# Environment Setup Guide

Prerequisites
- Install Java 8 or Java 11

Download Cassandra
- https://cassandra.apache.org/_/download.html

Configuration
- In conf/cassandra.yaml

Getting started
- http://cassandra.apache.org/doc/latest/getting_started/index.html

Python Driver (DataStax Cassandra Driver)
- https://docs.datastax.com/en/developer/python-driver/

# Detail steps for setup (Based on MacOS)

1. Download, extract and move Cassandra
Download the latest stable release: apache-cassandra-4.1.8-bin.tar
Extract: tar -xvzf apache-cassandra-4.1.8-bin.tar
Move the extracted folder to a permanent location: ex sudo mv apache-cassandra-4.1.8 /opt/cassandra
Nevigate to the Cassandra directory

2. Setup environment variables
Open your shell configuration file: sudo nano ~/.zshrc
Add the following lines at the bottom:

```
export CASSANDRA_HOME=/opt/cassandra
export PATH=$CASSANDRA_HOME/bin:$PATH
```

Save and exit
Reload the terminal configuration: source ~/.zshrc

3. Start Cassandra

```
study_cassandra % cassandra -f (foreground)
study_cassandra % cassandra (background)
```

nodetool status (check if Cassandra is running): If you see UN(Up/Normal), Cassandra is running successfully

```
study_cassandra % nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns (effective)  Host ID                               Rack
UN  127.0.0.1  75.75 KiB  16      100.0%            c0ed76ed-ae82-46da-a767-e14d6269fa60  rack1
```

4. Connect to Cassandra by CQL shell (Cassandra Query Language shell) to interact with the database

```
study_cassandra % cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.1.0 | Cassandra 4.1.8 | CQL spec 3.4.6 | Native
protocol v5]
Use HELP for help.
cqlsh> exit
```

5. Stop Cassandra

```
study_cassandra % pkill -f cassandra
```

or

```
study_cassandra % nodetool drain && pkill -f cassandra
```

6. Install the Cassandra Python Driver

```
study_cassandra % pip install cassandra-driver
```

or

```
study_cassandra % conda install -c conda-forge cassandra-driver
```

7. Verify the Python Driver installation

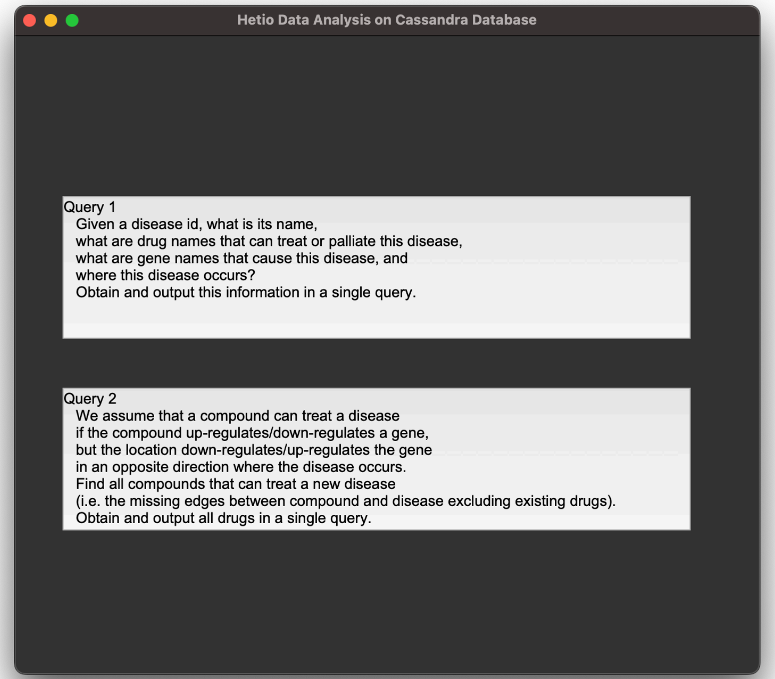
```
study_cassandra % python
Python 3.9.21 | packaged by conda-forge | (main, Dec  5 2024,
13:47:18)
[Clang 18.1.8 ] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import cassandra
>>> print(cassandra.__version__)
3.29.2
>>>
```

# Run python script (python 3.9, tkinter 8.6)

1. Run python script for GUI:

```
study_cassandra % python hetio_cassandra.py
```

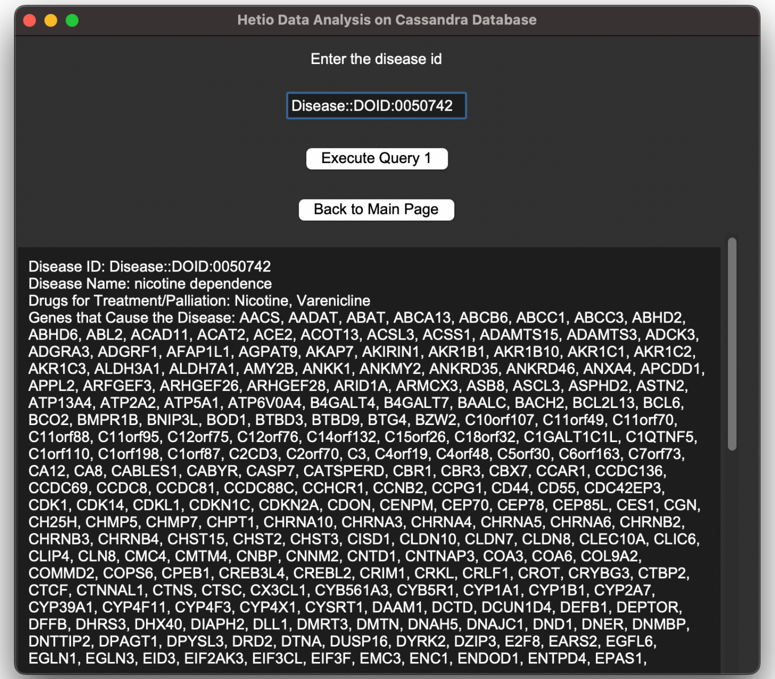
2. Result file:
Test result files (cassandra_query1.txt and cassandra_query2.txt) will be stored under test_results directory



```
Query 1
  Given a disease id, what is its name,
  what are drug names that can treat or palliate this disease,
  what are gene names that cause this disease, and
  where this disease occurs?
  Obtain and output this information in a single query.
```

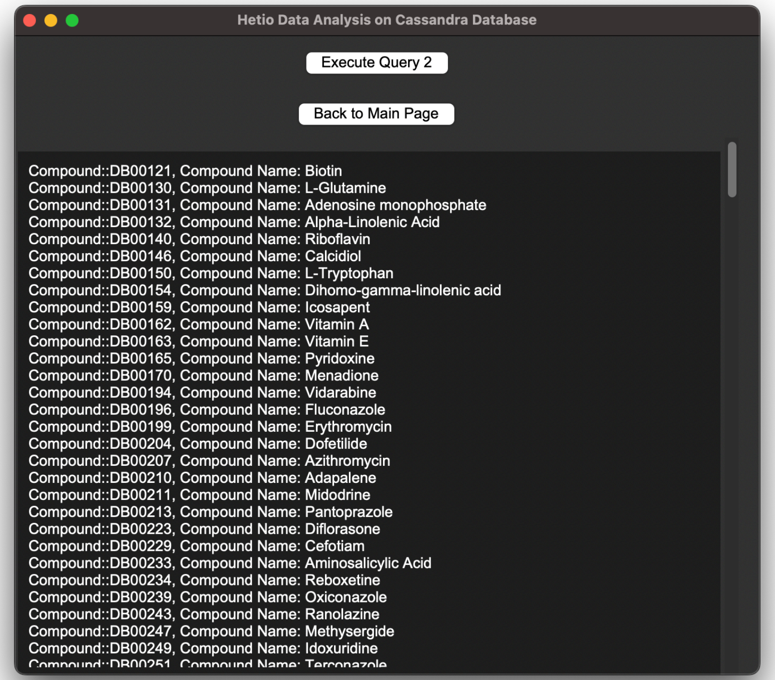
```
Query 2
  We assume that a compound can treat a disease
  if the compound up-regulates/down-regulates a gene,
  but the location down-regulates/up-regulates the gene
  in an opposite direction where the disease occurs.
  Find all compounds that can treat a new disease
  (i.e. the missing edges between compound and disease excluding existing drugs).
  Obtain and output all drugs in a single query.
```

Initial Page



```
Disease ID: Disease::DOID:0050742
Disease Name: nicotine dependence
Drugs for Treatment/Palliation: Nicotine, Varenicline
Genes that Cause the Disease: AACS, AADAT, ABAT, ABCA13, ABCB6, ABCC1, ABCC3, ABHD2,
ABHD6, ABL2, ACAD11, ACAT2, ACE2, ACOT13, ACSL3, ACSS1, ADAMTS15, ADAMTS3, ADCK3,
ADGRA3, ADGRF1, AFAP1L1, AGPAT9, AKAP7, AKIRIN1, AKR1B1, AKR1B10, AKR1C1, AKR1C2,
AKR1C3, ALDH3A1, ALDH7A1, AMY2B, ANKK1, ANKMY2, ANKRD35, ANKRD46, ANXA4, APCDD1,
APPL2, ARFGEF3, ARHGEF26, ARHGEF28, ARID1A, ARMCX3, ASB8, ASCL3, ASPHD2, ASTN2,
ATP13A4, ATP2A2, ATP5A1, ATP6V0A4, B4GALT4, B4GALT7, BAALC, BACH2, BCL2L13, BCL6,
BCO2, BMPR1B, BNIP3L, BOD1, BTBD3, BTBD9, BTG4, BZW2, C10orf107, C11orf49, C11orf70,
C11orf88, C11orf95, C12orf75, C12orf76, C14orf132, C15orf26, C18orf32, C1GALT1C1L, C1QTNF5,
C1orf110, C1orf198, C1orf87, C2CD3, C2orf70, C3, C4orf19, C4orf48, C5orf30, C6orf163, C7orf73,
CA12, CA8, CABLES1, CABYR, CASP7, CATSPERD, CBR1, CBR3, CBX7, CCAR1, CCDC136,
CCDC69, CCDC8, CCDC81, CCDC88C, CCHCR1, CCNB2, CCPG1, CD44, CD55, CDC42EP3,
CDK1, CDK14, CDKL1, CDKN1C, CDKN2A, CDON, CENPM, CEP70, CEP78, CEP85L, CES1, CGN,
CH25H, CHMP5, CHMP7, CHPT1, CHRNA10, CHRNA3, CHRNA4, CHRNA5, CHRNA6, CHRNB2,
CHRNB3, CHRNB4, CHST15, CHST2, CHST3, CISD1, CLDN10, CLDN7, CLDN8, CLEC10A, CLIC6,
CLIP4, CLN8, CMC4, CMTM4, CNBP, CNNM2, CNTD1, CNTNAP3, COA3, COA6, COL9A2,
COMMD2, COPS6, CPEB1, CREB3L4, CREBL2, CRIM1, CRKL, CRLF1, CROT, CRYBG3, CTBP2,
CTCF, CTNNAL1, CTNS, CTSC, CX3CL1, CYB561A3, CYB5R1, CYP1A1, CYP1B1, CYP2A7,
CYP39A1, CYP4F11, CYP4F3, CYP4X1, CYSRT1, DAAM1, DCTD, DCUN1D4, DEFB1, DEPTOR,
DFFB, DHRS3, DHX40, DIAPH2, DLL1, DMRT3, DMTN, DNAH5, DNAJC1, DND1, DNER, DNMBP,
DNTTIP2, DPAGT1, DPYSL3, DRD2, DTNA, DUSP16, DYRK2, DZIP3, E2F8, EARS2, EGFL6,
EGLN1, EGLN3, EID3, EIF2AK3, EIF3CL, EIF3F, EMC3, ENC1, ENDOD1, ENTPD4, EPAS1,
```

Query 1 Pagae

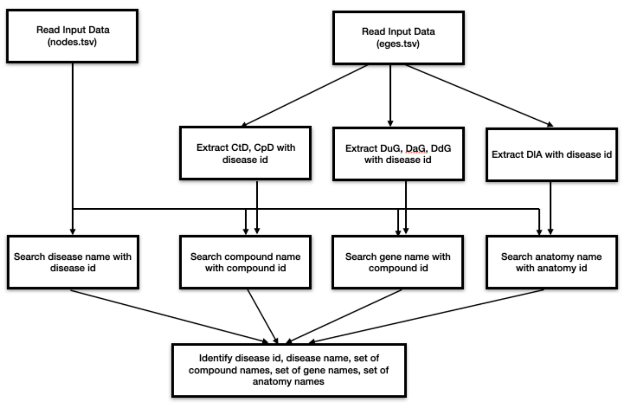


```
Compound::DB00121, Compound Name: Biotin
Compound::DB00130, Compound Name: L-Glutamine
Compound::DB00131, Compound Name: Adenosine monophosphate
Compound::DB00132, Compound Name: Alpha-Linolenic Acid
Compound::DB00140, Compound Name: Riboflavin
Compound::DB00146, Compound Name: Calcidiol
Compound::DB00150, Compound Name: L-Tryptophan
Compound::DB00154, Compound Name: Dihomo-gamma-linolenic acid
Compound::DB00159, Compound Name: Icosapent
Compound::DB00162, Compound Name: Vitamin A
Compound::DB00163, Compound Name: Vitamin E
Compound::DB00165, Compound Name: Pyridoxine
Compound::DB00170, Compound Name: Menadione
Compound::DB00194, Compound Name: Vidarabine
Compound::DB00196, Compound Name: Fluconazole
Compound::DB00199, Compound Name: Erythromycin
Compound::DB00204, Compound Name: Dofetilide
Compound::DB00207, Compound Name: Azithromycin
Compound::DB00210, Compound Name: Adapalene
Compound::DB00211, Compound Name: Midodrine
Compound::DB00213, Compound Name: Pantoprazole
Compound::DB00223, Compound Name: Diflorasone
Compound::DB00229, Compound Name: Cefotiam
Compound::DB00233, Compound Name: Aminosalicylic Acid
Compound::DB00234, Compound Name: Reboxetine
Compound::DB00239, Compound Name: Oxiconazole
Compound::DB00243, Compound Name: Ranolazine
Compound::DB00247, Compound Name: Methysergide
Compound::DB00249, Compound Name: Idoxuridine
Compound::DB00251, Compound Name: Terconazole
```

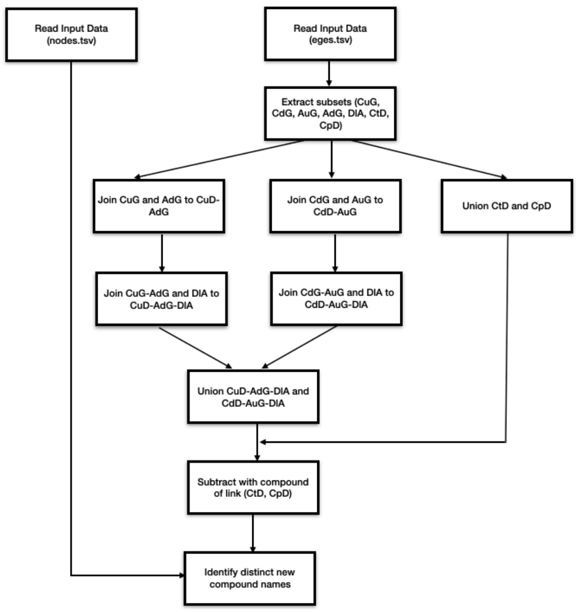
Query 2 Page

# Design diagram

Design diagram: Query 1


Read Input Data (nodes.tsv)
Read Input Data (eges.tsv)
Extract CtD, CpD with disease id
Extract DuG, DaG, DdG with disease id
Extract DIA with disease id
Search disease name with disease id
Search compound name with compound id
Search gene name with compound id
Search anatomy name with anatomy id
Identify disease id, disease name, set of compound names, set of gene names, set of anatomy names

Design diagram: Query 2


Read Input Data (nodes.tsv)
Read Input Data (eges.tsv)
Extract subsets (CuG, CdG, AuG, AdG, DIA, CtD, CpD)
Join CuG and AdG to CuD-AdG
Join CdG and AuG to CdD-AuG
Union CtD and CpD
Join CuG-AdG and DIA to CuD-AdG-DIA
Join CdG-AuG and DIA to CdD-AuG-DIA
Union CuD-AdG-DIA and CdD-AuG-DIA
Subtract with compound of link (CtD, CpD)
Identify distinct new compound names

Design Diagram

```
# Cassandra database for HetioNet
1. HetioNet
node data: nodes.tsv
edge data: edges.tsv

2. keyspace: hetio_db
    session.execute("""
        CREATE KEYSPACE IF NOT EXISTS hetio_db
        WITH replication = {'class': 'SimpleStrategy',
'replication_factor': '1'};
    """)

3. tables:
disease_info
    session.execute("""
        CREATE TABLE IF NOT EXISTS disease_info (
            disease_id TEXT PRIMARY KEY,
            disease_name TEXT,
            drug_names SET<TEXT>,
            gene_names SET<TEXT>,
            location_names SET<TEXT>
        );
    """)

    session.execute("""
            INSERT INTO disease_info (disease_id, disease_name,
drug_names, gene_names, location_names)
            VALUES (%s, %s, %s, %s, %s);
        """, (disease_id, disease_name, drugs, genes,
locations))

compound_info
    session.execute("""
        CREATE TABLE IF NOT EXISTS compound_info (
            compound_id TEXT PRIMARY KEY,
            compound_name TEXT,
            is_connected_with_disease BOOLEAN
        );
    """)

    session.execute("""
            INSERT INTO compound_info (compound_id,
compound_name, is_connected_with_disease)
            VALUES (%s, %s, %s);
        """, (compound_id, drugs_names[compound_id], False))
    session.execute("""
            INSERT INTO compound_info (compound_id,
compound_name, is_connected_with_disease)
            VALUES (%s, %s, %s);
        """, (compound_id, drugs_names[compound_id], True))



# Queries for Cassandra
Query 1:
    Given a disease id, what is its name,
    what are drug names that can treat or palliate this disease,
    what are gene names that cause this disease, and
    where this disease occurs?
    Obtain and output this information in a single query.

    session.execute("SELECT * FROM disease_info WHERE disease_id
= %s", [disease_id])

Query 2:
    We assume that a compound can treat a disease
    if the compound up-regulates/down-regulates a gene,
    but the location down-regulates/up-regulates the gene
    in an opposite direction where the disease occurs.
    Find all compounds that can treat a new disease
    (i.e. the missing edges between compound and disease
excluding existing drugs).
    Obtain and output all drugs in a single query.

    session.execute("SELECT compound_id, compound_name FROM
compound_info WHERE is_connected_with_disease = %s ALLOW
FILTERING", [False])



# Potential improvement
1. Since Cassandra does not support joins, we use pre-processing
functions with basic Python data structures like lists and sets.
This could be improved by utilizing Pandas' merge function for
more efficient joins.
See test_alternative_approach_with_pandas.py under test
directory.
2. The current column family (table) structure is simple and
optimized for two specific queries.
By storing a Pandas DataFrame after joining edge and node
information, we could create a more flexible and generic table
structure.
```

3. We will explore alternative methods that reduce dependency on Python.