**PRACTICAL 2**

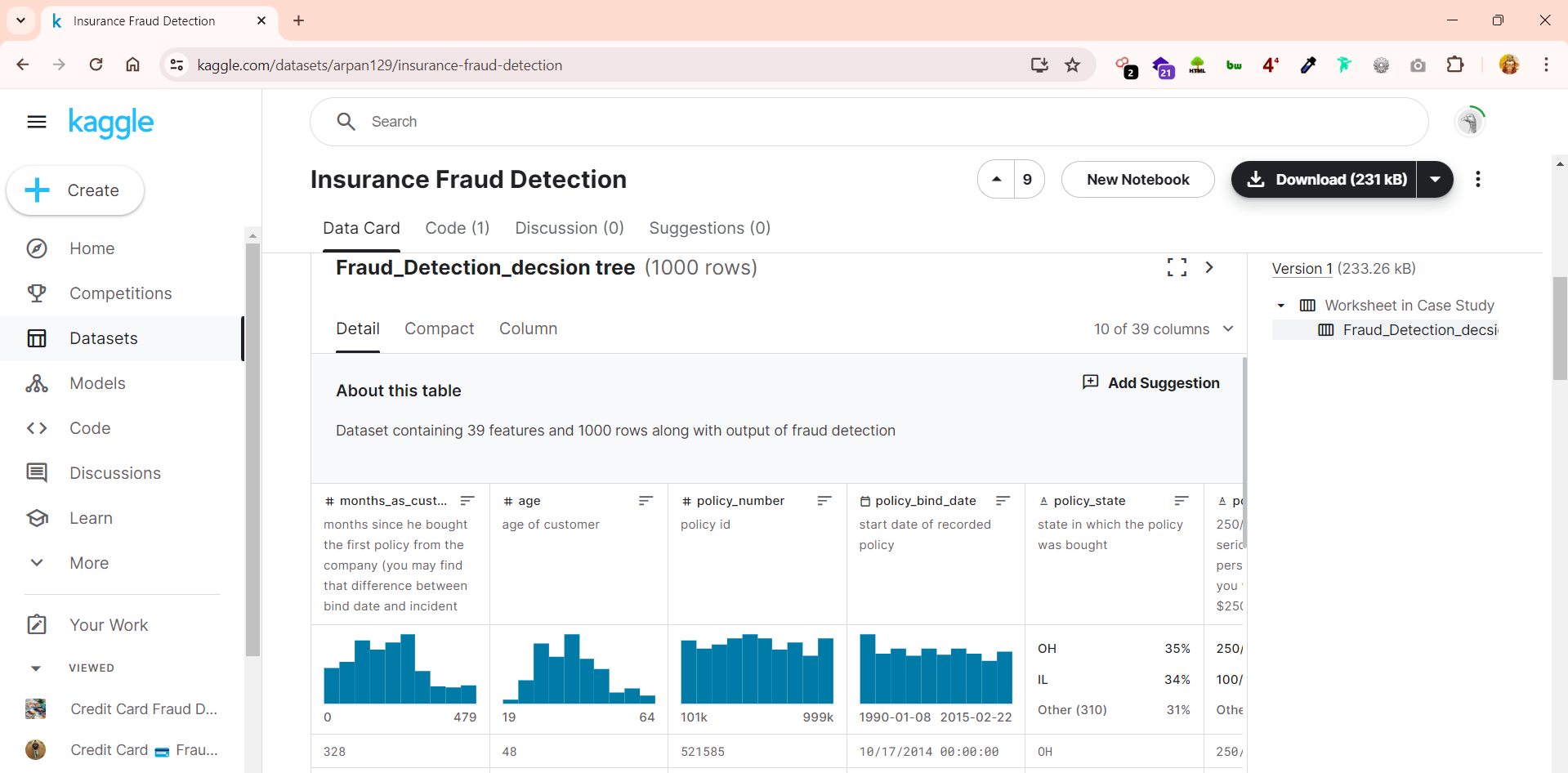
**INSURANCE FRAUD DETECTION AND PREDICTION USING VARIOUS MACHINE LEARNING ALGORITHMS AND NEO4J VISUALIZATION**

* **LOGISTIC REGRESSION**
* **DECISION TREE**
* **RANDOM FOREST**
* **GRADIENT BOOSTING**

**[A] DOWNLOAD DATASET**

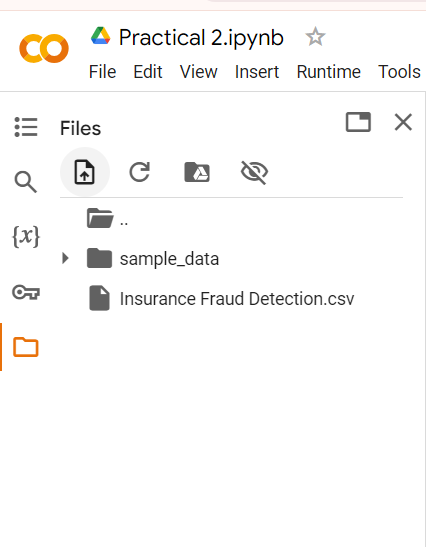
Download the dataset from kaggle

<https://www.kaggle.com/datasets/arpan129/insurance-fraud-detection>



**[B] IMPLEMENTING ML ALGORITHMS FOR PREDICTIONS**

Upload the csv file into colab files and copy its path clicking on **:**



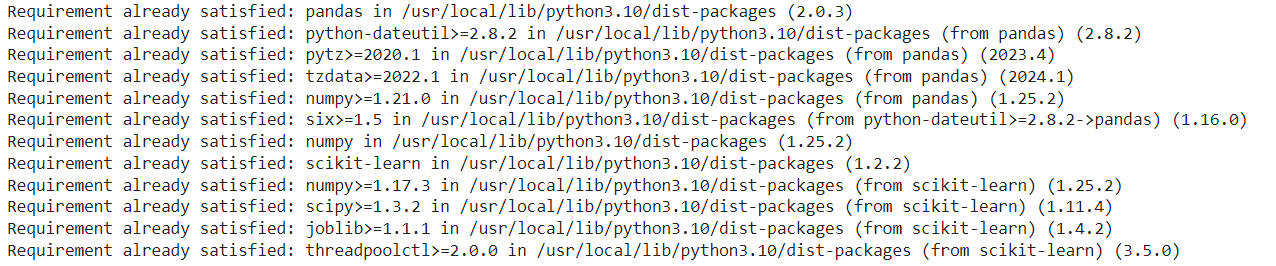
**Data Preprocessing**

* Install packages:

!pip install pandas

!pip install numpy

!pip install scikit-learn



* Load the dataset and libraries:

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

# Load the dataset

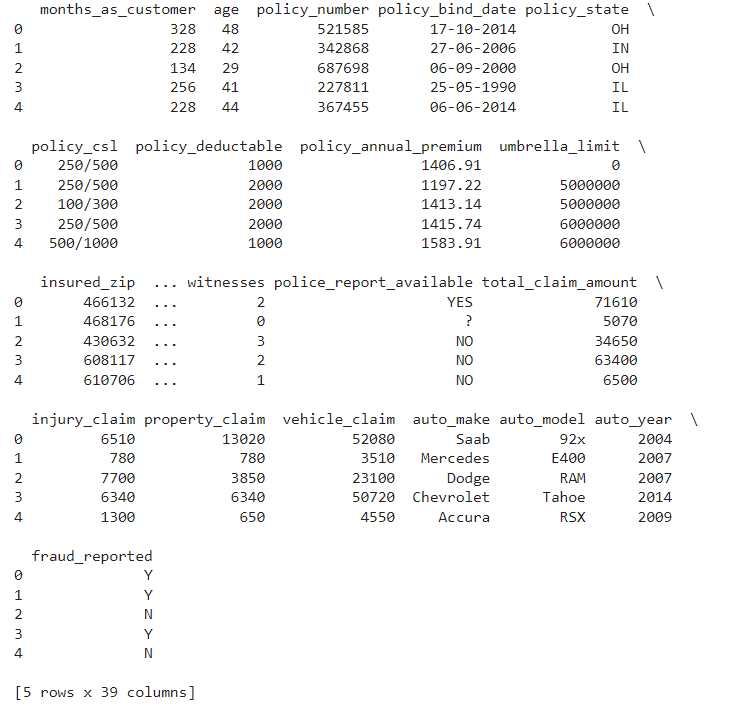
data = pd.read\_csv('/content/Insurance Fraud Detection.csv')

# Display the first few rows of the dataset

print(data.head())

# Copy the original data for export

original\_data = data.copy()



* Handle missing values and encode categorical variables:

# fill missing values

data = data.fillna('NA')

# encode categorical variables

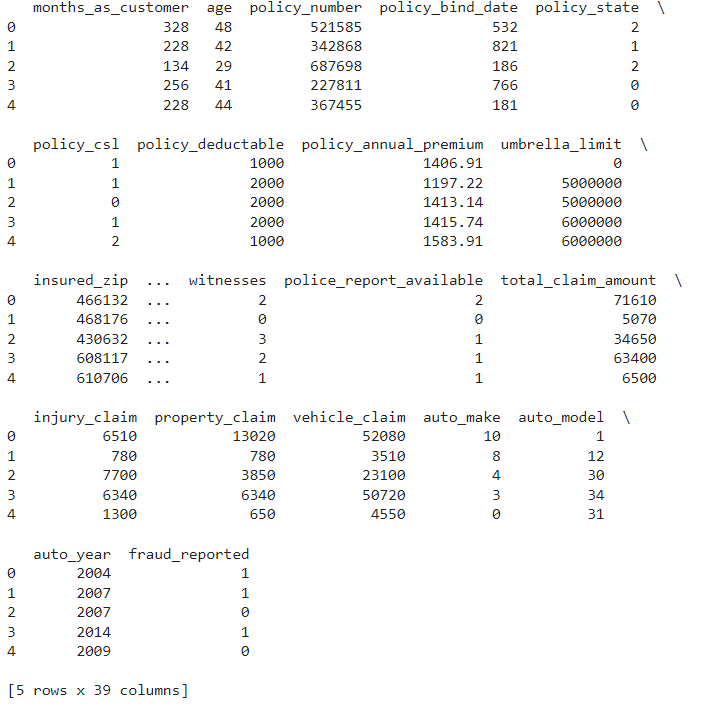
le = LabelEncoder()

for column in data.select\_dtypes(include=['object']).columns:

    data[column] = le.fit\_transform(data[column])

# dataset after preprocessing

print(data.head())



* Split the dataset into features and target variable:

# features and target variable

X = data.drop(columns=['fraud\_reported'])

y = data['fraud\_reported']

# Spliting the data to train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

**Model Training**

* Logistic Regression:

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report

# Train logistic regression model

lr\_model = LogisticRegression(max\_iter=1000)

lr\_model.fit(X\_train, y\_train)

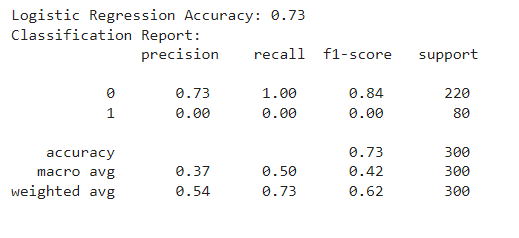
# Predict and evaluate

y\_pred\_lr = lr\_model.predict(X\_test)

lr\_accuracy = accuracy\_score(y\_test, y\_pred\_lr)

print("Logistic Regression Accuracy:", lr\_accuracy)

print("Classification Report:\n", classification\_report(y\_test, y\_pred\_lr))



* Decision Tree:

from sklearn.tree import DecisionTreeClassifier

# Train decision tree model

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

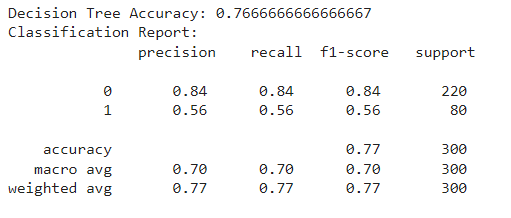
# Predict and evaluate

y\_pred\_dt = dt\_model.predict(X\_test)

dt\_accuracy = accuracy\_score(y\_test, y\_pred\_dt)

print("Decision Tree Accuracy:", dt\_accuracy)

print("Classification Report:\n", classification\_report(y\_test, y\_pred\_dt))



* Random Forest:

from sklearn.ensemble import RandomForestClassifier

# Train random forest model

rf\_model = RandomForestClassifier(random\_state=42)

rf\_model.fit(X\_train, y\_train)

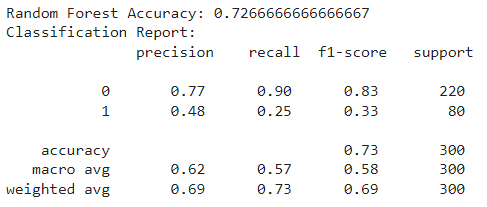
# Predict and evaluate

y\_pred\_rf = rf\_model.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, y\_pred\_rf)

print("Random Forest Accuracy:", rf\_accuracy)

print("Classification Report:\n", classification\_report(y\_test, y\_pred\_rf))



* Gradient Boosting:

from sklearn.ensemble import GradientBoostingClassifier

# Train gradient boosting model

gb\_model = GradientBoostingClassifier(random\_state=42)

gb\_model.fit(X\_train, y\_train)

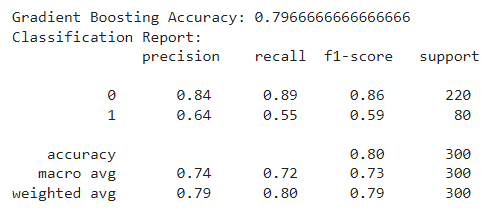
# Predict and evaluate

y\_pred\_gb = gb\_model.predict(X\_test)

gb\_accuracy = accuracy\_score(y\_test, y\_pred\_gb)

print("Gradient Boosting Accuracy:", gb\_accuracy)

print("Classification Report:\n", classification\_report(y\_test, y\_pred\_gb))



**Compare Models and Select the Best**

# Define models with accuracy scores

models = {

    "Logistic Regression": lr\_accuracy,

    "Decision Tree": dt\_accuracy,

    "Random Forest": rf\_accuracy,

    "Gradient Boosting": gb\_accuracy

}

# best model

best\_model\_name = max(models, key=models.get)

best\_model\_accuracy = models[best\_model\_name]

print(f"Best model: {best\_model\_name} with accuracy {best\_model\_accuracy}")

# best model's predictions

if best\_model\_name == "Logistic Regression":

    best\_model = lr\_model

elif best\_model\_name == "Decision Tree":

    best\_model = dt\_model

elif best\_model\_name == "Random Forest":

    best\_model = rf\_model

else:

    best\_model = gb\_model



**Exporting Data**

# predictions for the entire dataset using the best model

original\_data['prediction'] = best\_model.predict(data.drop(columns=['fraud\_reported']))

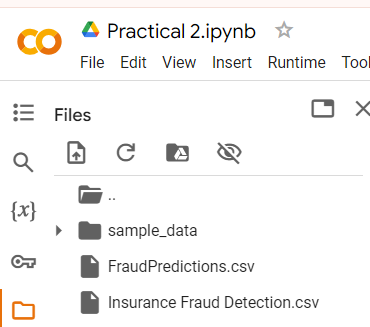
# Export the data with predictions

original\_data.to\_csv('FraudPredictions.csv', index=False)

print("Data exported to FraudPredictions.csv")

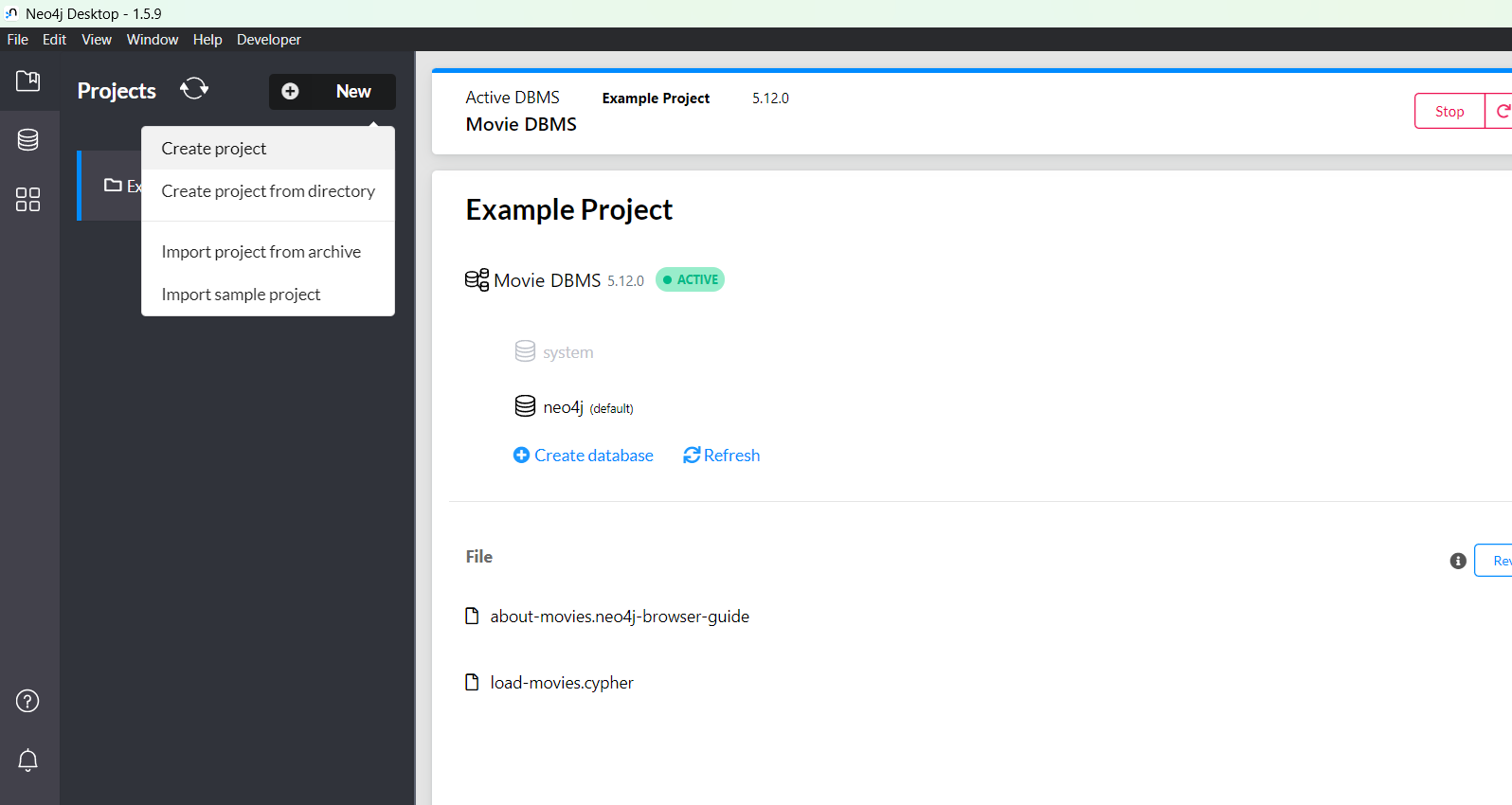


Download the FraudPredictions.csv after the export is completed (It will be saved in the same directory where the dataset was)

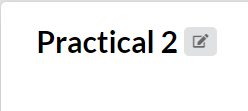


**[C] NEO4J VISUALIZATION ON PREDICTED DATA**

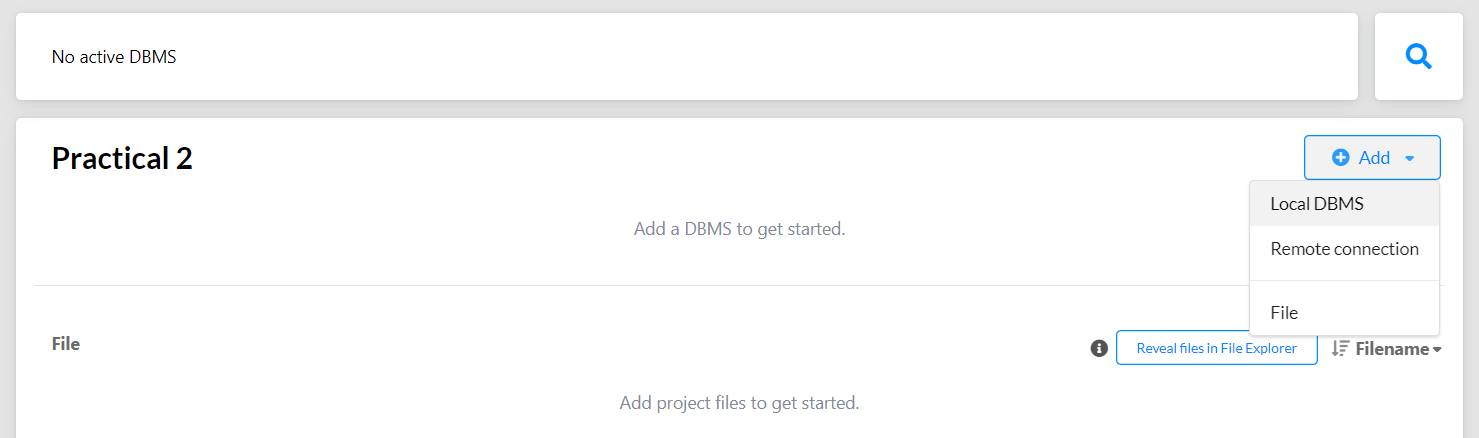
1. Click on the New panel and create a new project



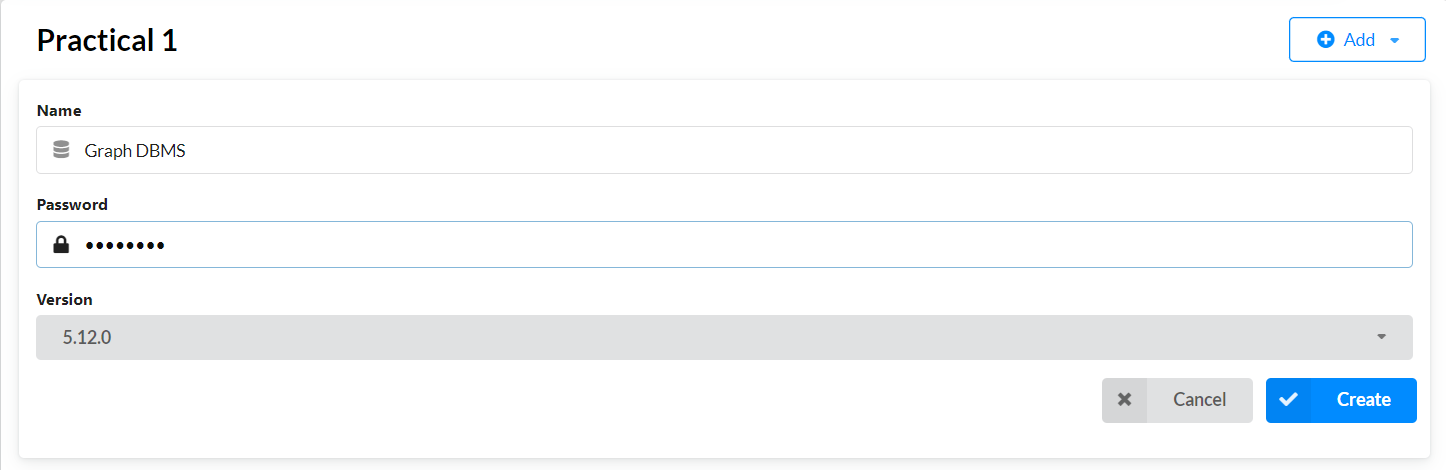
1. Rename the project to Practical 2



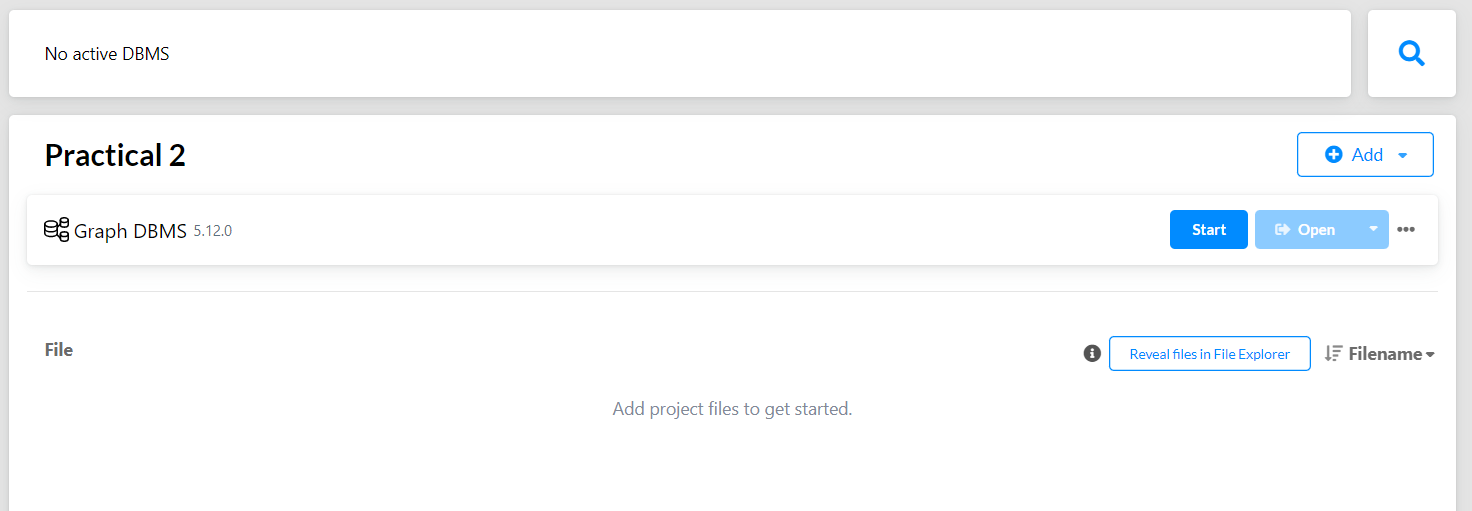
1. In the right side click on add panel and select local dbms



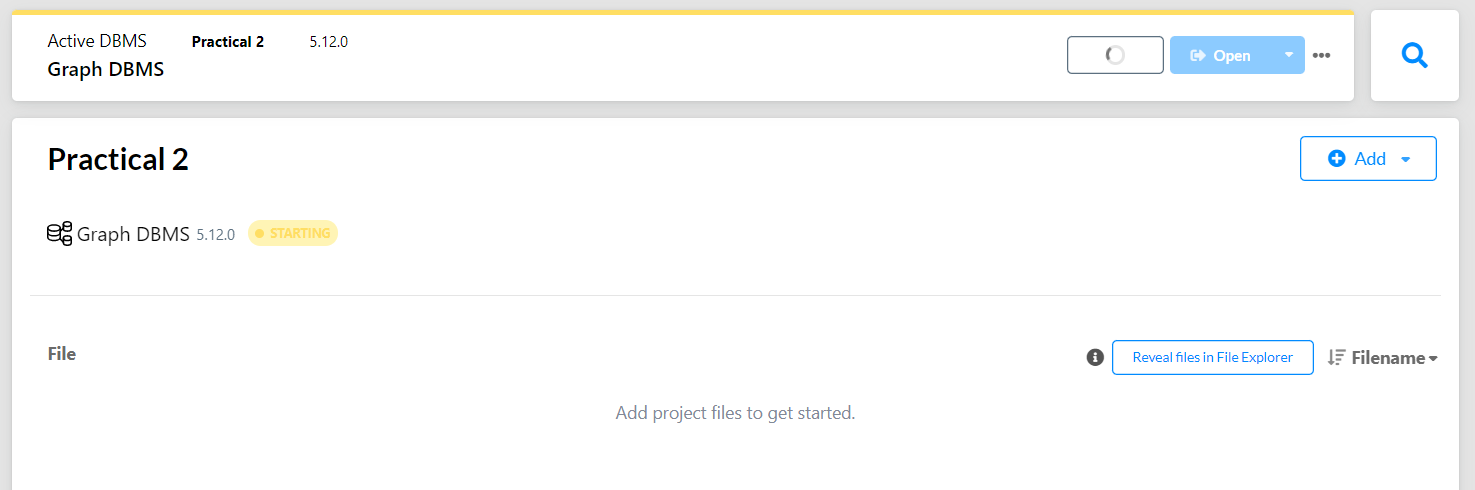
1. Set a password with eight characters long and click on create and wait till the dbms gets created



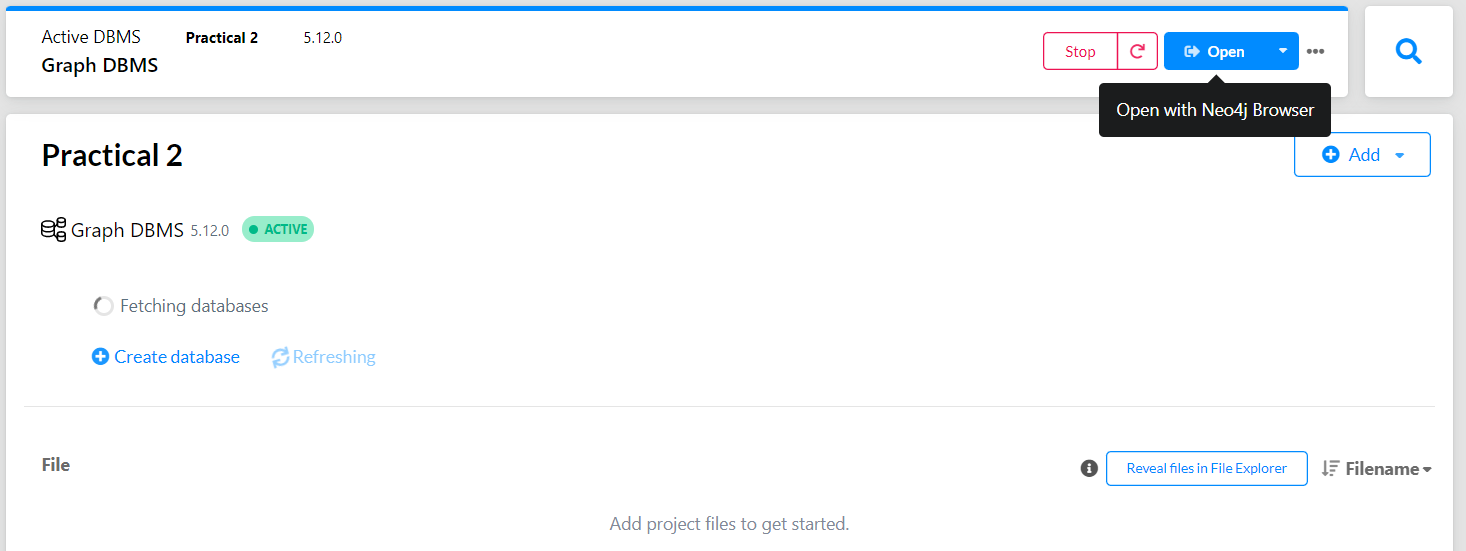
1. Now click on start button to strat the dbms



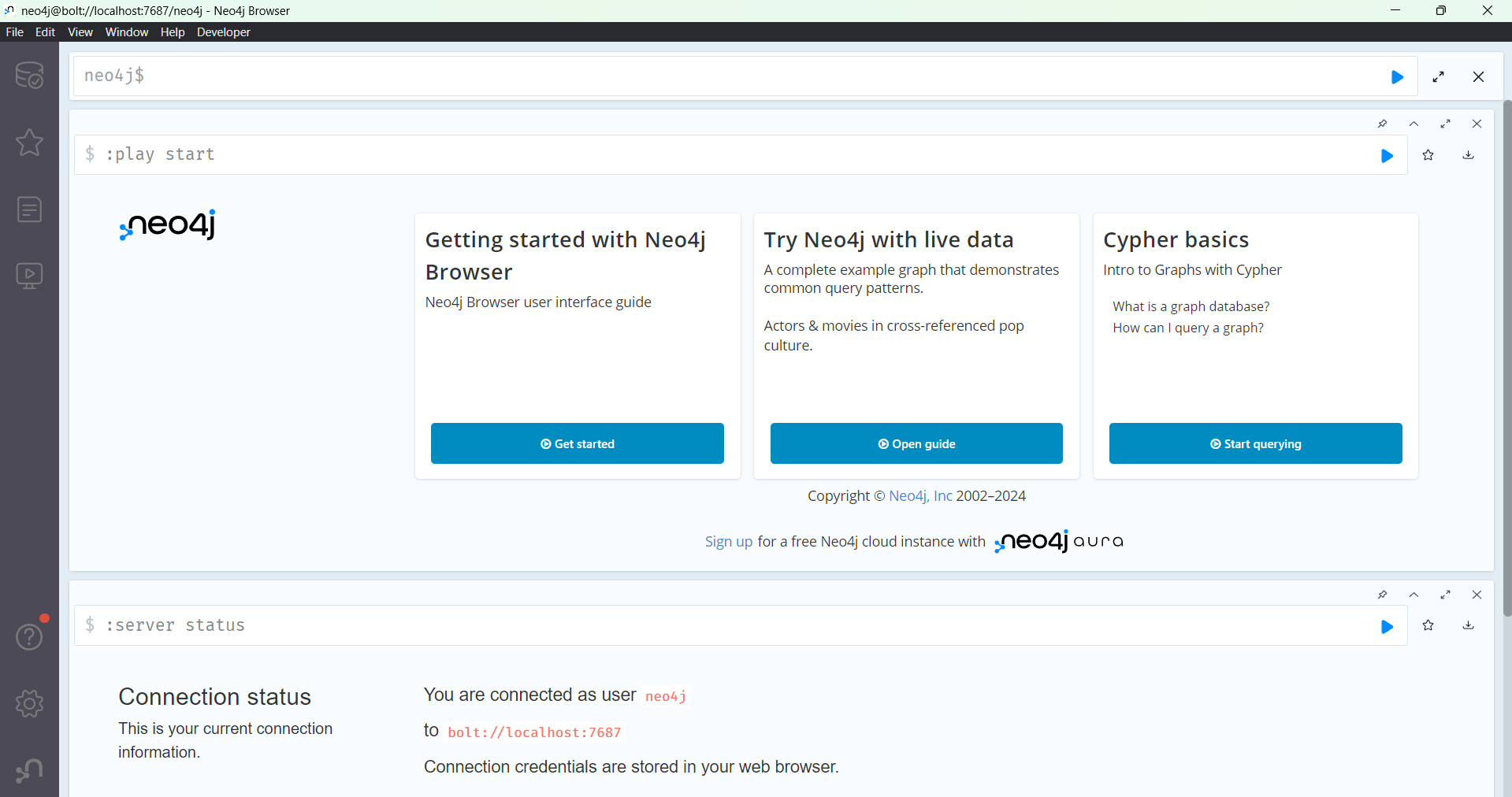
1. A terminal will open do not close it otherwise dbms will get stopped



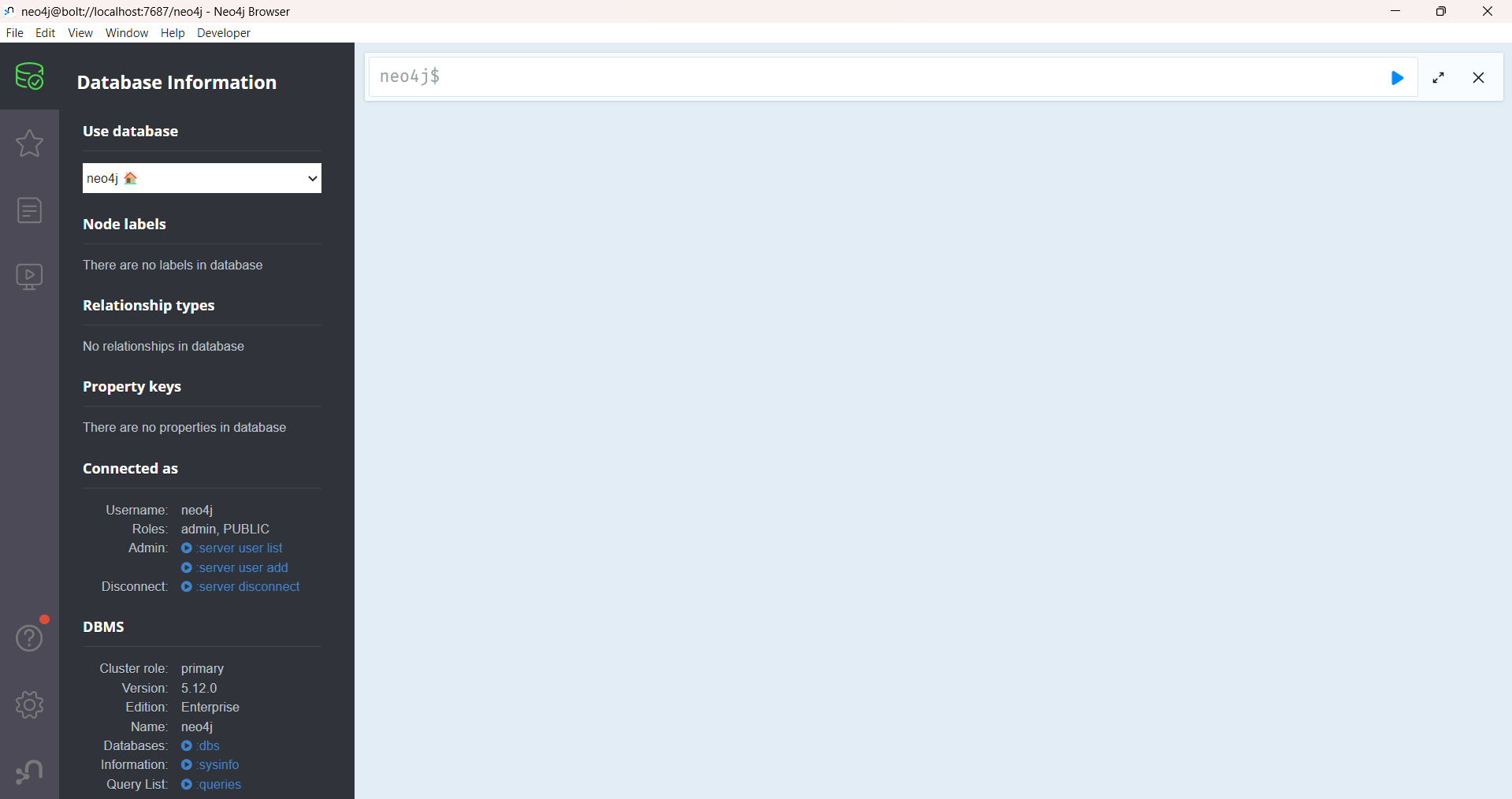
1. If the dbms gets started Click on open to start the neo4j browser on the localhost



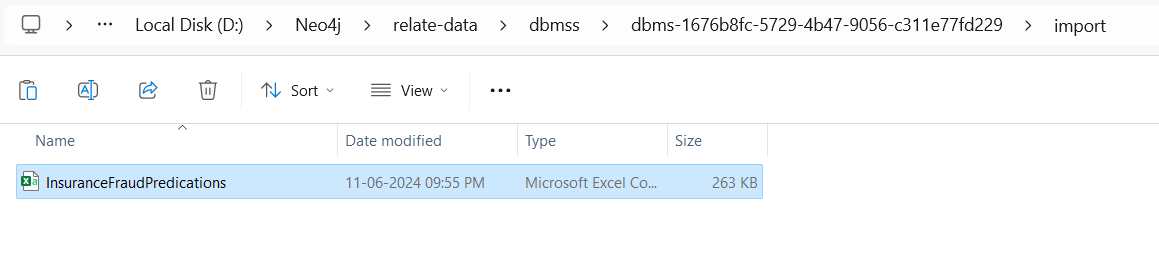
1. Neo4j Browser is opened if no issues detected



1. Clear all the cypher queries in the browser



1. Now copy and paste your processed InsuranceFraudPredications.csv file into your project import directory (/Neo4j/relate-data/dbmss/dbms-projectid /import/)



1. Get back to your neo4j browser and start your cypher query, first will import the dataset and assign the datatype of the columns
2. Load CSV and create Transaction nodes

LOAD CSV WITH HEADERS FROM 'file:///InsuranceFraudPredications.csv' AS row

CREATE (t:Transaction {

    months\_as\_customer: toInteger(row.months\_as\_customer),

    age: toInteger(row.age),

    policy\_number: toInteger(row.policy\_number),

    policy\_bind\_date: row.policy\_bind\_date,

    policy\_state: row.policy\_state,

    policy\_csl: row.policy\_csl,

    policy\_deductable: toInteger(row.policy\_deductable),

    policy\_annual\_premium: toFloat(row.policy\_annual\_premium),

    umbrella\_limit: toInteger(row.umbrella\_limit),

    insured\_zip: toInteger(row.insured\_zip),

    insured\_sex: row.insured\_sex,

    insured\_education\_level: row.insured\_education\_level,

    insured\_occupation: row.insured\_occupation,

    insured\_hobbies: row.insured\_hobbies,

    insured\_relationship: row.insured\_relationship,

    capital\_gains: toInteger(row.capital\_gains),

    capital\_loss: toInteger(row.capital\_loss),

    incident\_date: row.incident\_date,

    incident\_type: row.incident\_type,

    collision\_type: row.collision\_type,

    incident\_severity: row.incident\_severity,

    authorities\_contacted: row.authorities\_contacted,

    incident\_state: row.incident\_state,

    incident\_city: row.incident\_city,

    incident\_location: row.incident\_location,

    incident\_hour\_of\_the\_day: toInteger(row.incident\_hour\_of\_the\_day),

    number\_of\_vehicles\_involved: toInteger(row.number\_of\_vehicles\_involved),

    property\_damage: row.property\_damage,

    bodily\_injuries: toInteger(row.bodily\_injuries),

    witnesses: toInteger(row.witnesses),

    police\_report\_available: row.police\_report\_available,

    total\_claim\_amount: toFloat(row.total\_claim\_amount),

    injury\_claim: toFloat(row.injury\_claim),

    property\_claim: toFloat(row.property\_claim),

    vehicle\_claim: toFloat(row.vehicle\_claim),

    auto\_make: row.auto\_make,

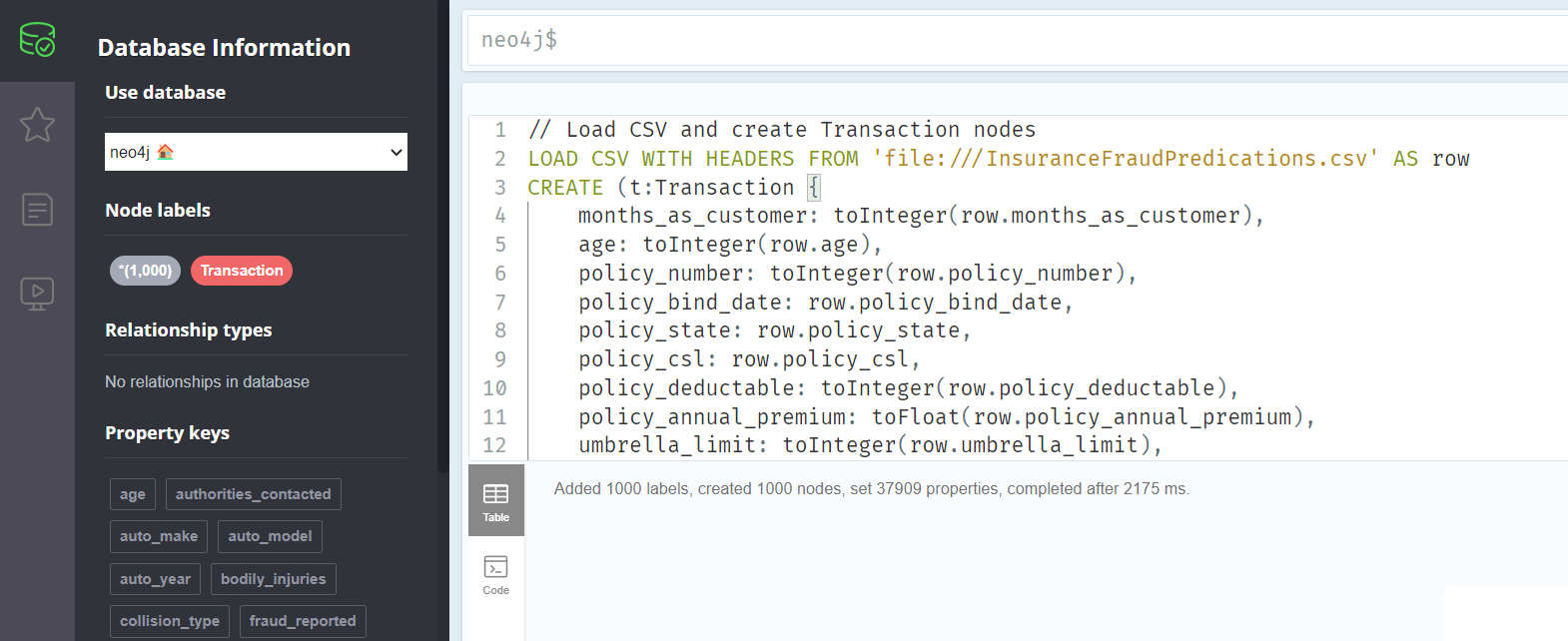
    auto\_model: row.auto\_model,

    auto\_year: toInteger(row.auto\_year),

    fraud\_reported: row.fraud\_reported,

    prediction: row.prediction

});



1. Create relationships based on specific policy state

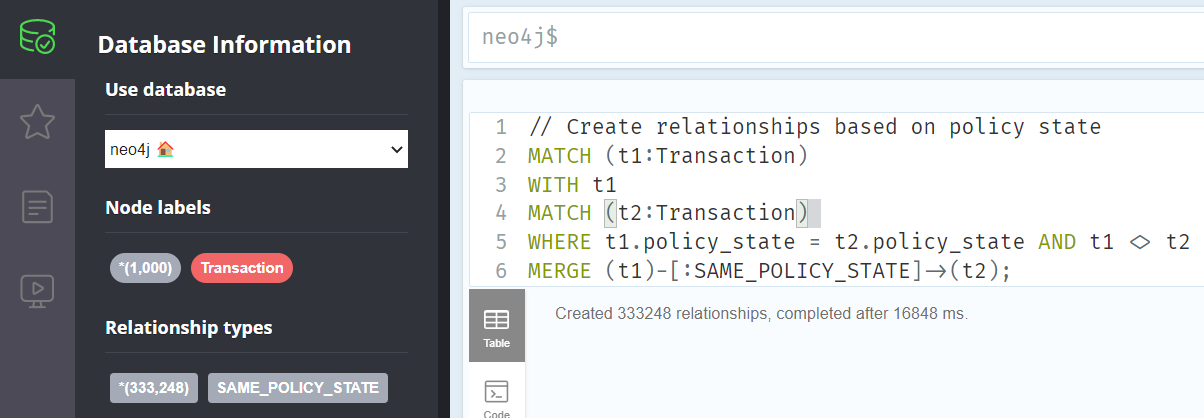
MATCH (t1:Transaction)

WITH t1

MATCH (t2:Transaction)

WHERE t1.policy\_state = t2.policy\_state AND t1 <> t2

MERGE (t1)-[:SAME\_POLICY\_STATE]->(t2);



1. Create relationships based on specific incident type

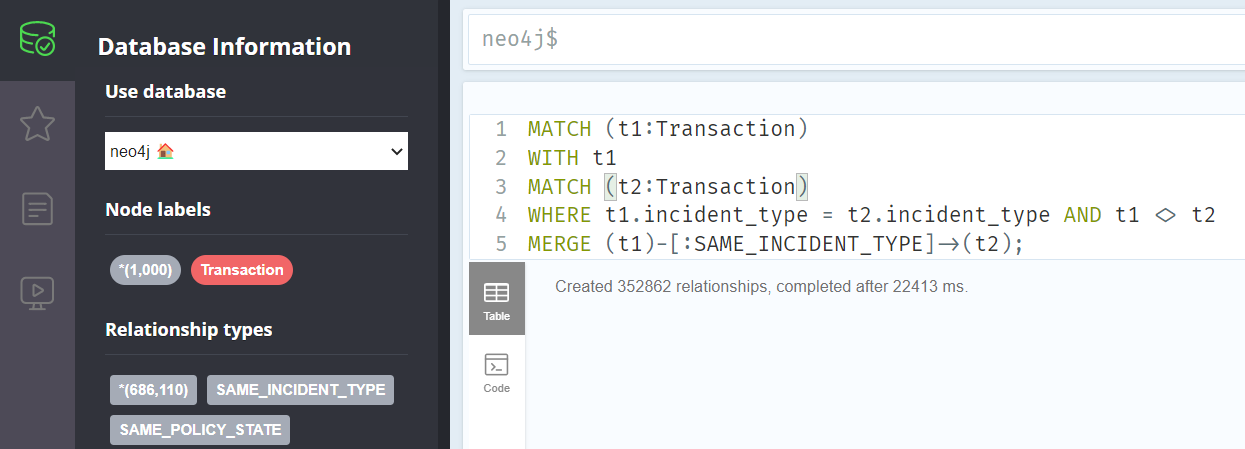
MATCH (t1:Transaction)

WITH t1

MATCH (t2:Transaction)

WHERE t1.incident\_type = t2.incident\_type AND t1 <> t2

MERGE (t1)-[:SAME\_INCIDENT\_TYPE]->(t2);



1. Create relationships based on specific auto make

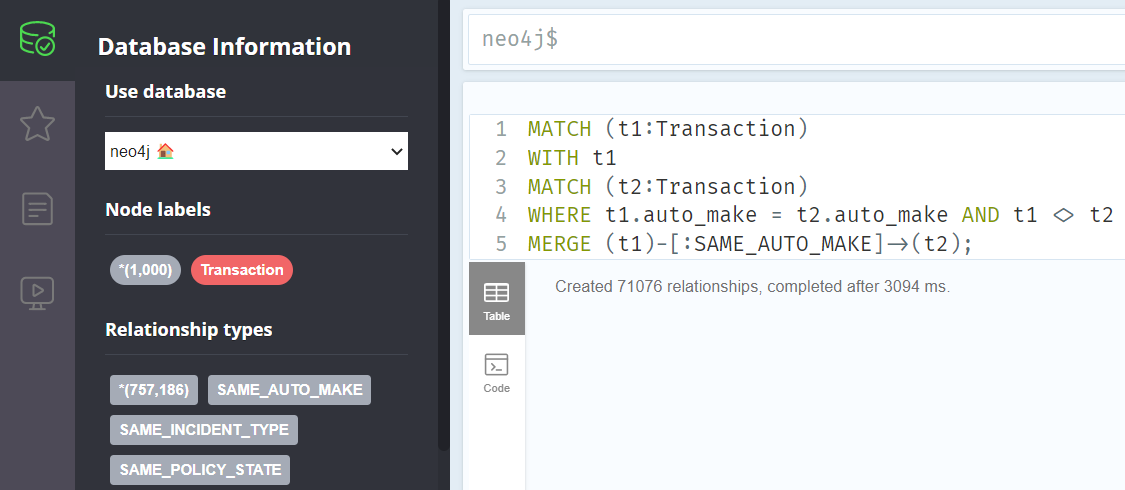
MATCH (t1:Transaction)

WITH t1

MATCH (t2:Transaction)

WHERE t1.auto\_make = t2.auto\_make AND t1 <> t2

MERGE (t1)-[:SAME\_AUTO\_MAKE]->(t2);

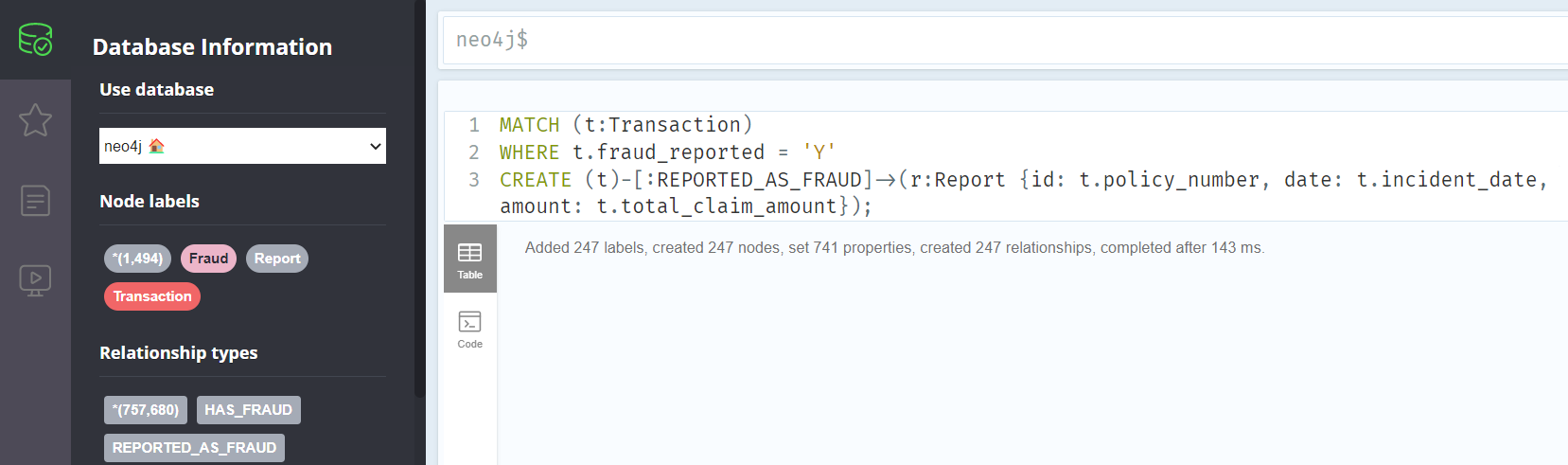


1. Create relationships for transactions reported as fraud

MATCH (t:Transaction)

WHERE t.fraud\_reported = 'Y'

CREATE (t)-[:REPORTED\_AS\_FRAUD]->(r:Report {id: t.policy\_number, date: t.incident\_date, amount: t.total\_claim\_amount});

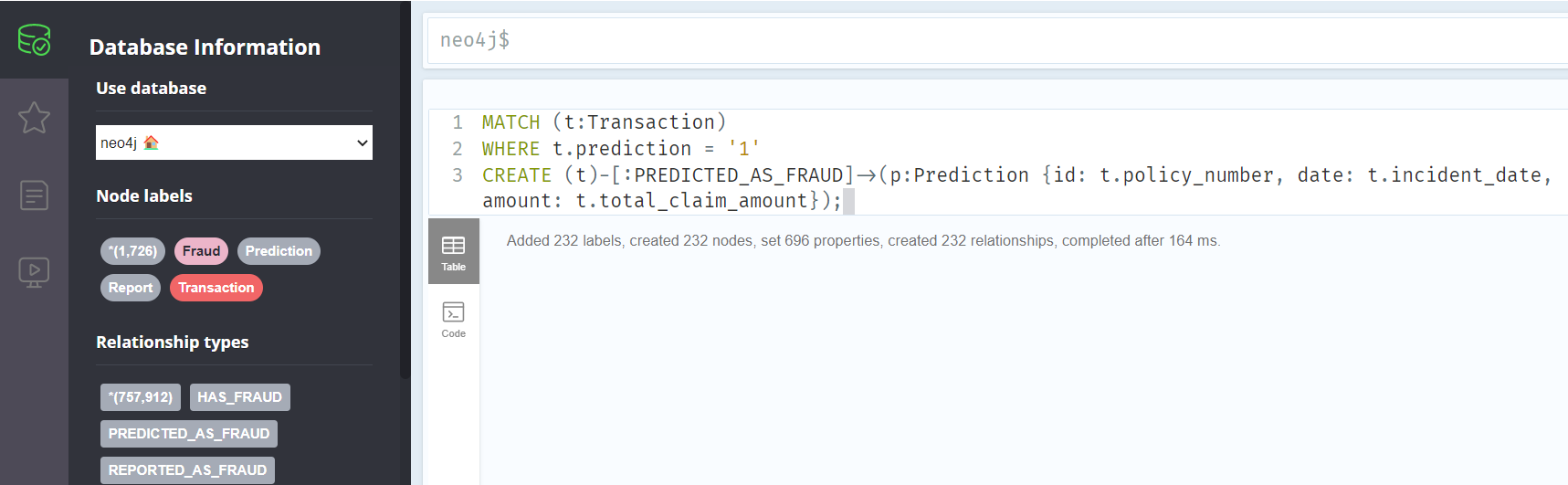


1. Create relationships for transactions predicted as fraud

MATCH (t:Transaction)

WHERE t.prediction = '1'

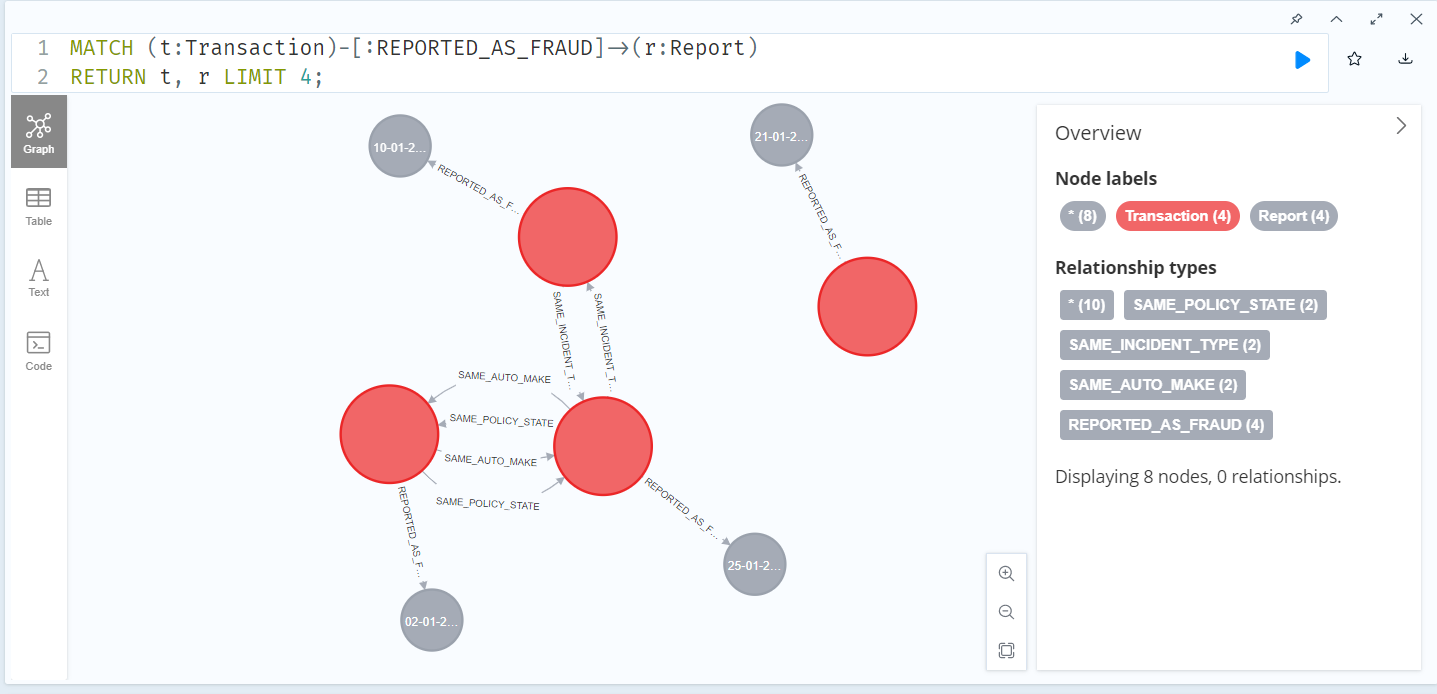
CREATE (t)-[:PREDICTED\_AS\_FRAUD]->(p:Prediction {id: t.policy\_number, date: t.incident\_date, amount: t.total\_claim\_amount});



1. View all transactions that were reported as fraud

MATCH (t:Transaction)-[:REPORTED\_AS\_FRAUD]->(r:Report)

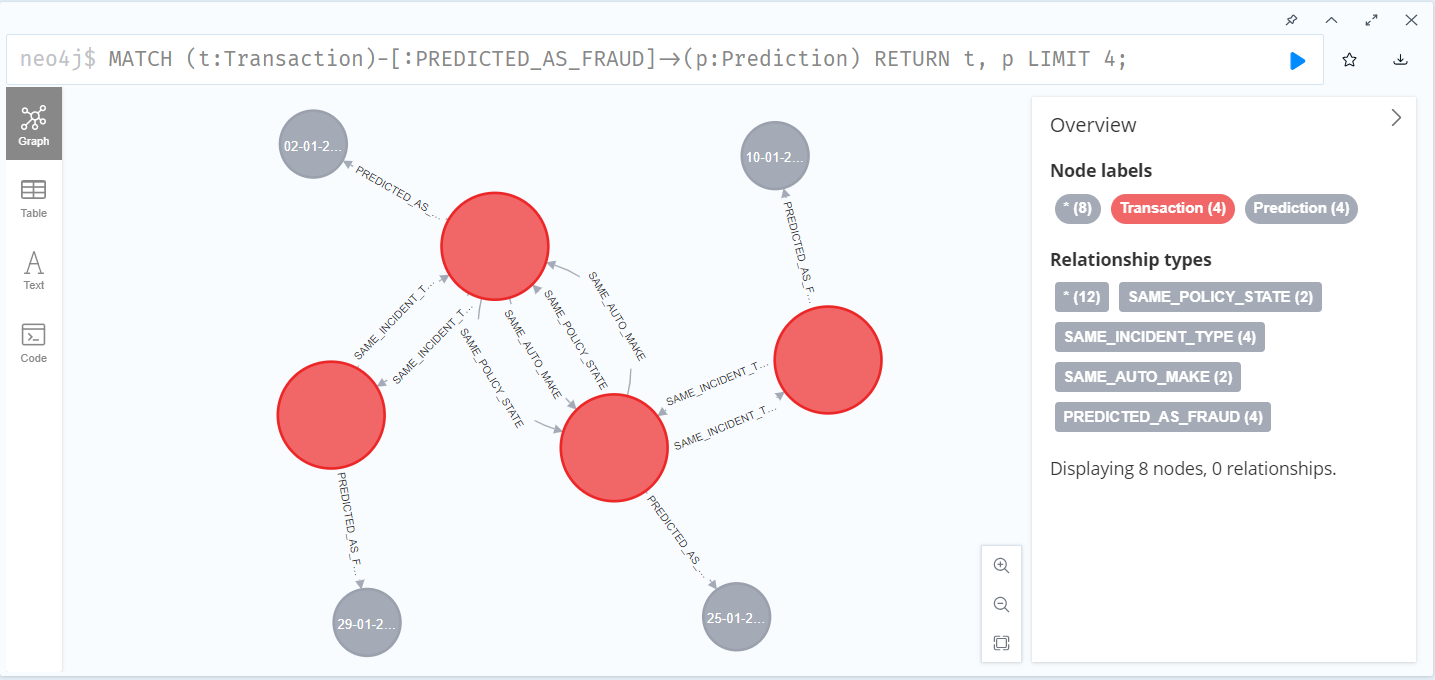
RETURN t, r LIMIT 4;



1. View all transactions that were predicted as fraud

MATCH (t:Transaction)-[:PREDICTED\_AS\_FRAUD]->(p:Prediction)

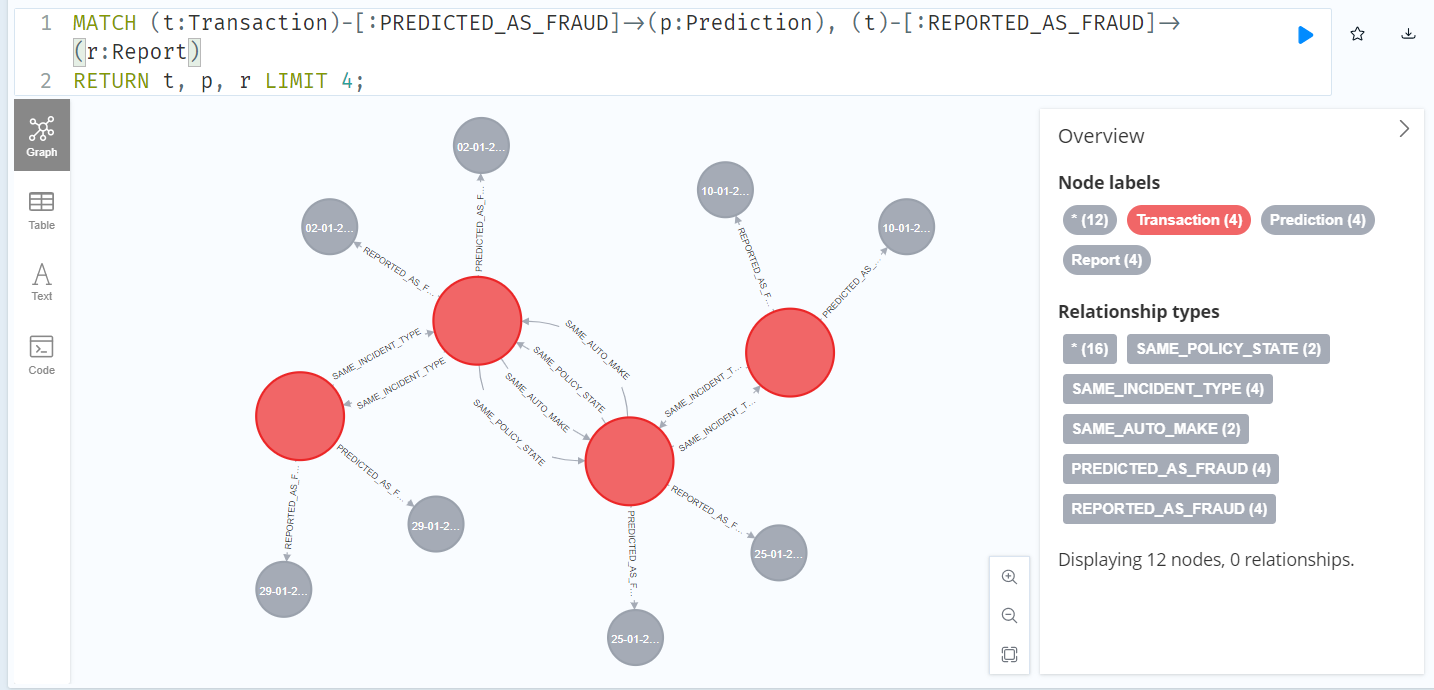
RETURN t, p LIMIT 4;



1. Compare transactions that are both predicted and reported as fraud (To see the details you can click on the nodes or its tie and see the values of it labels can be modified by clicking on it)

MATCH (t:Transaction)-[:PREDICTED\_AS\_FRAUD]->(p:Prediction), (t)-[:REPORTED\_AS\_FRAUD]->(r:Report)

RETURN t, p, r LIMIT 4;

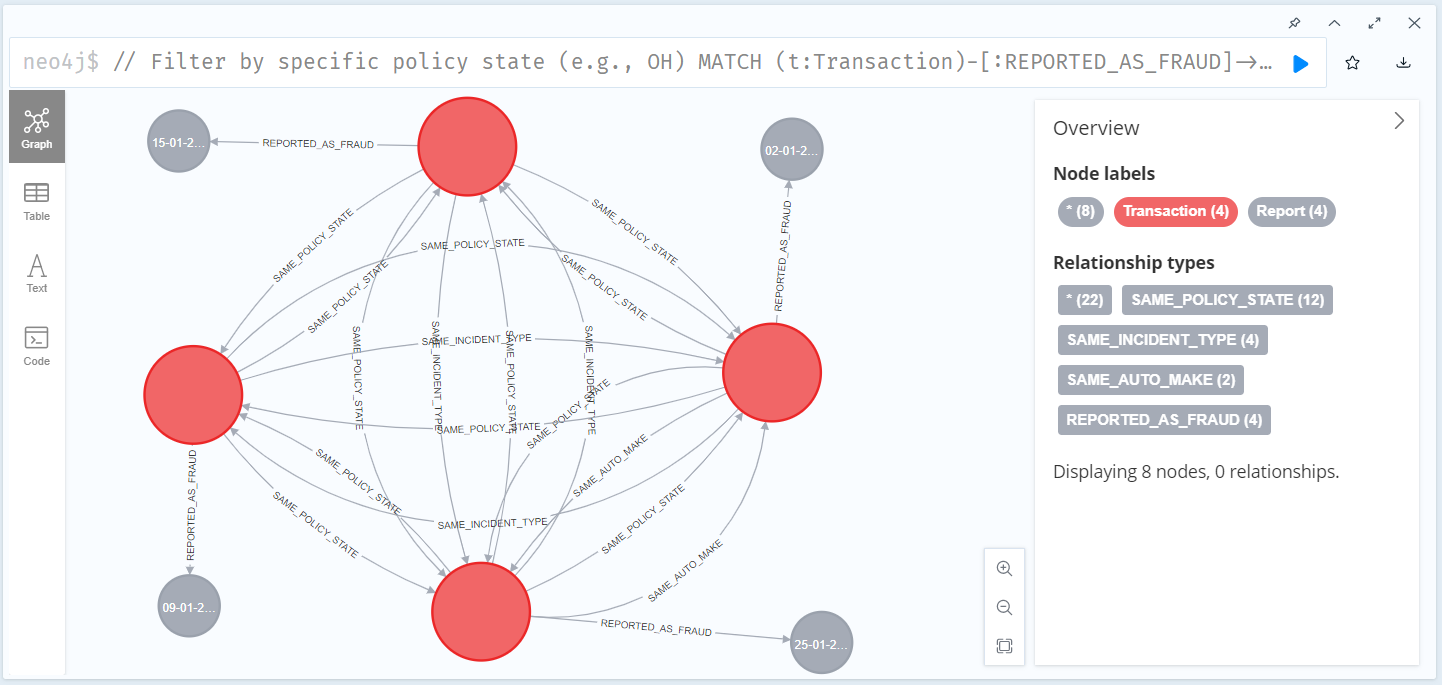


1. Filter by specific policy state (In this case “OH”)

MATCH (t:Transaction)-[:REPORTED\_AS\_FRAUD]->(r:Report)

WHERE t.policy\_state = 'OH'

RETURN t, r LIMIT 4;

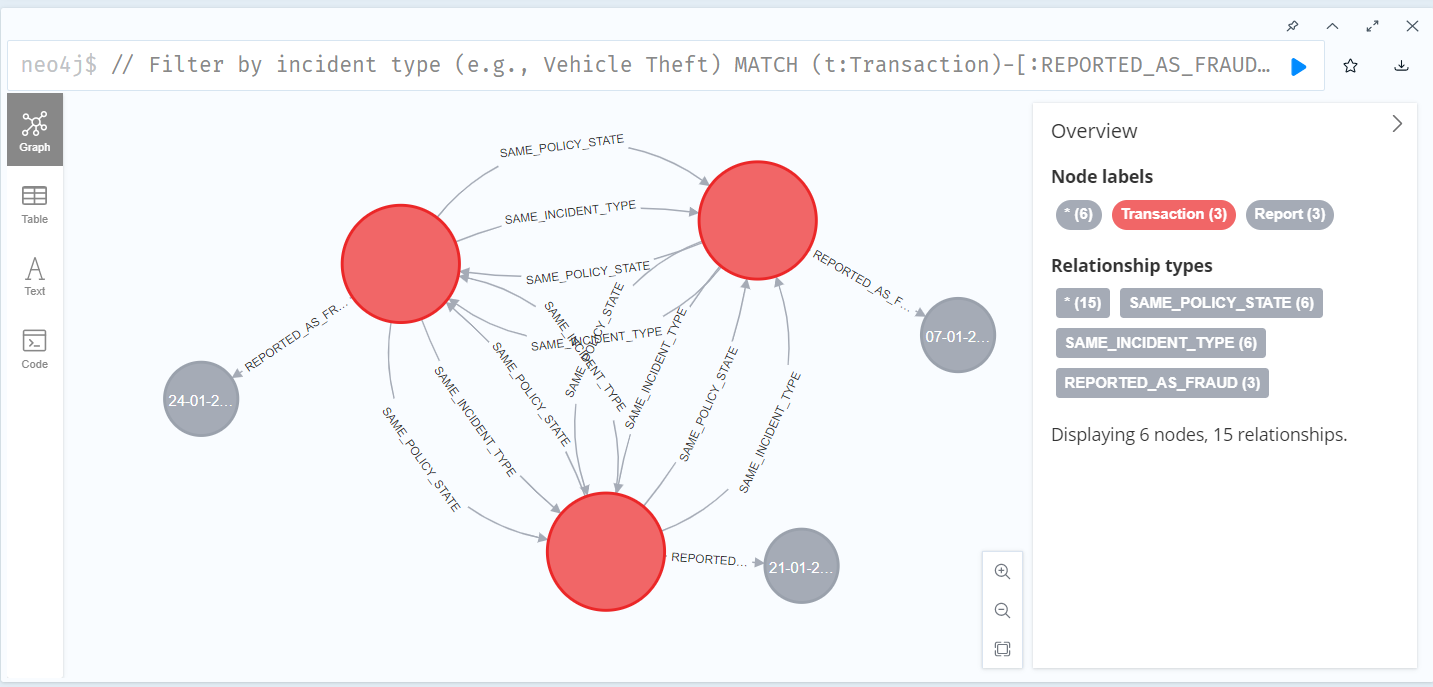


1. Filter by incident type (In this case “Vehicle Theft”)

MATCH (t:Transaction)-[:REPORTED\_AS\_FRAUD]->(r:Report)

WHERE t.incident\_type = 'Vehicle Theft'

RETURN t, r LIMIT 3;



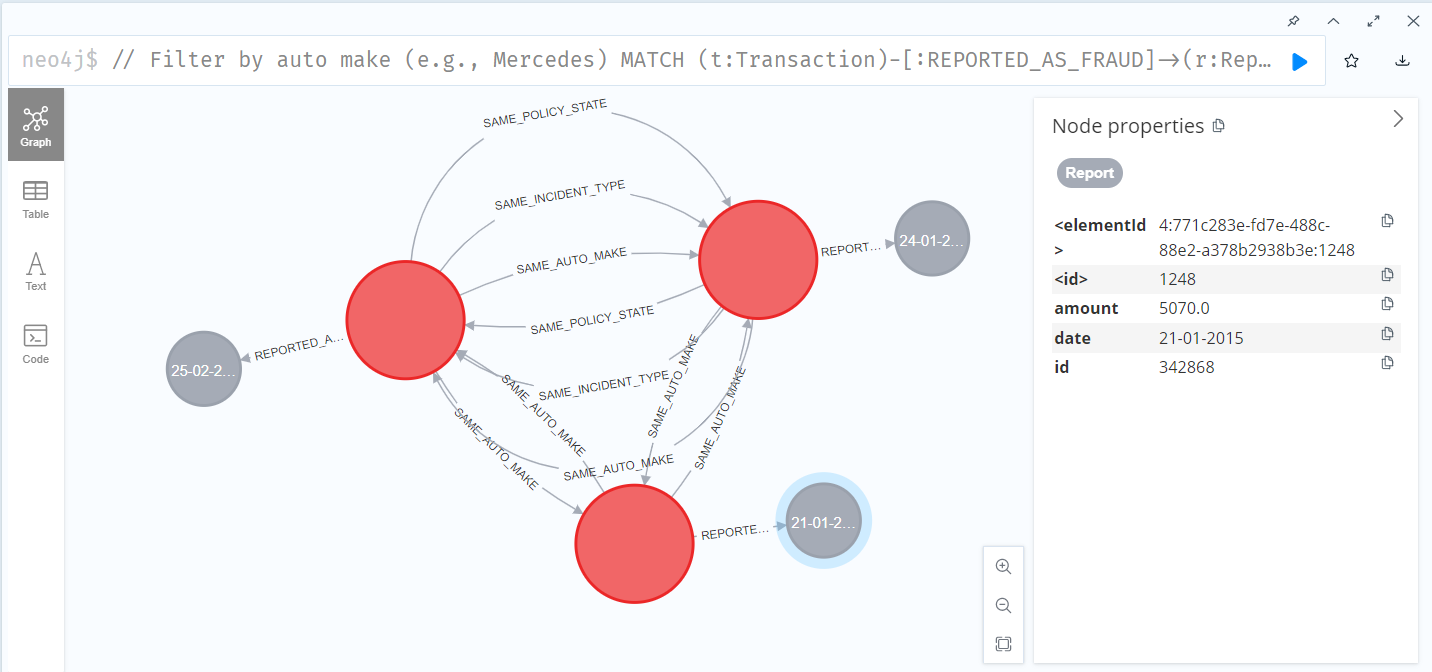
1. Filter by auto manufacturer (In this case “Mercedes”)

// Filter by auto make (e.g., Mercedes)

MATCH (t:Transaction)-[:REPORTED\_AS\_FRAUD]->(r:Report)

WHERE t.auto\_make = 'Mercedes'

RETURN t, r LIMIT 3;



1. You can see the details by clicking on nodes and also can modify according to your need by clicking on the label