Table of Contents 目录

1. Install neo4j

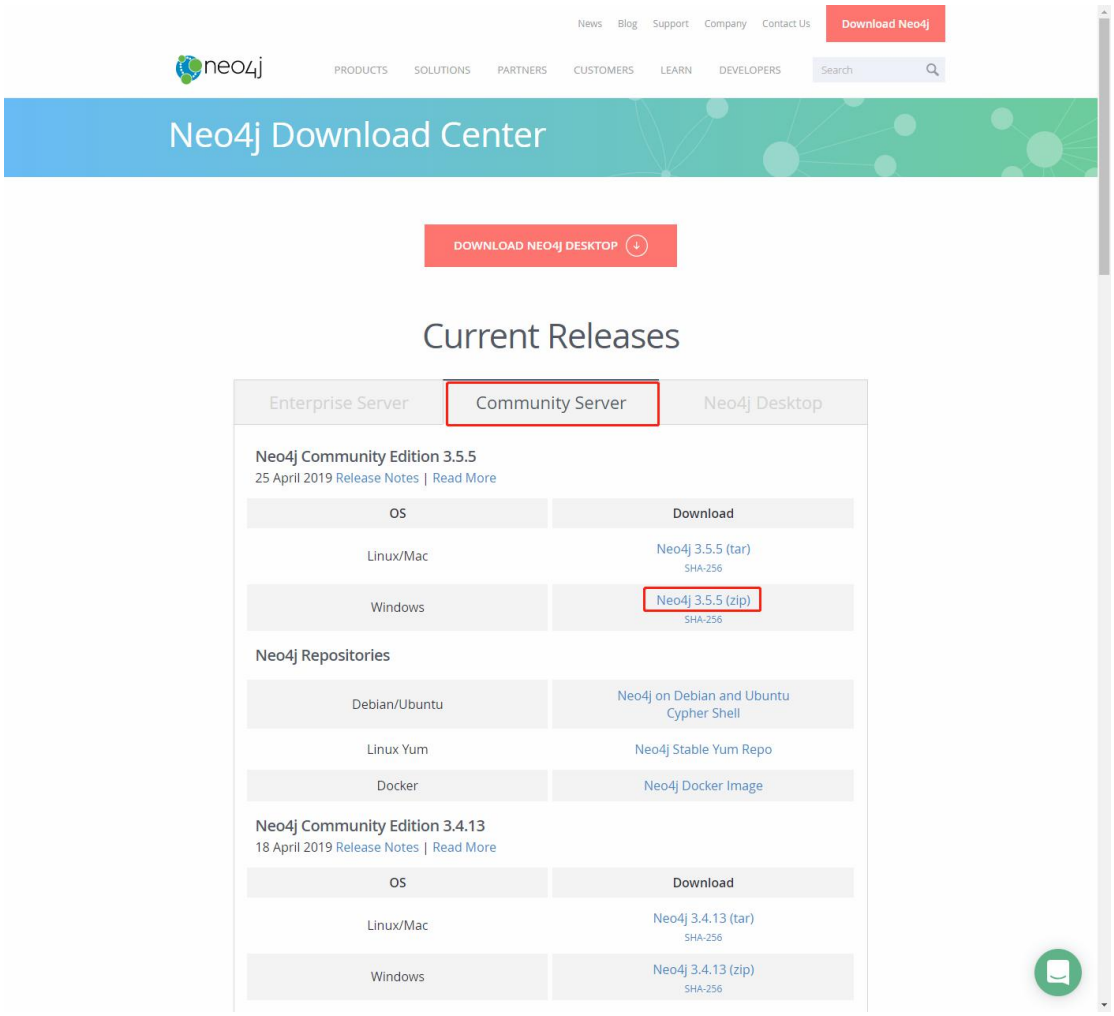
1.1 Confirm installation of Java

You must install OpenJDK 8 or Oracle Java 8 on your windows system before you install neo4j.

Note: recommended for Neo4j 3.0.x Version 7 is recommended for releases prior to 2.3.0.

1.2 Download neo4j

Website: https://neo4j.com/download-center/#releases



1.3 Unzip the file

Find the zip file you just downloaded and right-click, extract all.

1.4 Define environment variables

Place the extracted files in a permanent home on your server, for example D:\neo4j\. The top level directory is referred to as NEO4J_HOME.

1.5 To run Neo4j as a console application

Open Command Prompt and input:

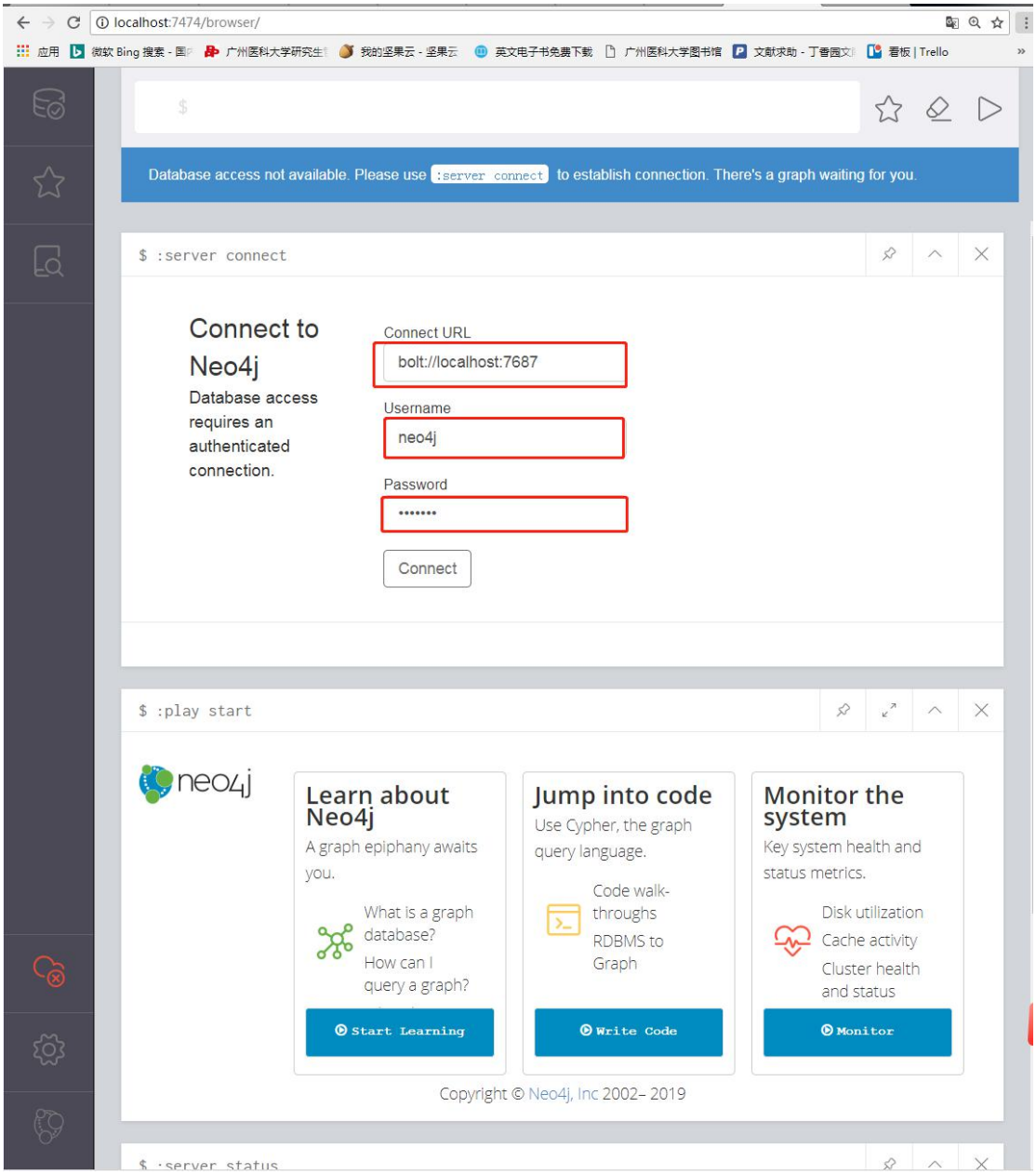
```
f:
cd path/neo4j/bin
neo4j.bat console
```

For additional commands and to learn about the Windows PowerShell module included in the Zip file, see the Windows installation documentation.

1.6 Open Neo4j Browser

Visit http://localhost:7474 in your web browser.

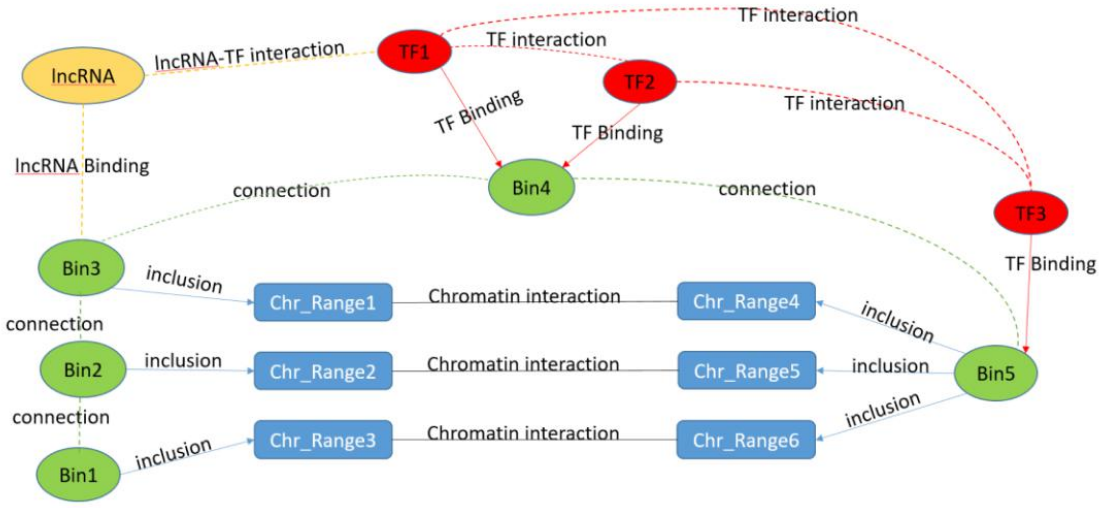
Connect using the username 'neo4j' with default password 'neo4j'.



2 GREG

GREG (The Gene Regulation Graph Database) is a graph database written in the Neo4J language. Its goal is to allow genomic researchers to see in a graphical way all the known interactions between proteins, lncRNAs, and DNA for a given transcription factor, lncRNA, or genomic region (ie., its “regulatory landscape”).

We built a GREG website http://210.38.57.54:8080/index.html for free to academic use.



As you can see, we find relationships directly between proteins, ncRNAs and DNA in GREG graph.

2.1 The characteristics of GREG

2.1.1 Easily Find relationships between proteins, ncRNAs and DNA

The first reason is integration: You can always use different available resources to get regulatory information, such as the UCSC Genome Browser, the WashU Epigenome Browser, and the websites of the individual databases here included such as Cistrome, iRefIndex, and 4DGenome, or their alternatives (such as ENCODE, STRING or 3CDB). With all that information, you can certainly build your regulatory landscape.

However, GREG offers an unified view of all the different interactions, saving the user time and energy. You could get, for example, a genomic region with enhancers connecting different promoters, promoters different enhancers, and TFs acting on enhancers and promoters, which is better than watching the above-mentioned

browsers independently. Besides, we have been careful with the integration process, whose fine details may not be obvious for a biological researcher.

2.1.2 Directly get a graph having relationships about your projects

The second reason is the graphical output: The user does not need to get a text file as a result and then convert it to a graph. The user gets the graph instead (both displayed on screen and as a graph file).

2.1.3 The possibility of graphical queries

The third reason is the possibility of graphical queries: we are using graph databases, so we can use fancy graph queries, such as extending the query to the first or second neighbors of the query nodes, or to find the shortest paths between two nodes; for example, finding a path in the graph that relates two genomic regions, or a TF and a genomic region. That means that we can also ask questions such as "what happens in the neighborhood of that module/node?" or reformulate the question "how are these two regions/nodes correlated?" to "how are these two regions/nodes connected?", among other questions that make use of graph theory concepts (such as detecting critical nodes and so on).

All of this can be done from a very simple website; no need to get data from different resources, load it to R, write scripts for data integration, make network analysis with igraph, and so on. Just follow three simple steps.

2.2 GREG information

2.2.1 Where is GREG information coming from?

GREG is built by consolidating data from one protein-protein interaction database (iRefIndex), one DNA-DNA interaction database (4DGenome), one lncRNA-DNA binding database (LnChrom), one lncRNA-protein database (POSTAR2/starBase), and one protein-DNA binding database (Cistrome), and building a graph database with all the available data.

2.2.2 organism

For now, GREG only supports hg38 genome version, includes 8 human cell types. These are three stem cells (H1, IPS19.11, IPS6.9), four cancer cells (A549, K562, MCF-7, HeLa), and one other cell line (IMR-90).

2.2.3 relationship types

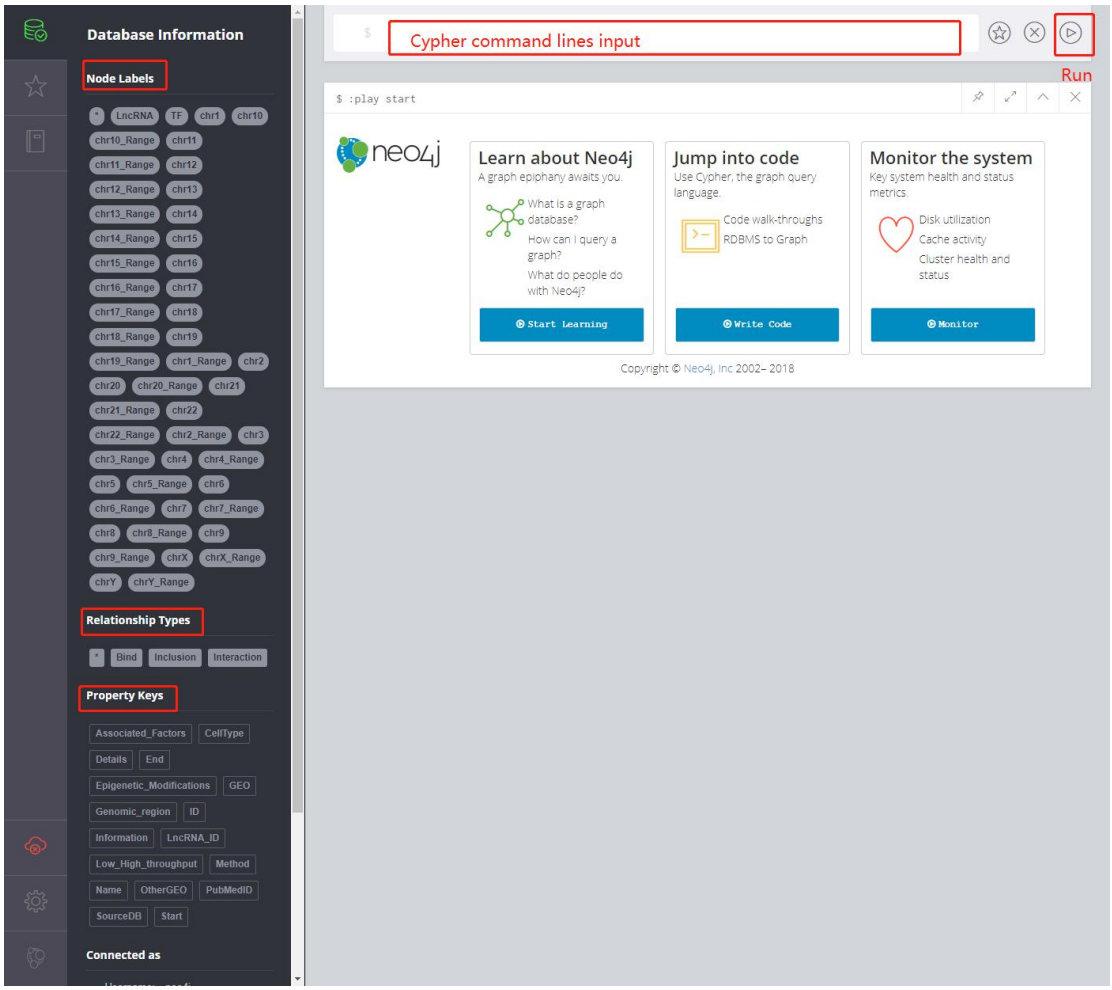
Currently, GREG has many Node Labels, 3 Relationship Types(Bind, Inclusion and Interaction) and many Property keys.

2.2.4 GREG accepts

We can use gene names to query GREG for TFs and lncRNAs, also, GREG can accepts DNA annotation such as Ensemble gene ID, gene type (protein coding, miRNA, lincRNA, etc), and gene name.

2.2.5 GREG tutorial

The tutorial of GREG web vision is on the website http://210.38.57.54:8080/Tutorial.html
we can view next chapter to perform GREG-neo4j.



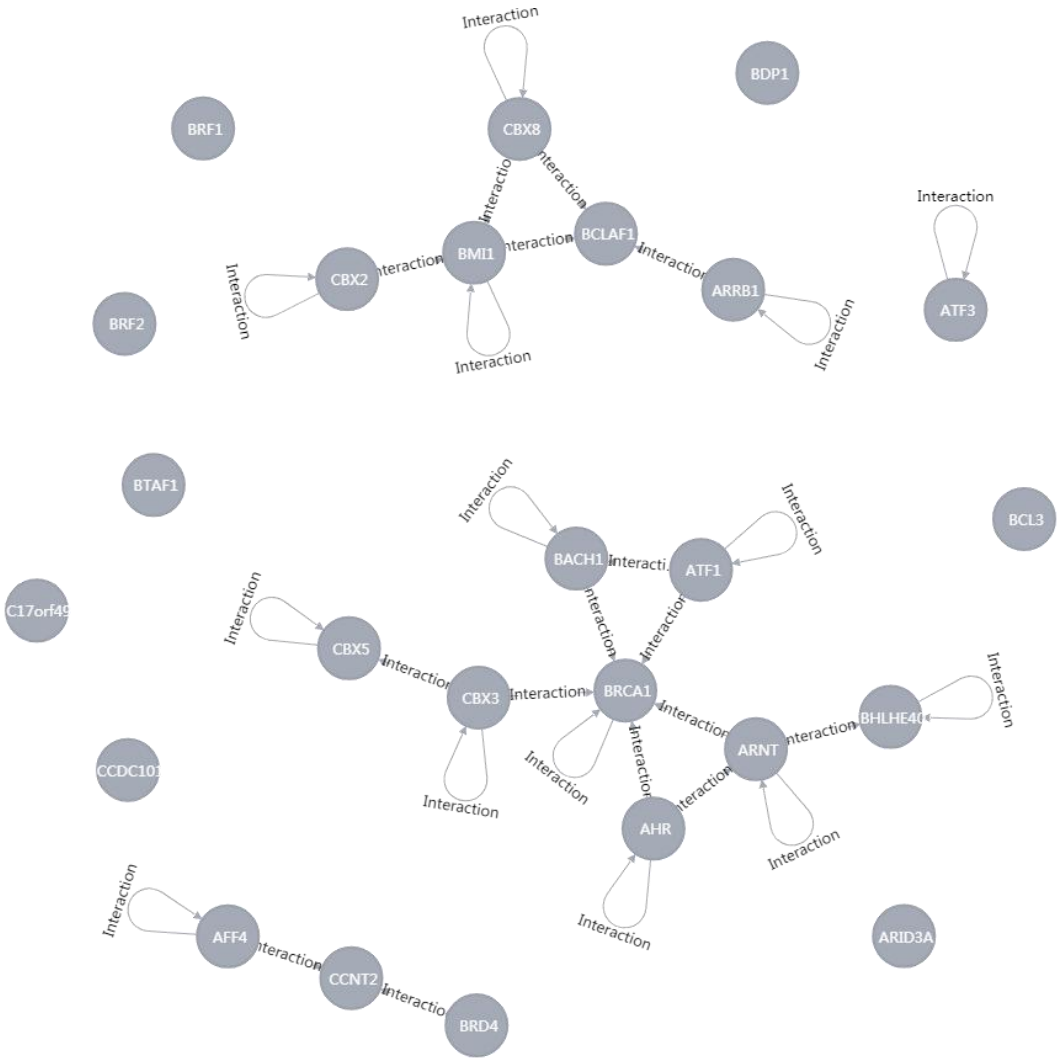
3 Cypher

3.1 Retrieve all nodes by lable.

MATCH to get all nodes

The “MATCH” command can be used to retrieve all nodes in GREG with a specific label. The following example return the TF nodes but limit the results to only 25 TFs:

```
MATCH (n:TF) RETURN n LIMIT 25
```



In this example, TF is the label of the nodes

n is Variable name, used to represent the results obtained.

MATCH(n:TF) selects all nodes with the TF label

LIMIT 25 limits the results to 25 nodes with the TF label.

The following commands offer a quick exploration to all the other nodes (chr, chr_range) present in GREG.

* MATCH (n:chr1) RETURN n LIMIT 25
* MATCH (n:chr1_Range) RETURN n LIMIT 25

More exercise

1. Please query all the Bin nodes in chromosome one(chr1), limits the results to 50 nodes.

2. Please query all the Range nodes in chromosome one(chr1), limits the results to

```
50 nodes.
```

3.2 Query by property

3.2.1 Get a node by property

1. Query by TF name.

When you have transcription factors name, you can get its nodes. Here we show an example, the transcription factors name is BCL3, we can use the command as follows to get BCL3 node:

```
MATCH (n: TF {Name:'BCL3'} )    RETURN n
```

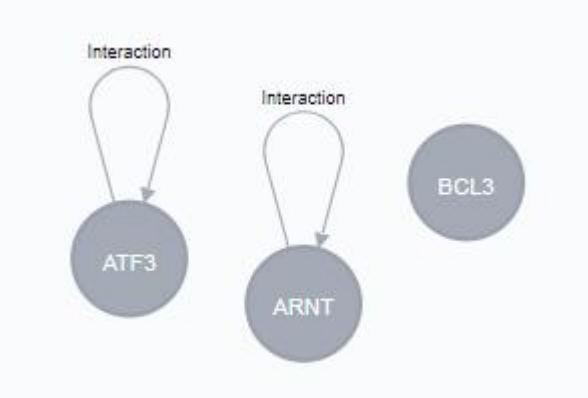
You can see a graph only having a node, is BCL3.

Also, we can use the following command to get the same result:

```
MATCH (n: TF ) where n.Name = 'BCL3' RETURN n
```

When we want to query a lot of nodes in one time, you can use the **IN** command.

```
MATCH (n:TF) where n.Name    in    ['BCL3', 'ARNT', 'ATF3'] RETURN n
```



We get BCL3, ARNT and ATF3 nodes in a graph.

Here we use where to fiter the resulting of match(n:TF).

IN operator

In operator provide a collect way for the value. The command syntax is:

```
IN[<Collection-of-values>]
```

More exercise

1. Please query a TF named BDP1.

   Use the { } style (MATCH (n: TF {Name:' BDP1'}))and the Where command (MATCH (n: TF) where n.Name = ' BDP1') to set the query condition.

What is the different between the two style?

2. Please query a TF list named BRF1, BRF2, GATA1.

2. Query by genomic range.

GREG can be used to query all the information associated to a given genomic range in a specific chromosome.

```
MATCH (m:chr3) where toInteger(m.Start)>195540000 and
toInteger(m.End)<195560000
return m
```

The result show all the information associated to a given genomic range(195540000 to 195560000) in a specific chromosome(chr3).

toInteger(): is a function change string to integer numbers.

More exercise

1. What is the function (gene information) of the genomic range (95520000 to 95540000) in chromosome 13?

3 Get nodes in a given DNA annotation

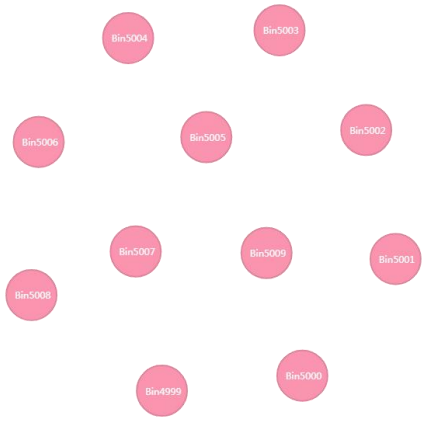
GREG can be used to query all the information associated to a given DNA annotation.

The example is querying a node including gene RBP7.

```
MATCH(n) where n.Details contains 'RBP7' RETURN n
```

Or we use following command to query:

```
MATCH(n:chr1) where n.Details contains 'RBP7' RETURN n
```



In this result, we got 11 nodes contains RBP7.

Contains always be used to query nodes contains the words which you are interesting.

More exercise

1. What is the Gene ID of gene PAFAH2.

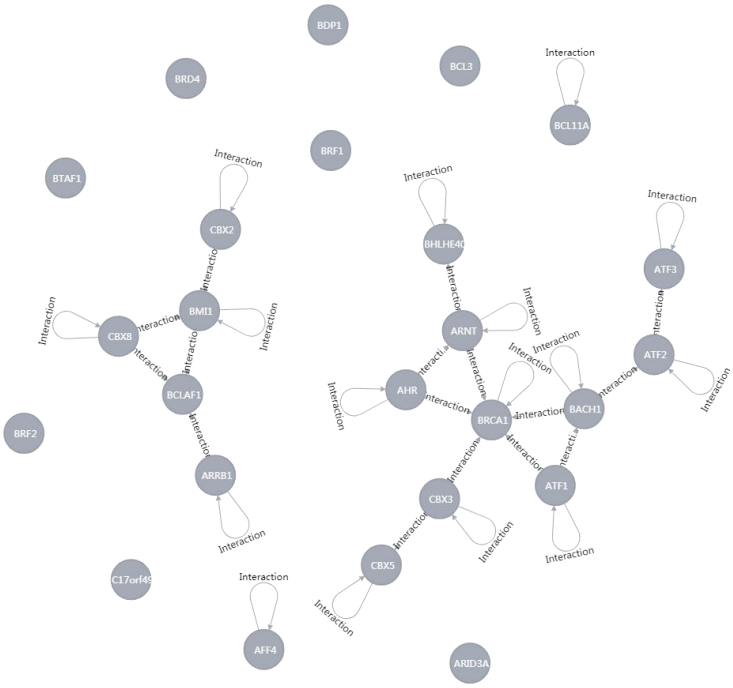
3.2.3 Order the results by property

After you queried, you want order those nodes information to get best, here show an example that use ORDER BY for ordered the results by property.

```
MATCH (n :TF) RETURN    n    ORDER BY n.Name    DESC LIMIT 25
```

You will get 25 nodes with started letters from z to a.

Note: DESC will make order is descending. If DESC didn't use, we will get 25 nodes with stared letters from a to z.

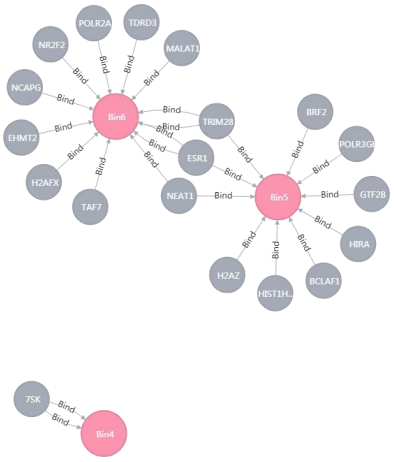


3.3 Retrieve relationships

3.3.1 Retrieve all Bind relationships

Relationships are the database goal. The first thing is to make our know that the basic query to relationship.Let us see the Bind relationships in the GREG. Command is following:

```
MATCH p=()-[r:Bind]->() RETURN p LIMIT 25
```



We only show 25 nodes. As you can see thouse genes are binding to Bin5 or Bin6.

Notes:

(a)-[r]->(b) pattern is the basic pattern of relationships, (a) is start node, (b) is the end node, [r] is relationship between a and b, -> arrow means the relationship has a direction (a) to (b) . If we use (a)-[r]-(b) pattern. Which means the relationship between a and b has no direction.

[r :Bind] : set the relationship type is Bind.

MATCH to get all relationships

The following commands offer a quick exploration to all the relationships (Inclusion, Interaction, TF Interaction) present in GREG.

* MATCH p=(:TF)-[r:Bind]->(:chr1) RETURN p LIMIT 25
* MATCH p=()-[r:Inclusion]->() RETURN p LIMIT 25
* MATCH p=(: chr1_Range)-[r:Interaction]->(: chr1_Range) RETURN p LIMIT 25
* MATCH p=(:TF)-[r:Interaction]->(:TF) RETURN p LIMIT 25

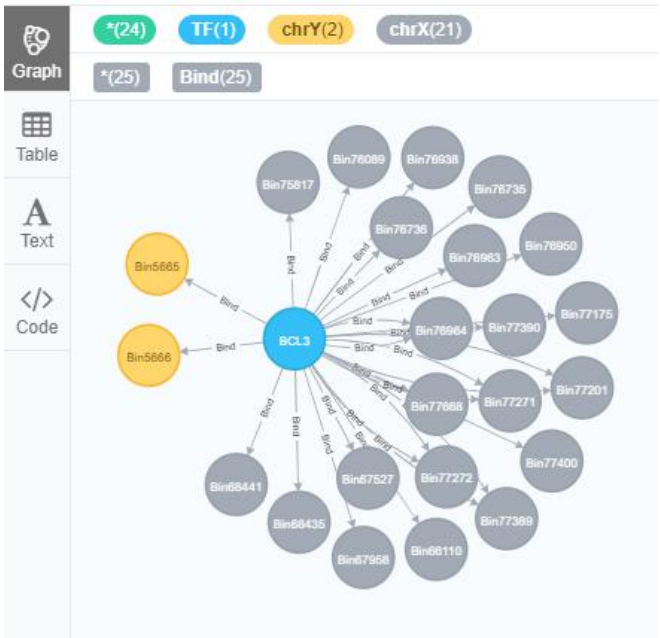
3.3.2 Which gene are a TF binding to?

GREG can support to find out Which gene are a TF binding to.

the example is showing the bin nodes which is BCL3 binding to.

You can click the bin nodes to get the gene information.

```
MATCH p = ( n: TF {Name:'BCL3'} ) -[r :Bind]-() RETURN p LIMIT 25
```



Notes: (n: TF {Name:'BCL3'}) is the same as patterns we match those nodes.

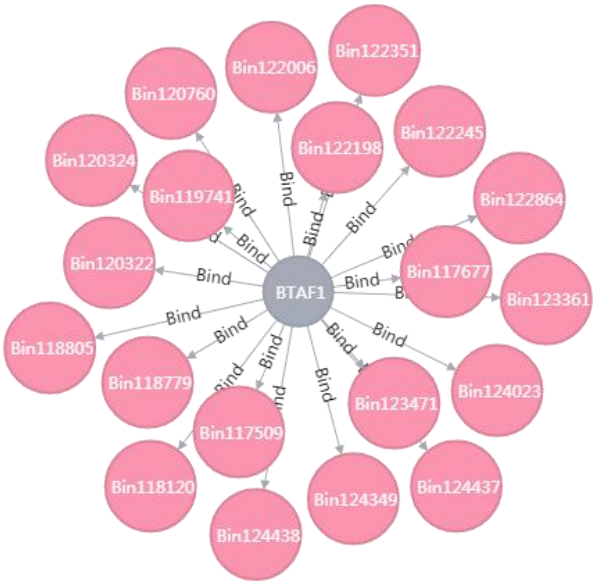
3.3.3 Which gene in special Chromosome are a TF binding to?

GREG can find out all the gene information in special Chromosome that a TF bind to.

The example here is showing the gene information BTAF1 binding to chr1.

You can click the bin nodes to get the gene information.

```
MATCH (n:TF {Name:'BTAF1'} ) -[r:Bind]- (m:chr1)
RETURN n , r, m LIMIT 20
```

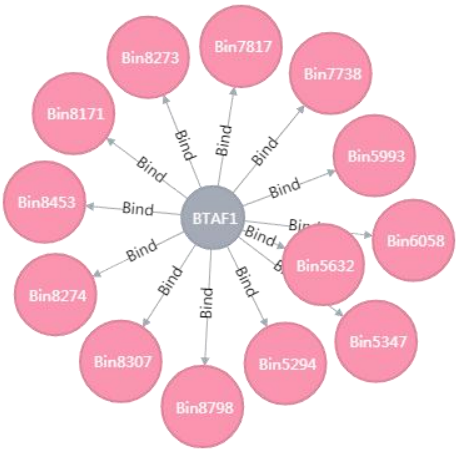


3.3.4 Which gene in genomic range are a TF binding to?

We can query what a TF work in a specific region.

For Example: BTAF1 in chr1: 10000000-20000000**.**

```
MATCH (n:TF {Name:'BTAF1'} ) -[r:Bind]-> (m:chr1)
where toInteger(m.Start)>10000000 and toInteger(m.End)<20000000
RETURN n , r, m LIMIT 25
```



3.3.5 Is TF BTAF1 binding the gene PEX14?

GREG support you to find out is a TF binding to a special gene.
For Example: IS BTAF1 binding PEX14 in chromosome one.

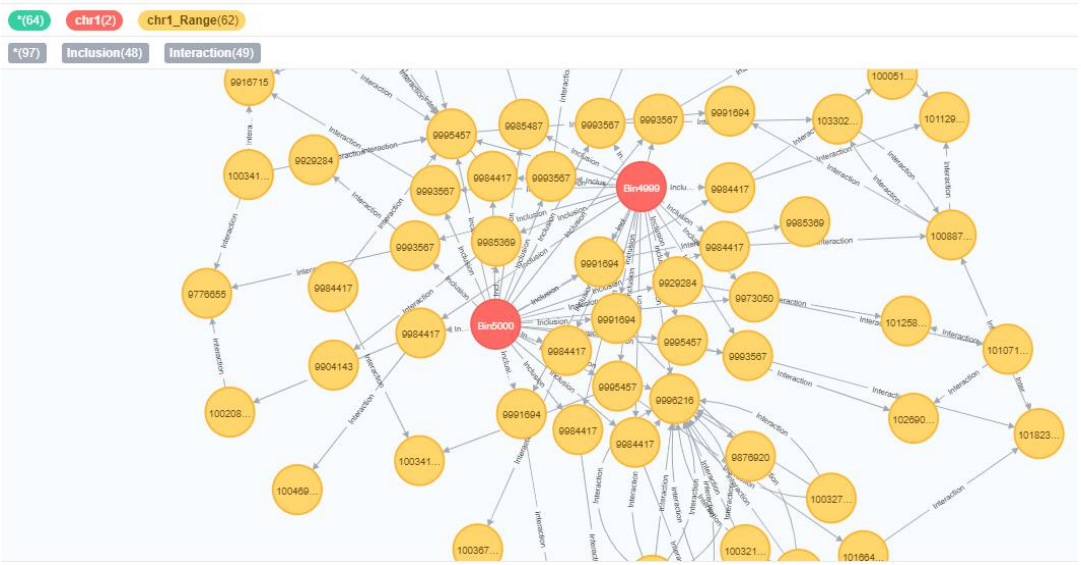
```
MATCH p =    (m1:TF{Name: 'BTAF1'}) - [r1:Bind]-> (n:chr1)
 where n.Details contains 'PEX14'
RETURN p
```

3.4 multiple-relationship queries

3.4.1 get genomic interactions for a special DNA annotation

GREG can support Find the genomic interactions for a special DNA annotation.
The example is to find out the genomic interactions on chromosome one of RBP7

MATCH p=(n:chr1) -[r1:Inclusion]-> (m1) -[r2:Interaction]- (m2)
where n.Details contains 'RBP7' and tointeger(n.Start) < 10000001
RETURN p

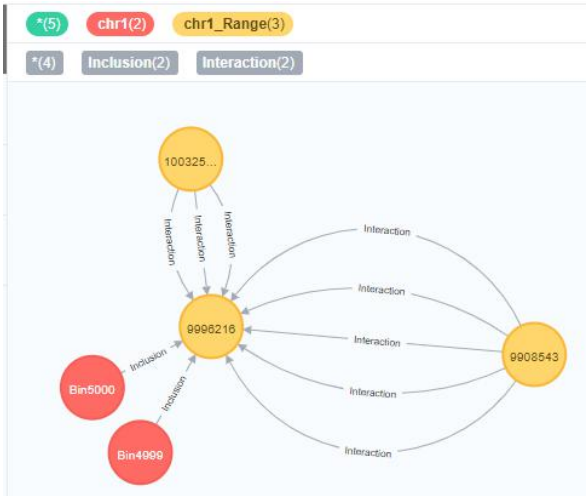


(n:chr1) -[r1:Inclusion]-> (m1) used to find out which range is 'RBP7 in.
(m1) -[r2:Interaction]- (m2) used to find out the genomic interactions

The example is to find out the genomic interactions on chromosome one of RBP7,

and in cell line K562.

MATCH p=(n:chr1) -[r1:Inclusion]-> (m1) -[r2:Interaction]- (m2)
where n.Details contains 'RBP7' and r2.CellType contains 'K562'
RETURN p

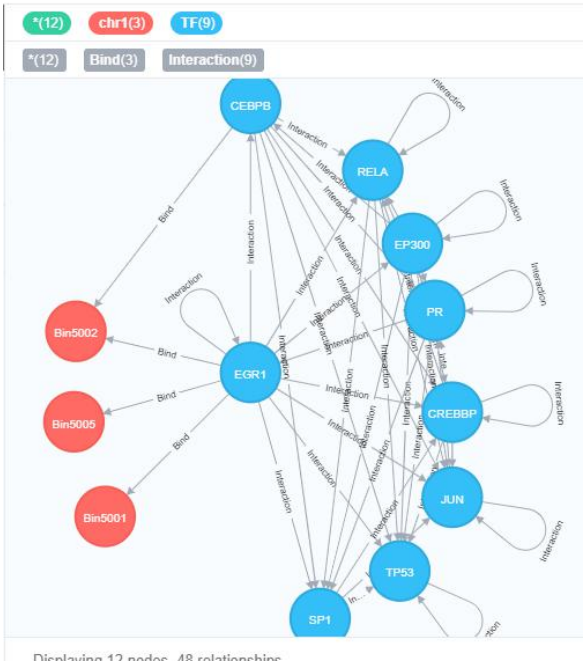


3.4.2 get TFs for a special DNA annotation

GREG lets you find multiple relationships between two nodes that are not directly related. For example, we find out relationship of two nodes that didn't have directly related. One node, which in chr1 annotation contains RBP7, bind related with EGR1, and other nodes also related with EGR1 as interaction relationship.

The example is to find out which TF is include in TF EGR1 binding to Gene RBP7

```
MATCH (n:chr1) <-[r1:Bind]- (m1:TF{Name: 'EGR1'}) -[r2:Interaction]- (m2)
where n.Details contains 'RBP7'
RETURN n,r1,m1,r2,m2
```

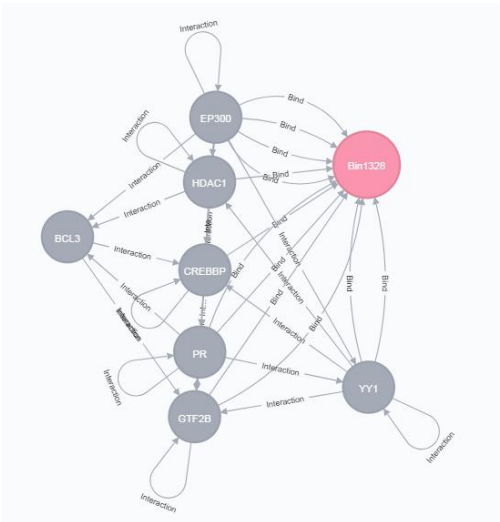


3.4.3 Find multiple nodes related the same nodes

GREG also can help you to get those genes which related more genes.

For example, we found genes, which not only related BCL3 genes but also related Bin1328 node, related YY1 genes.

```
match (n1:TF{Name:'BCL3'}) --(m1:TF) -- (n2:chr1{ Name:'Bin1328'})
with m1, n1, n2
match (m1:TF)--(m2:TF{ Name:'YY1'}) return m1,m2,n1,n2 LIMIT 10
```



Five genes are not only related to BCL3, Bin1328, but also related to YY1.

Notes:

The first command line is relationship pattern to get results m1,n1 and n2 for next command line as input. We always use with to connect two or more “match” command.

3.5 count nodes and relationships

Count() can help you to get how many data you query. Examples show you to count nodes and relationships.

```
MATCH (n:TF) RETURN count (n)
```

Result: 261

```
MATCH (n:chr10)
where toInteger(n.Start)>1000 and toInteger(n.End)<10000 RETURN count(n)
```

Result: 3

```
MATCH p=()-[r:Bind]->() RETURN count(r)
```

Result: 17945082

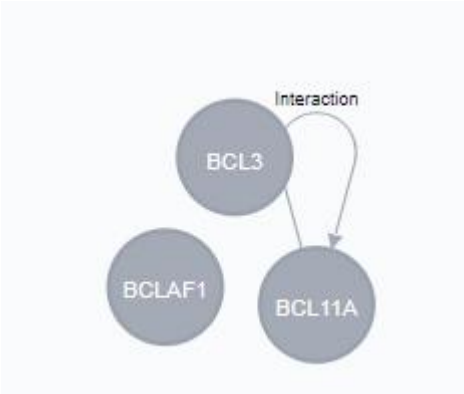
```
MATCH p=(n: TF {Name:'BCL3'})-[r:Bind]->() RETURN count(r)
```

Result: 7677

3.6 fuzzy query

GREG can help you to get nodes you don’t remember those full names. Here show you example to get name of TF nodes having ‘BCL’ letters:

```
MATCH (n:TF) where n.Name =~ '.*BCL.*' return n LIMIT 10
```



Notes:

=~ : represents that the following value is a regular expression

3.7 Collect value to list

collect() help you collect those property keys what you are interesting to a list, and you can export it as a CSV file.

For example: we can find out all the TFS binding to chr1: 10020000-10030000

```
MATCH (n:TF ) -[r:Bind]-> (m:chr1)
where toInteger(m.Start)>10020000 and toInteger(m.End)<10030000
RETURN collect(n.Name) as TFs
```

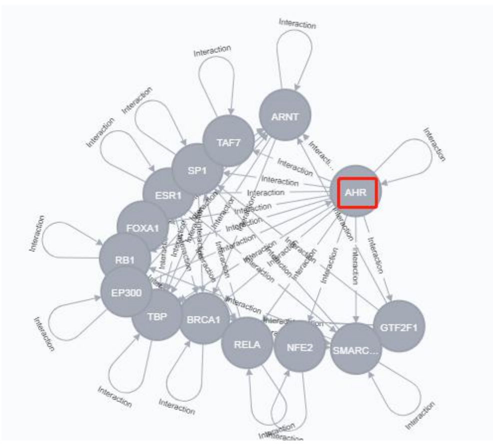
```
╒══════════════════════════════════════════╕
│"TFs"                                     │
╞══════════════════════════════════════════╡
│["POLR2A","MYC","SIN3A","NR2F2","HDAC2","HIRA","LMNB1","CBX3","CEBPA",│
│"GATA1","GATA2","TRIM28","ZBTB7A","ZNF143","H2AZ","E2F1","ESR1","FOXA1│
│","FOXM1","GATA3","SPI1","MAFK","MAFF","MAFK","LMNB1"]                │
└──────────────────────────────────────────┘
```

3.8 SUBGRAPHS

3.8.1 Subgraph of a TF/lncRNA:

GREG can help you get all information about a node only by a kind of relationship. Here show you AHR genes all relationships are Interaction.

```
match(n:TF{Name:'AHR'})
call apoc.path.subgraphAll(n, {maxLevel:1, relationshipFilter: 'Interaction'})
yield nodes ,relationships
return nodes ,relationships
```



3.8.2 Subgraph of a genomic range:

To get nodes directly related with those genes that are from 10,002,000 to 10,000,000 genomic

range in chr1, and those relationship are Bind or Interaction.

```
match(n:chr1) where not(toInteger(n.Start) >10002000 or toInteger(n.End)
<=10000000)
call apoc.path.subgraphAll(n,{maxLevel:1, relationshipFilter: 'Bind|Interaction'})
yield nodes ,relationships
return nodes ,relationships
```