilen bir RDF / grafik veri tabanı.Blazegraph yüksek kullanılabilirlik (HA) modunu gömülü modu, tek sunucu modunu destekler. |
| Bitsy | 1.5.0 | AGPL | Java | Küçük gömülebilir dayanıklı bellek grafik veritabanıdır. |
| BrightstarDB |  | MİT Lisansı | C # | .NET platformu için gömülebilir NoSQL veritabanı. |
| Cayley | 0.4.0 (Ağustos 2014) | Apache 2 | Git | Freebase ve Google'ın Bilgi Grafiği arkasındaki grafik veritabanıdır. |
| DEX / Sparksee | 5.1.0 (2014) | (ücretsiz) ticari kullanım | C ++ | Sparsity Technologies, DAMA-UPC bir teknoloji geçiş şirketinden yüksek performanslı ve ölçeklenebilir grafik veri tabanı yönetim sistemi.  Sparksee 5 mobil cihazlar için ilk grafik veritabanıdır. |
| Filaman |  | BSD | Java | Bir grafik kalıcılık çerçeve ve bir navigasyon sorgu tarzı dayalı ilişkili araçlarında kullanılır. |
| GraphBase | 1.0.03a | Özel | Java | FactNexus kümesi zengin bir araç ile özelleştirilebilir, dağıtılmış, az yer kaplayan grafik mağazasıdır. |
| Horton |  | Özel | C # | Microsoft Research Extreme Computing Group (XCG) A grafik veritabanı bulut programlama altyapısıdır. Orleans dayalıdır. |
| HyperGraph Database | 1,2 (2012) | LGPL | Java | Genelleştirilmiş destekleyen bir grafik veritabanıdır. Hypergraphs kenarları, diğer kenarlara işaret edebilir. |
| IBM System Mağaza | v1.0 (Temmuz 2014) | Özel | C , [C ++](http://en.wikipedia.org/wiki/C%2B%2B) ,[Java](http://en.wikipedia.org/wiki/Java_(programming_language)) | Doğal hayata grafik veri yapıları ve üstün verimlilik elde etmek için ilkel kullanarak yüksek performanslı grafik mağaza.  ve RDF grafikler, işleyebilir. |
| InfiniteGraph | 3,0 (Ocak 2013) | Özel | Java | Esnek lisanslama ile dağıtılan ve bulut özellikli ticari üründür. |
| InfoGrid | 2.9.5 (2011) | AGPLv3 küçük kişiler için ücretsiz | Java | Web ön uç ve yapılandırılabilir depolama motorları (MySQL, PostgreSQL, Dosyalar, Hadoop) ile bir grafik veri tabanı. |
| MapGraph | v3 (2014) | Apache 2 | CUDA | MapGraph Devasa GPU'lar üzerinde paralel Grafik işleme sağlar. MapGraph API GPU'lar yüksek performanslı grafik analitik geliştirmek için kullanılır. |
| Neo4j | 2.2.0 (Mart 2015) | GPLv3 Community Edition. İşletme ve İleri sürümleri için Ticari & AGPLv3 seçenekleri | Java | Destekleyen bir yüksek ölçeklenebilir açık kaynak grafik veritabanıdır. Tam işlem desteği ve görsel düğüm bağlantı grafik kaşif içeren web tabanlı bir yönetim aracı ile birlikte geliyor. Neo4j kullanarak birçok programlama dilleri erişilebilir. Neo4j kullanımda bugün en popüler grafik veritabanları arasındadır. |
| Ontotext GraphDB | 6.1 | GraphDB Lite ücretsiz  GraphDB Standardı ve GraphDB | Java | W3C Semantik Web standartlarına tamamen dayalı bir grafik veritabanı motoru: RDF, RDFS, OWL, SPARQL. GraphDB Lite "bellekte" motorudur. GraphDB Standart sağlam bağımsız bir veritabanı motorudur. |
| Orly | (Mart 2014) | Apache 2 | C ++ | Bir yüksek ölçeklenebilir açık kaynak grafik veri tabanı. Orly kendi yerleşik kullanarak programlama dillerinin çoğunu erişilebilir. |
| OpenLink [Virtuoso](http://en.wikipedia.org/wiki/Virtuoso_Universal_Server) | 7.1 (Mart 2014) | GPLv2 Open Source Edition için. TescilliEnterprise Edition için. | C, C ++ | RDF ve diğer grafik verilerini, RDB / SQL veri, XML verilerini, dosya belgeleri / nesneleri ve serbest metin işleme bir melez veritabanı sunucusu. Yerel gömülü örneği (kullanılan olarak konuşlanmış olabilir. |
| Oracle Spatial ve Grafik | 11.2 (2012) | Özel | Java, PL / SQL | 1) RDF Semantik Grafik: yerli akıl ve üçlü düzey etiket güvenlik Oracle Veritabanı kapsamlı W3C RDF grafiği yönetimi. 2) Ağ Veri Modeli özelliği grafiği: kalıcı depolama ve bellek grafiği analiz için bir Java API ile fiziksel / mantıksal ağlar için. |
| Oracle Database NoSQL | 2.0.39 (2013) | Özel | Java | Oracle Database NoSQL için RDF Grafik NoSQL Veritabanı W3C RDF grafiği yetenekleri sağlayan Enterprise Edition bir özelliğidir. |
| OrientDB | 2.0.1 (Ocak 2015) | Community Edition , Apache 2Enterprise Edition | Java | OrientDB bir Açık Kaynak ticari dost lisans (Apache 2 lisans) ile tek bir üründe Belgelerin esneklik Grafik Veritabanı çoğaltma ve Sharding vardır |
| sones GraphDB |  | AGPLv3 | C # | (Tarafından finanse Bir grafik veri tabanı ve evrensel erişim katmanı ) Sones Ocak 2012'de iflasını ilan etti ve GraphDB marka Ontotext tarafından satın alınmıştır. |
| SPARQLCity | v1.0.95 (Ekim 2014) | GPLv3 | C , C++  ve JavaScript | Standartları ve yapılandırılmış ve yarı yapılandırılmış veriler üzerinde zengin iş analitiği gerçekleştirmek için Hadoop tabanlı analitik grafik motoru: SPARQLCity SPARQLVerse üretir. |
| Sqrrl Kurumsal | v1.5.1 (Ağustos 2014) | Özel | Java | Dağıtılmış, gerçek zamanlı grafik veritabanı hücre düzeyinde güvenliği ve masif ölçeklenebilirlik sunan. |
| Stardog | v2.2 (Temmuz 2014) | Özel | Java | Hızlı, ölçeklenebilir, saf Java anlamsal grafik veri tabanı. |
| Teradata Aster | v6 (2013) | Özel | Java ,SQL , Python , C ++ | Yüksek performanslı, çok amaçlı, yüksek ölçeklenebilir ve genişletilebilir MPP veritabanı yerli SQL, MapReduce ve Grafik veri depolama ve işleme destekleyen motorları patentli da kapsayan. Analitik fonksiyon kütüphaneleri ve veri görselleştirme yetenekleri bulunur. |
| Titan | 0.5.2 (2014) | Apache 2 | Java | Aurelius tarafından geliştirilen bir dağıtılmış, gerçek zamanlı, ölçeklenebilir işlem grafik veri tabanı. |
| Trinity |  |  | C # , C ,X64 Assembly | Bir bellek bulut üzerinde bir dağıtılmış genel amaçlı grafik motoru. |
| Triplebit |  |  | C / C ++ | Merkezi bir RDF mağaza. |
| VelocityGraph |  | Tescilli back-end ile Açık kaynak | C # | Yüksek performans, VelocityDB nesne veritabanı ile ölçeklenebilir ve esnek grafik veritabanıdır. |
| VertexDB |  | Revize BSD | C | Otomatik çöp toplama destekleyen bir grafik veritabanı sunucusu. |
| Weaver | 0.1 (Aralık 2014) | BSD lisansları | C , Python | Çoğaltma ve göç ile hızlı, ölçeklenebilir bir grafik veri tabanıdır. |
| WhiteDB | 0.7.0 (Ekim 2013) | GPLv3 ve ücretsiz ticari lisans | C | Bir grafik / N-küpe bellek veri tabanı kütüphanesi paylaştı. |
| OhmDB | 1.0.0 (Ağustos 2014) | Apache 2 | Java | Java RDBMS + NoSQL Veritabanı. |

**Graph Database Uygulama Örnekleri**

**OrientDB:** OrientDB’nin kurulumunu adım adım uygulayacağız.

OrientDB oldukça küçük bir boyuta sahiptir. Uygun sürümü indirip açmamız gerekiyor.

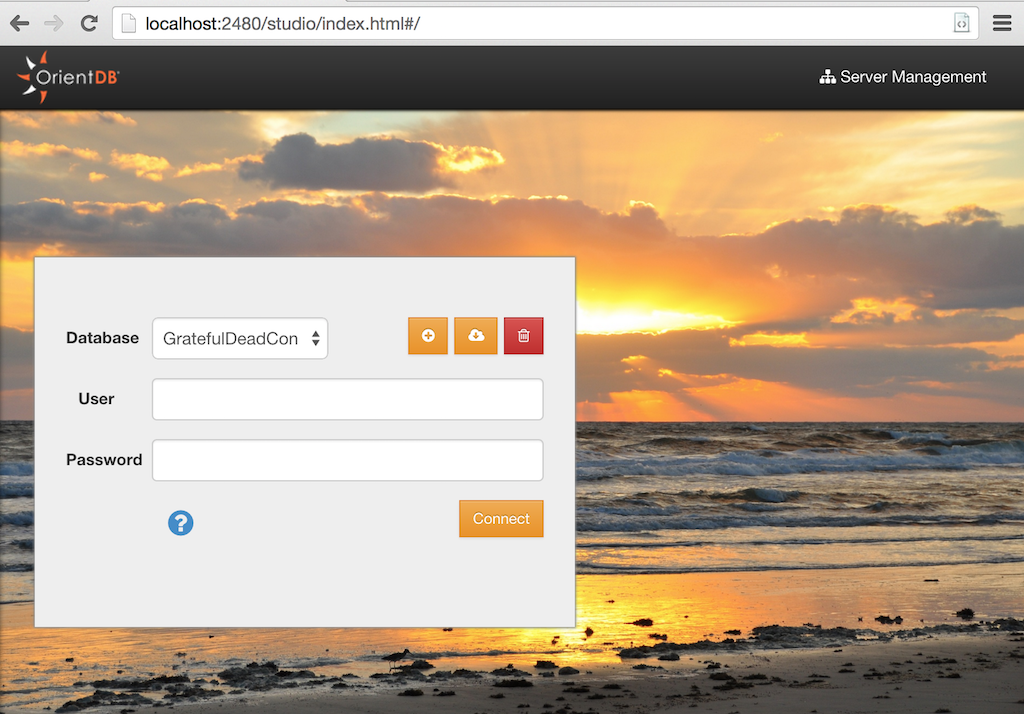
wget --output-document orientdb-community-2.0.4.tar.gz "http://www.orientechnologies.com/download.php?file=orientdb-community-2.0.4.tar.gz&os=multi

tar xvf orientdb-community-2.0.4.tar.gz

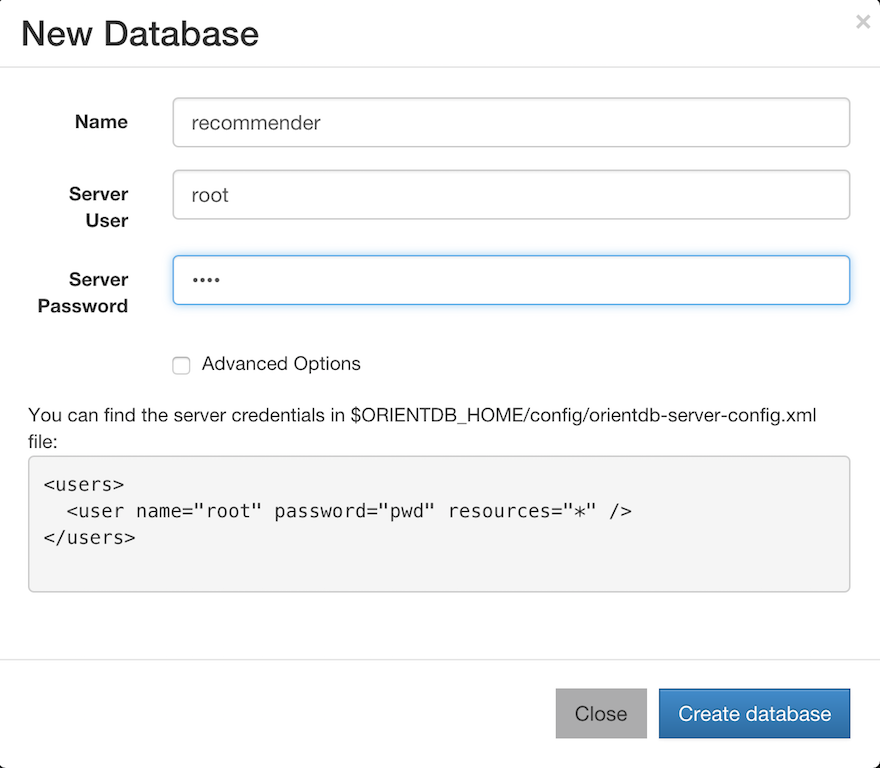
cd orientdb-community-2.0.4

bin/server.sh

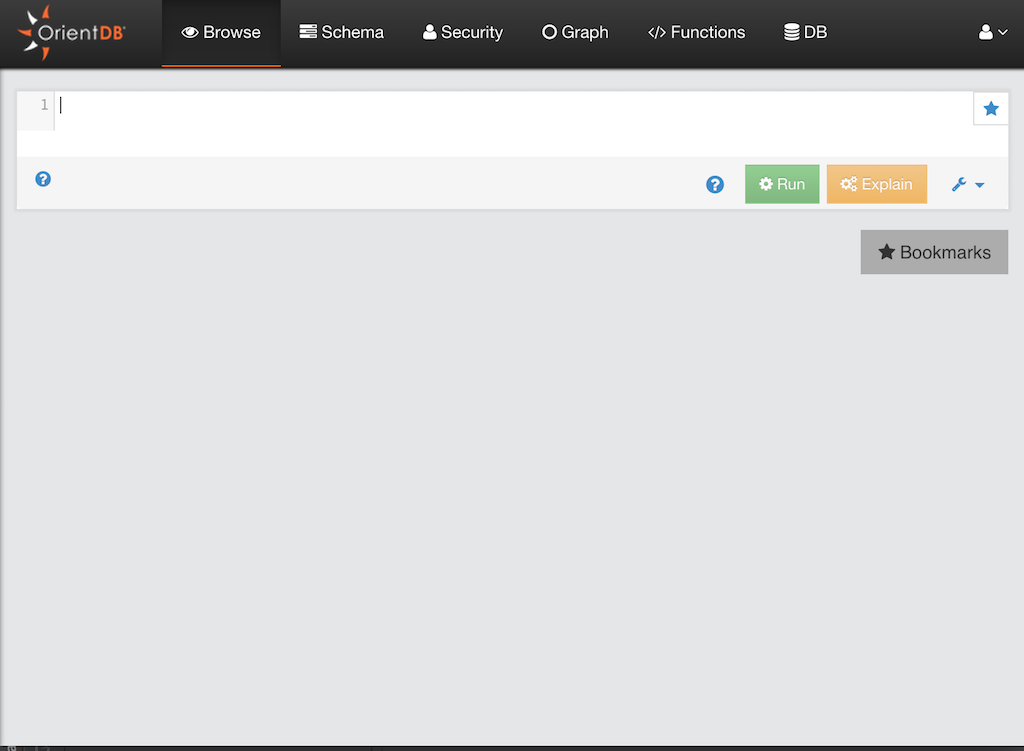
Sunucu ilk seferinde ayağa kalkarken bize root şifresini soruyor. Bunu belirledikten sonra OrientDB’nin ara yüzüne girebiliriz. Aslında konsol üzerinden de kullanılabilir ama OrientDB pek çok NoSQL’de eksik olan basit ama oldukça kullanışlı bir admin ara yüzüne sahip. Bu ara yüzden birçok işlemi gerçekleştirebiliyoruz. http://localhost:2480 üzerinden ara yüzü açabiliriz.



Buradan yeni bir veritabanı oluşturmayı seçip ismini giriyoruz:

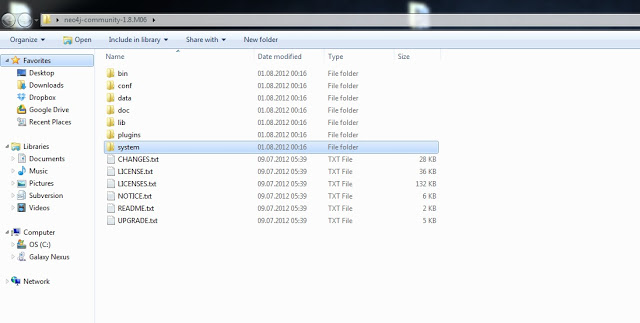


Girişi yaptıktan sonra karşımıza ara yüz ekranı geliyor. Browse kısmında sorguları çalıştırabilir, Graph kısmında verileri görsel olarak inceleyebilirsiniz:

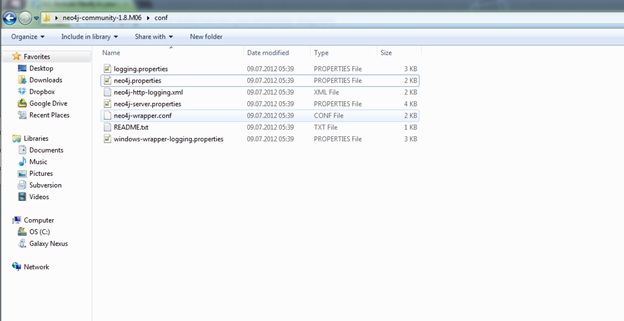


# [Neo4j Graph Database Kurulumu](http://www.ikimuhendis.com/2012/08/neo4j-graph-database-kurulumu.html)

Sisteminize uygun olan kurulumu indiriyorsunuz. İndirdiğimiz klasörün içi aşağıdaki gibi olacak.

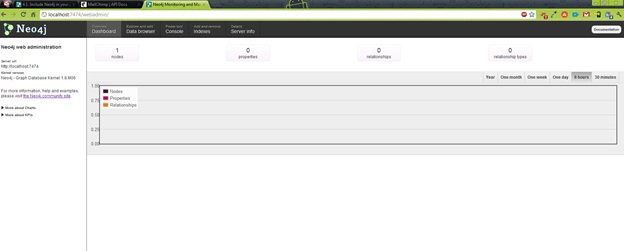


Neo4j'yi başlatmadan önce konfigürasyonunu yapalım. Bunun için kurulum klasöründeki conf dosyasının içine giriyoruz.Bizim için önemli olan neo4j.properties ve neo4j-server.properties dosyalarıdır.

neo4j.properties dosyasında bizim için önemli olan otomatik indeksi ve hangi özelliklerin otomatik indeksleneceği ayarını giriyoruz.

Dosyada bu özellikler ilk olarak yorum satırı şeklinde gelir bunların başındaki # kaldırırsak etkinleştirmiş oluruz. node ve relationlar ayrı olarak indekslerinden parametreleri farklıdır. \*indexable alanları ise bizim db'de kullanacağımız node ve ilişkide indexlemek istediğimiz alanları yazıyoruz. name,age, id gibi...

#node\_auto\_indexing=true  
#node\_keys\_indexable=name,age  
#relationship\_auto\_indexing=true  
#relationship\_keys\_indexable=name,age  
  
neo4j-server.properties dosyasında db lokasyonu, portlar ve path'ler tanımlanır. Bizim için önemli olanları aşağıda verelim.  
  
org.neo4j.server.database.location={KlasörYolu}  
Bu ayar database'in nereye kaydedileceği bilgisini tutuyor.  
  
org.neo4j.server.webserver.port=7474  
Bu ayar db çalışırken dışarıya rest api'leri ile de bağlanılmasını mümkün kılıyor. Bunun için yayın yapacağı portu veriyoruz.  
  
org.neo4j.server.webadmin.data.uri=/db/data/  
Burada ise yukarıdaki rest api'yi çağıracakları path'i veriyoruz örneğin çağırırken http://localhost:7474/db/data adresine rest requesti yaparak cevap alırız.  
  
Bu ayarları yaptıktan sonra server'i başlatmak için kurulum dosyanın altındaki bin klasörüne giriyoruz. Neo4j.bat dosyasnı çalıştırdığımız zaman komut satırı açılıp arkada bir server başlatacak. DB'nin çalışıp çalışmadığını kontrol etmek için http://localhost:7474/ tıklayarak db'nizi web ara yüzü ile görüntüleyebilirsiniz.



# [(Java ile Neo4j Kullanımı)](http://www.turkishh.com/programlama/graph-database-java-ile-neo4j-kullanimi/)

 Kullandığınız işletim sistemine göre neo4j’yi download ediniz ve graph database ara yüzümüzü  <https://github.com/neo4j/neoclipse/downloads>  adresinden indirelim…

Oluşturduğumuz java projemize indirdiğimiz neo4j kütüphanemizi ekleyelim …

Şimdi kodumuzu yazmaya başlayalım ilk önce databasemizi oluşturalım .

public class DenemeNeoStudent {

public final static String NEO\_INDEX\_PATH = "C:\\Users\\MeMoCeLL\\Desktop\\neo\\neo";

static Node root;

static GraphDatabaseService graphDb;

public static void createDatabase() {

        graphDb = new EmbeddedGraphDatabase(DenemeNeoMeMo.NEO\_INDEX\_PATH);

//     Burada databasemizi export edecek bir dizin soruyor bize bizde yukarda dizinimizi vermiştik.

        registerShutdownHook(graphDb);

//      veritabanımızın kapamasını sağlar

        root = graphDb.getReferenceNode();

//      Oluşturduğumuz veritabanımızdan bir tane Node için referans alıyoruz bunu aşağıda user eklerken kullanacağız..

    }

}

// Databasemizi sağlıklı bir şekilde kapatmaya yarayan method

private static void registerShutdownHook(final GraphDatabaseService graphDb) {

        Runtime.getRuntime().addShutdownHook(new Thread() {

            @Override

            public void run() {

                graphDb.shutdown();

            }

        });

    }

Bu şekilde run edersek eğer kodumuzu  gösterdiğimiz dizine databasemizi create eder ama içi boş olur .

Şimdi gelelim  içini doldurmaya  zaten graph database  kullanıcının yoğun olduğu sistemlerde çok hızlı sorgular için kullanılır ve nesneler arası bağlantıları daha iyi analiz etmemiz için tasarlanmıştır. Şimdi küçük bir  örnek yapalım . 10 tane öğrencimiz olsun (adı , soyadı , okulu , fakültesi ve grup no) ve  her öğrencinin de  5 er tane dersi olsun.

public static void createStudent() throws SQLException {

// Neo4j  Transaction ile çalışır yani işlem  sırasında bir hata oluştuğunda hiç birşey yapmaz işlem başa geri alınır.

Transaction tx = null;

        try {

            tx = graphDb.beginTx();

     // Databasemizi kullanmamız için Transaction başlatıyoruz.

            int b = 0;

            StudentList sl = new StudentList();

            while (b < 10) {

                Node node = graphDb.createNode();

       // Burada her öğrenci nesnemizi bir node olarak düşünelim ..

                node.setProperty("userid", sl.getStudentList()[b][0]);

              // Her öğrenciniz bilgilerini set ediyoruz.

                node.setProperty("firstName", sl.getStudentList()[b][1]);

                node.setProperty("lastName", sl.getStudentList()[b][2]);

                node.setProperty("faculty", sl.getStudentList()[b][3]);

                node.setProperty("group", sl.getStudentList()[b][4]);

                root.createRelationshipTo(node, Gorevi.STUDENT);

              // Node ‘i ve ilişki tipini burada yukarda oluşturduğumuz Node yolluyoruz

                System.out.println("\tNeo Profile Olusturuldu : " + node.getProperty("firstName"));

                b++;

            }

            tx.success();

          //  Transaction tamamlanması ..

        } catch (Exception e) {

            tx.failure();

         // hata zamanı

            System.out.println("\tNeo Transaction Olusturulamadi" + e);

        } finally {

            if (tx != null) {

                tx.finish();

        // Transaction bitirilmesi

                System.out.println("\tNeo Transaction Bitti");

            }

        }

    }

// İlişki tipini enum şekilde oluşturuyoruz

public static enum Gorevi implements RelationshipType {

STUDENT, LESSONS

}

Şuanda  kodumuzu çalıştırsak kaç tane öğrenci eklediysek bunu oluşturacaktır.Burada direkt relational databasemizi (oracle , mysql vs ) gerekli connectionları oluşturup neo4j’ye aktarabilirdik ama daha sade ve anlaşılır olması için böyle bir yolu seçtik …

Şimdi gelelim oluşturduğumuz her öğrenciye derslerini ekleyelim …

private static void createLessons() throws SQLException {

        Transaction tx = null;

        try {

            tx = graphDb.beginTx();

            Traverser traverser = root.traverse(Order.DEPTH\_FIRST,

                    StopEvaluator.END\_OF\_GRAPH,

                    ReturnableEvaluator.ALL\_BUT\_START\_NODE,

                    Gorevi.STUDENT, Direction.OUTGOING);

            int count = 0;

            for (Node node : traverser.getAllNodes()) {

                Node dersler = graphDb.createNode();

                dersler.setProperty("Yapay Zeka", "88");

                dersler.setProperty("Gelişmiş Bilgisayar Şebekeleri", "73");

                dersler.setProperty("Bilişim Etiği ", "92");

                dersler.setProperty("Gerçek Zaman Sistemleri ", "93");

                dersler.setProperty("Bilgi Güvenliği", "95");

                node.createRelationshipTo(dersler, Gorevi.LESSONS);

                tx.success();

                System.out.println(count+" . "+"öğrenciye dersleri eklendi..");

                count++;

            }

        } catch (Exception ex) {

            tx.failure();

            System.out.println("\tDersler Oluşturulmadı");

            System.out.println("===================================");

            ex.printStackTrace();

            System.out.println("===================================");

        } finally {

            tx.finish();

        }

    }

  Burada dikkat edilmesi gereken olay  Traverser’dir . for a girerken traverser.getAllNodes()) şeklinde bir kod yazmışız bu şu demek oluyor bizim databasemizindeki bütün nesneleri çağırır ve her döngüde bir nesneye aşağıdaki belirttiğimiz dersleri  ekler ve bunu  for un üstündeki şu kod parçasıyla anlar hangi ilişki türünden nesneye hangi dersleri ekleyeceğiz.Altta açıklayalım onu …

Traverser traverser = root.traverse(Order.DEPTH\_FIRST,

StopEvaluator.END\_OF\_GRAPH,

ReturnableEvaluator.ALL\_BUT\_START\_NODE,

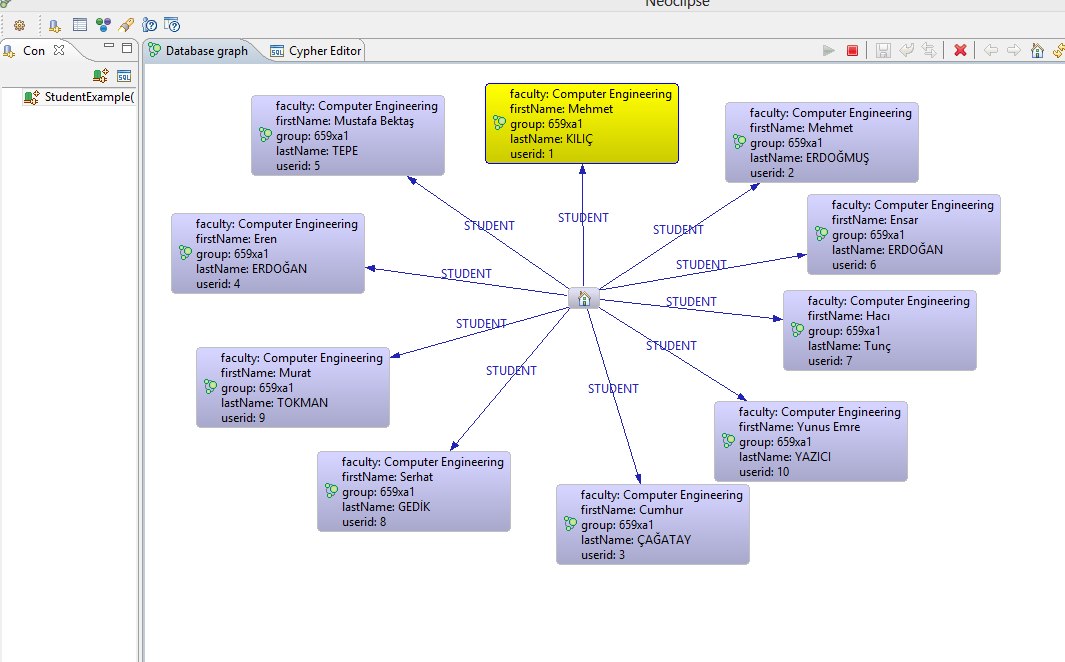
Gorevi.STUDENT, Direction.OUTGOING);

Buradaki görevi.STUDENT  hani biz yukarda databasemizi çağırdığımızda bize tüm nesneleri (öğrencileri) getirmişti .Oradaki ilişki  tipi student olan  nesnelere bu dersleri eklemesini söyledik.. Direction.OUTGOING  bu dışa doğru yayılma yani ağaç mantığını düşünün …

Kodumuzu bu şekilde çalıştırdığımızda sorunsuz çalışması lazım ve yukarda verdiğimiz dizin de yüklediğimiz neoeclipse de çalıştıracağımız türden bize klasör oluşturacak..

Neoeclipseyi açtıktan sonra sol taraftaki connection bloguna gelerek sağ tıklayıp new conneciton diyoruz ve sonra bir isim yazıyoruz ve dizini browse diyerek seçiyoruz ve tamam diyoruz.

Ve sonrasında da oluşturduğumuz connecitonu çalıştırdığımız zaman graph databasemizi kullanama başlayabiliriz. Her öğrenciye tıkladığımızda onun derslerinin listesi ve notları gelecek .



**Serter Öztürk**

**13125032 (Graph Database)**