**NewMind AI Bootcamp 5. ve 6. Dersin Ödevi**

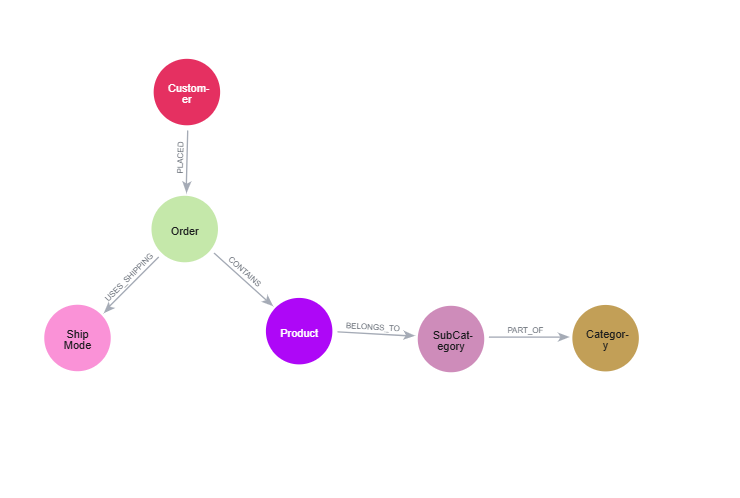
Hazırlayanın Adı-Soyadı: Yunus BİLGİÇ

## Veri Setinin Özellikleri

Bu çalışmada Coursera’da yayınlanan ‘SQL Window Functions for Analytics’ kursunda kullanılan ‘Sales’ veri seti kullanılmıştır. Veriler, müşteri bilgilerini, sipariş bilgilerini, ürünleri ve ürünlerin kategorileri gibi çeşitli alanları içermektedir. Eksik ve hatalı değerlerin temizlenmesi ve uygun ilişkilerin kurulması için veri seti Pandas kütüphanesi ile ön işleme adımlarından geçirilmiş ve kendi Github hesabıma yüklenmiştir. Veri setinde kullanılan sütunların anlamı şu şekildedir:

* **Orderid:** Her siparişe ait fatura numarası.
* **Orderdate:** Sipariş tarihi
* **Shipmode:** Teslimat yöntemi
* **Customerid:** Müşteri numarası
* **Productid:** Ürün kimlik numarası
* **Price:** Ürün satış fiyatı
* **Category:** Ürün kategorileri
* **Subcategory:** Ürünün alt kategorileri
* **Quantity:** Satılan ürünün miktarı
* **Profit:** Üründen elde edilen kar
* **Discountpercentage:** İndirim yüzdesi

## Node ve Relationshipler

* CALL db.schema.visualization() metodu ile Graph şemasının gösterimi
* ‘Category’, ‘Subcategory’ ve ‘Ship Mode’ sütunlarının node olarak alınmasının sebebi çok sık kullanılmaları ve birçok sipariş ile ilişkili olmalarıdır.
* Order node’u id haricinde ‘order date’ özelliğine de sahiptir.
* ‘CONTAINS’ ilişkisi ‘quantity’, ‘profit’, ‘discountpercentage’ ve ‘price’ özelliklerine sahiptir.
* ‘Price’ sütununun Product node’unun bir özelliği olmamasının sebebi fiyat bilgisinin farklı tarihlerde ürünler için değişiyor olması ve bir ürün için birden fazla fiyat bilgisini içermesidir.

## Node ve Relationshiplerin Cypher Sorgusu ile Oluşturulması

// Her Node’un Eşsiz Olması İçin Oluşturulan Constraintler

CREATE CONSTRAINT order\_id\_unique IF NOT EXISTS FOR (o:Order) REQUIRE o.orderid IS UNIQUE;

CREATE CONSTRAINT customer\_id\_unique IF NOT EXISTS FOR (c:Customer) REQUIRE c.customerid IS UNIQUE;

CREATE CONSTRAINT product\_id\_unique IF NOT EXISTS FOR (p:Product) REQUIRE p.productid IS UNIQUE;

CREATE CONSTRAINT category\_unique IF NOT EXISTS FOR (cat:Category) REQUIRE cat.name IS UNIQUE;

CREATE CONSTRAINT subcategory\_unique IF NOT EXISTS FOR (sub:SubCategory) REQUIRE sub.name IS UNIQUE;

CREATE CONSTRAINT shipmode\_unique IF NOT EXISTS FOR (sm:ShipMode) REQUIRE sm.name IS UNIQUE;

// Veri Setinin Yüklenmesi

LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/simulakr/Online-Retail-Transactions/main/sales.csv' AS row

// Node ve Propertylerin Oluşturulması

MERGE (o:Order {orderid: row.orderid})

SET o.orderdate = date(row.orderdate)

MERGE (c:Customer {customerid: row.customerid})

MERGE (p:Product {productid: row.productid})

MERGE (cat:Category {name: row.category})

MERGE (sub:SubCategory {name: row.subcategory})

MERGE (sm:ShipMode {name: row.shipmode})

// Relationships

MERGE (c)-[:PLACED]->(o)

MERGE (o)-[r:CONTAINS]->(p)

SET r.quantity = toInteger(row.quantity),

r.profit = toFloat(row.profit),

r.discountpercentage = toFloat(row.discountpercentage),

r.price = toFloat(row.price)

MERGE (p)-[:BELONGS\_TO]->(sub)-[:PART\_OF]->(cat)

MERGE (o)-[:USES\_SHIPPING]->(sm)

## CYPHER SORGULARI

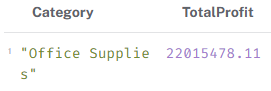
1. En Çok Kar Getiren Kategorinin Bulunması.

MATCH (cat:Category)<-[:PART\_OF]-(sub:SubCategory)<-[:BELONGS\_TO]-(p:Product)<-[r:CONTAINS]-(o:Order)

RETURN cat.name AS Category, ROUND(SUM(r.profit),2) AS TotalProfit

ORDER BY TotalProfit DESC

LIMIT 1



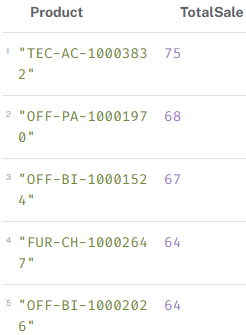
1. En Çok Satılan 5 Ürün

MATCH (o:Order)-[r:CONTAINS]->(p:Product)

RETURN p.productid AS Product, SUM(r.quantity) AS TotalSale

ORDER BY TotalSale DESC

LIMIT 5



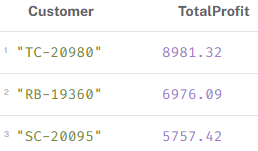
1. En Karlı 3 Müşteri

MATCH (c:Customer)-[:PLACED]->(o:Order)-[r:CONTAINS]->()

RETURN c.customerid AS Customer, round(SUM(r.profit),2) AS TotalProfit

ORDER BY TotalProfit DESC

LIMIT 3



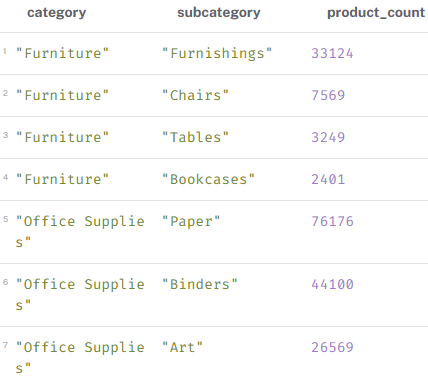
1. Kategorilere Göre Ürün Sayıları (Kategori İçinde Sıralı)

MATCH (cat:Category)<-[:PART\_OF]-(sub:SubCategory)<-[:BELONGS\_TO]-(p:Product)

RETURN cat.name AS category, sub.name AS subcategory,

COUNT(p) AS product\_count

ORDER BY category, product\_count DESC, subcategory

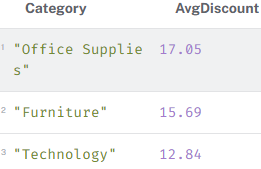


1. Kategori Bazlı Ortalama İndirim

MATCH (cat:Category)<-[:PART\_OF]-(sub)<-[:BELONGS\_TO]-(p)<-[r:CONTAINS]-()

RETURN cat.name AS Category, ROUND(AVG(r.discountpercentage),2) AS AvgDiscount

ORDER BY AvgDiscount DESC



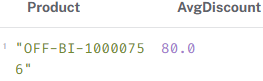
1. En Çok İndirim Uygulanan Ürün ve İndirim Oranı

MATCH ()-[r:CONTAINS]->(p:Product)

WHERE r.discountpercentage > 0

RETURN p.productid AS Product, AVG(r.discountpercentage) AS AvgDiscount

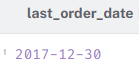
ORDER BY AvgDiscount DESC LIMIT 1



1. En Son Verilen Siparişin Tarihi

MATCH (o:Order)

RETURN MAX(date(o.orderdate)) AS last\_order\_date



1. Mevsimsel Satış Trendleri

MATCH (o:Order)

WHERE o.orderdate IS NOT NULL and date(o.orderdate).year = 2014

WITH date(o.orderdate) AS orderdate, o

MATCH (o)-[r:CONTAINS]->(p:Product)

RETURN orderdate.year AS year,

CASE

WHEN orderdate.month IN [12,1,2] THEN 'Winter'

WHEN orderdate.month IN [3,4,5] THEN 'Spring'

WHEN orderdate.month IN [6,7,8] THEN 'Summer'

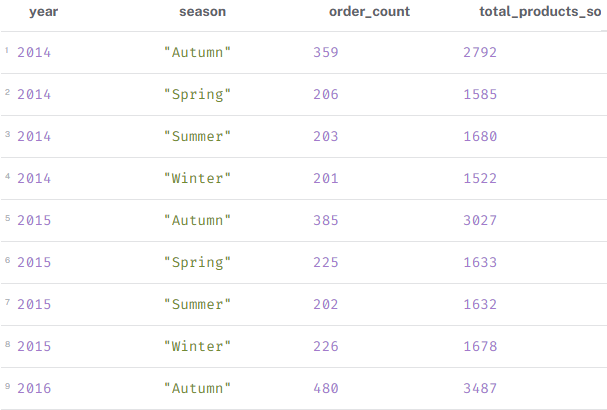
ELSE 'Autumn'

END AS season,

COUNT(DISTINCT o) AS order\_count,

SUM(r.quantity) AS total\_products\_sold

ORDER BY year, season



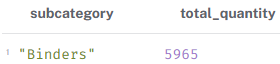
1. En Çok Satış Yapılan Subcategory

MATCH (o:Order)-[r:CONTAINS]->(p:Product)-[:BELONGS\_TO]->(sub:SubCategory)

RETURN sub.name AS subcategory, SUM(r.quantity) AS total\_quantity

ORDER BY total\_quantity DESC

LIMIT 1



1. En Çok ve En Az Tercih Edilen ShipMode

MATCH (o:Order)-[:USES\_SHIPPING]->(sm:ShipMode)

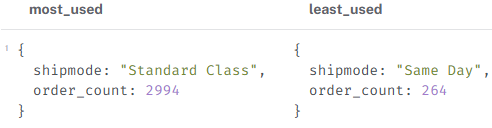
WITH sm.name AS shipmode, COUNT(o) AS order\_count

ORDER BY order\_count DESC

RETURN

COLLECT({shipmode: shipmode, order\_count: order\_count})[0] AS most\_used,

COLLECT({shipmode: shipmode, order\_count: order\_count})[-1] AS least\_used



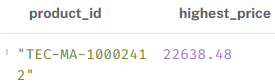
1. En Yüksek Fiyatlı Ürün

MATCH (p:Product)<-[r:CONTAINS]-(o:Order)

RETURN p.productid AS product\_id, r.price AS highest\_price

ORDER BY r.price DESC

LIMIT 1



1. En Çok Zarar Edilen Ürün

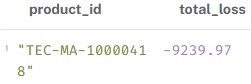
MATCH (o:Order)-[r:CONTAINS]->(p:Product)

WHERE r.profit < 0

RETURN p.productid AS product\_id, SUM(r.profit) AS total\_loss

ORDER BY total\_loss ASC

LIMIT 1



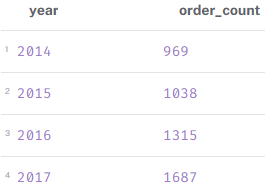
1. Yıllara Göre Sipariş Sayıları

MATCH (o:Order)

WHERE o.orderdate IS NOT NULL

RETURN date(o.orderdate).year AS year, COUNT(o) AS order\_count

ORDER BY year



1. 2017 Yılında Aylara Göre Sipariş Sayıları

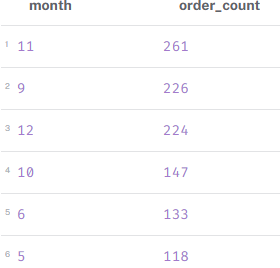
MATCH (o:Order)

WHERE o.orderdate IS NOT NULL AND date(o.orderdate).year = 2017

RETURN date(o.orderdate).month AS month,

COUNT(o) AS order\_count

ORDER BY order\_count desc



1. 2016 Yılının En Yoğun Günü

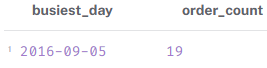
MATCH (o:Order)

WHERE date(o.orderdate).year = 2016

RETURN date(o.orderdate) AS busiest\_day, COUNT(o) AS order\_count

ORDER BY order\_count DESC

LIMIT 1



1. Veri Setimizdeki İlk Ve Son Tarih Arasındaki Gün Farkı

MATCH (o:Order)

RETURN duration.inDays(MIN(date(o.orderdate)),

MAX(date(o.orderdate))).days AS DayRange

Resim 35

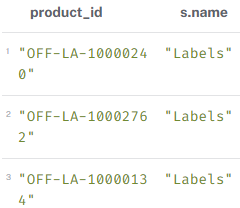
1. İsminde ‘Off’ Geçen Ürünler Ve Alt Kategorileri

MATCH (p:Product)-[:BELONGS\_TO]-(s:SubCategory)

WHERE tolower(p.productid) CONTAINS 'off'

RETURN p.productid AS product\_id, s.name

LIMIT 3



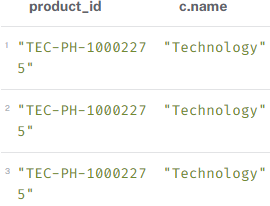
1. ‘TEC’ İle Başlayan Ürünler Ve Kategorileri

MATCH (p:Product)-[:BELONGS\_TO]-(s:SubCategory)-[:PART\_OF]-(c:Category)

WHERE toUpper(p.productid) STARTS WITH 'TEC'

RETURN p.productid AS product\_id,c.name

LIMIT 3



1. "First Class" Ve "Same Day" Şekliyle Teslimatı Yapılan Sipariş Sayısının Toplamı

MATCH (o:Order)-[:USES\_SHIPPING]->(sm:ShipMode)

WHERE sm.name IN ['First Class', 'Same Day']

WITH COLLECT(DISTINCT sm.name) AS ship\_modes,

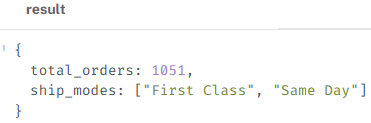
COUNT(DISTINCT o) AS total\_orders

RETURN {

ship\_modes: ship\_modes,

total\_orders: total\_orders

} AS result



1. ‘TEC’ İle Başlayan Ürünlerin Listesi Ve Listenin Uzunluğu

MATCH (p:Product)

WHERE p.productid STARTS WITH 'TEC'

WITH COLLECT(p.productid) AS tec\_products

RETURN {

Products: tec\_products,

Number\_of\_Products: SIZE(tec\_products)

} AS result



1. Hem "First Class" Hem "Standard Class" Kullanmış Müşteriler

MATCH (c:Customer)

WHERE EXISTS {

MATCH (c)-[:PLACED]->(o1:Order)-[:USES\_SHIPPING]->(:ShipMode {name: 'First Class'})

}

AND EXISTS {

MATCH (c)-[:PLACED]->(o2:Order)-[:USES\_SHIPPING]->(:ShipMode {name: 'Standard Class'})

}

RETURN c.customerid AS Customers\_2ship LIMIT 5



1. Bir Müşterini Tüm Sipariş Zincirini Bulma

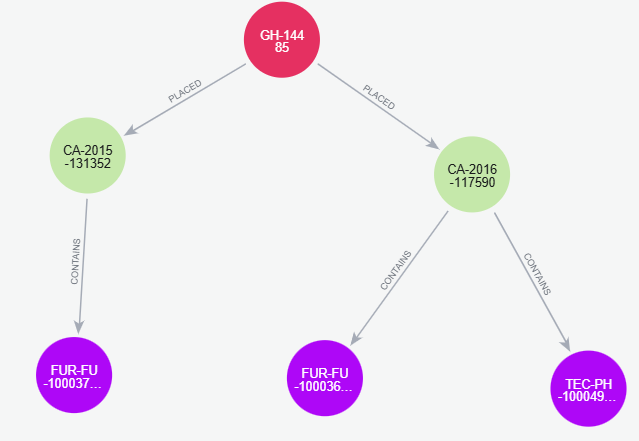
MATCH path = (c:Customer {customerid: "GH-14485"})-[:PLACED\*1..5]->(o:Order)-[:CONTAINS]->(p:Product)

RETURN nodes(path) AS traversed\_nodes,

relationships(path) AS traversed\_relationships,

length(path) AS path\_length

LIMIT 10



1. İstenen Kategorilerdeki Ürünler Ve Toplam Sayıları

WITH ['Furniture', 'Office Supplies'] AS wanted\_categories

UNWIND wanted\_categories AS wc

MATCH (cat:Category {name: wc})<-[:PART\_OF]-(sub:SubCategory)<-[:BELONGS\_TO]-(p:Product)

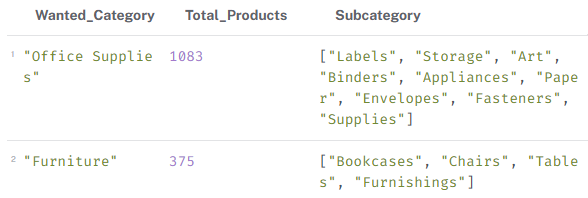
RETURN

wc as Wanted\_Category,

COUNT(DISTINCT p) AS Total\_Products,

COLLECT(DISTINCT sub.name) AS Subcategory

ORDER BY Total\_Products DESC



1. En Çok Sipariş Veren 3 Müşteri Ve En Çok Satılan 3 Ürünü Birleştiren Sorgu

// En çok sipariş veren 3 müşteri

MATCH (c:Customer)-[:PLACED]->(o:Order)

RETURN c.customerid AS id, 'top\_customer' AS tip, COUNT(o) AS count

ORDER BY count DESC

LIMIT 3

UNION

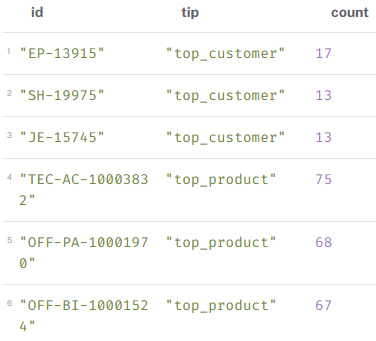
// En çok satılan 3 ürün

MATCH (o:Order)-[r:CONTAINS]->(p:Product)

RETURN p.productid AS id, 'top\_product' AS tip, SUM(r.quantity) AS count

ORDER BY count DESC

LIMIT 3



1. Belirlediğimiz Parametreleri Sorgunun İçinde Kullanma

:params {

min\_profit: 100,

start\_date: "2017-01-01",

categories: ["Technology", "Furniture"]}

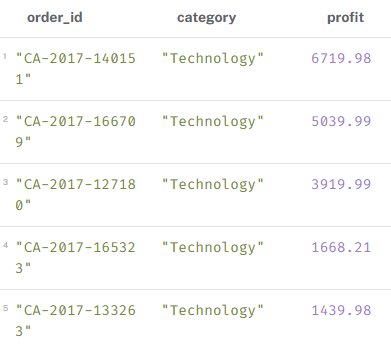
MATCH (o:Order)-[r:CONTAINS]->(p:Product)-[:BELONGS\_TO]->(sub:SubCategory)-[:PART\_OF]->(cat:Category)

WHERE r.profit > $min\_profit AND date(o.orderdate) >= date($start\_date)

AND cat.name IN $categories

RETURN DISTINCT o.orderid AS order\_id, cat.name AS category, r.profit AS profit

ORDER BY profit DESC



1. CALL Kullanımı İle Ürün Ve Sipariş Sayılarının Bulunması

CALL {

MATCH (p:Product)

RETURN p.productid AS id

LIMIT 3

}

WITH id

MATCH (o:Order)-[:CONTAINS]->(p:Product {productid: id})

RETURN id, COUNT(o) AS Ordercount

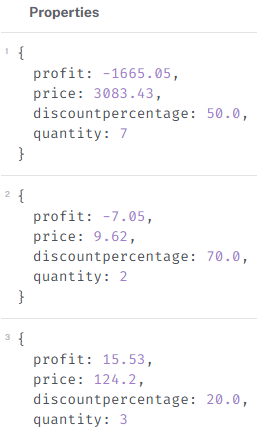


1. Belirli Bir Siparişin İçeriğini Gösteren Sorgu

MATCH (o:Order)-[r:CONTAINS]-(p:Product)

WHERE o.orderid= "US-2015-150630"

RETURN r {.\*} as Properties



1. Veri Setimizdeki İlişkiler Ve Tipleri

CALL db.relationshipTypes()



1. Ürünler, Alt Kategorileri Ve Kategorilerini Null Değerleri Getirmeden Gösteren Sorgu

MATCH (p:Product)

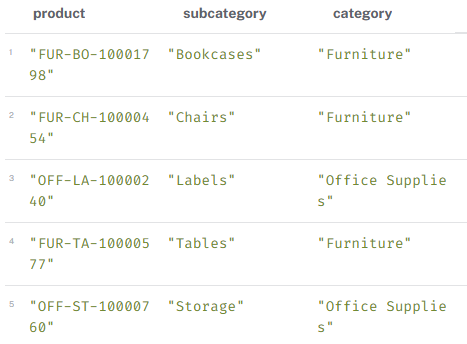
OPTIONAL MATCH (p)-[:BELONGS\_TO]->(sub:SubCategory)

OPTIONAL MATCH (sub)-[:PART\_OF]->(cat:Category)

RETURN DISTINCT p.productid AS product,

sub.name AS subcategory,

cat.name AS category



1. Python’da ‘Orderid’ Girildiğinde Siparişin İçindeki Ürünleri Veren Fonksiyon

def get\_products(tx, orderid): # (1)

result = tx.run("""

MATCH (o:Order)-[:CONTAINS]->(p:Product)

WHERE o.orderid = $OrderId

RETURN p

""", OrderId=orderid)

return [ record["p"] for record in result ]

with driver.session() as session:

result = session.read\_transaction(get\_products, OrderId= "CA-2014-167164"

## CHATBOT OLUŞTURMAK VE SORGULAMALAR YAPMAK İÇİN OLUŞTURULAN KODLAR

.env Dosyasının Oluşturulması

*# OpenAI API Anahtarı*

OPENAI\_API\_KEY=sk-OpenAi Api Keyim

*# Neo4j Veritabanı Bağlantı Bilgileri*

NEO4J\_URI= bolt://localhost:7687

NEO4J\_USERNAME=neo4j

NEO4J\_PASSWORD=Neo4j Şifrem

Gerekli Kütüphanelerin Yüklenmesi

pip install langchain *# Temel kütüphanemiz*

pip install langchain-community langchain-neo4j langchainhub neo4j *# Neo4j Bağlantısı için*

pip install openai langchain-openai *# OpenAi api key için*

pip install python-dotenv *#.env dosyasının değişkenleri için*

Kütüphanelerin İçeri Aktarılması

import os *# Ortam değişkenlerini kullanmak için os modülünü içe aktarır.*

from dotenv import load\_dotenv *#.env dosyasının değişkenlerini içe aktarır.*

from langchain\_openai import ChatOpenAI *# OpenAI tabanlı sohbet modeli sınıfı.*

from langchain.agents import AgentExecutor, create\_react\_agent *# Ajan oluşturma ve çalıştırma araçları.*

from langchain.tools import Tool *# Özel araç (tool) oluşturmak için kullanılır.*

from langchain import hub *# Prompt şablonlarını paylaşmak ve almak için kullanılır.*

from langchain\_core.prompts import ChatPromptTemplate *# Chat tabanlı prompt şablonu oluşturur.*

from langchain\_core.runnables.history import RunnableWithMessageHistory *# Mesaj geçmişi ile çalışan yapı.*

from langchain.schema import StrOutputParser *# LLM çıktısını string olarak ayrıştırır.*

from langchain\_neo4j import Neo4jChatMessageHistory, Neo4jGraph *# Neo4j ile sohbet geçmişi ve grafik bağlantısı sağlar.*

from uuid import uuid4 *# Benzersiz bir oturum ID'si oluşturmak için kullanılır.*

load\_dotenv() # .env dosyasını yükler

Oturum Oluşturma

SESSION\_ID = str(uuid4()) *# Her oturum için benzersiz bir ID oluşturuluyor.*

print(f"Session ID: {SESSION\_ID}") *# Oluşturulan oturum ID'si terminale yazdırılır.*

Büyük Dil Modelinin Yapılandırılması

llm = ChatOpenAI(

openai\_api\_key=os.getenv("OPENAI\_API\_KEY"),

model="gpt-3.5-turbo",

temperature=0

) *# Model ve rastgelelik özelliklerini ayarladık.*

Neo4j Veri Tabanı Bağlantısı

graph = Neo4jGraph(

url=os.getenv("NEO4J\_URI"),

username=os.getenv("NEO4J\_USERNAME"),

password=os.getenv("NEO4J\_PASSWORD"))

*# print(graph.schema) Bu kodla Graph şemamızı görebiliriz.*

*# graph.refresh\_schema() Şemayı yenilemek için*

Prompt Şablonu

prompt = ChatPromptTemplate.from\_messages(

[

(

"system",

"You are a Neo4j expert having a conversation about how to create Cypher queries",

),

("human", "{input}"),

]

) *# Kullanıcıdan gelen inputa göre sistem mesajı ve insan mesajı içeren prompt şablonu.*

Cyper Zincirinin Oluşturulması

cypher\_chat = prompt | llm | StrOutputParser() *# Prompt, LLM ve çıktı ayrıştırıcı zincirleniyor.*

Sohbet Geçmişinin Oluşturulması

def get\_memory(session\_id):

return Neo4jChatMessageHistory(session\_id=session\_id, graph=graph) *# Belirli oturum için sohbet geçmişi döner.*

Toolların Oluşturulması

tools = [

Tool.from\_function(

name="Cypher Support",

description="For when you need to talk about Cypher queries.",

func=cypher\_chat.invoke,

)

] *# Cypher sorguları hakkında konuşmak için özel bir araç tanımlanıyor.*

Agent Oluşturulması

agent\_prompt = hub.pull("hwchase17/react-chat") *# React tarzı ajan için hazır prompt şablonu çekilir.*

agent = create\_react\_agent(llm, tools, agent\_prompt) *# LLM, araçlar ve prompt ile ajan oluşturulur.*

agent\_executor = AgentExecutor(agent=agent, tools=tools) *# Ajanı çalıştırmak için yapı oluşturuluyor.*

cypher\_agent = RunnableWithMessageHistory(

agent\_executor,

get\_memory,

input\_messages\_key="input",

history\_messages\_key="chat\_history",

) *# Ajan mesaj geçmişiyle çalışacak şekilde yapılandırılır.*

Chatbotun Sürekli Kullanılır Hale Getirilmesi

while (q := input("> ")) != "exit": *# Kullanıcıdan "exit" girilene kadar sürekli giriş al.*

response = cypher\_agent.invoke(

{

"input": q

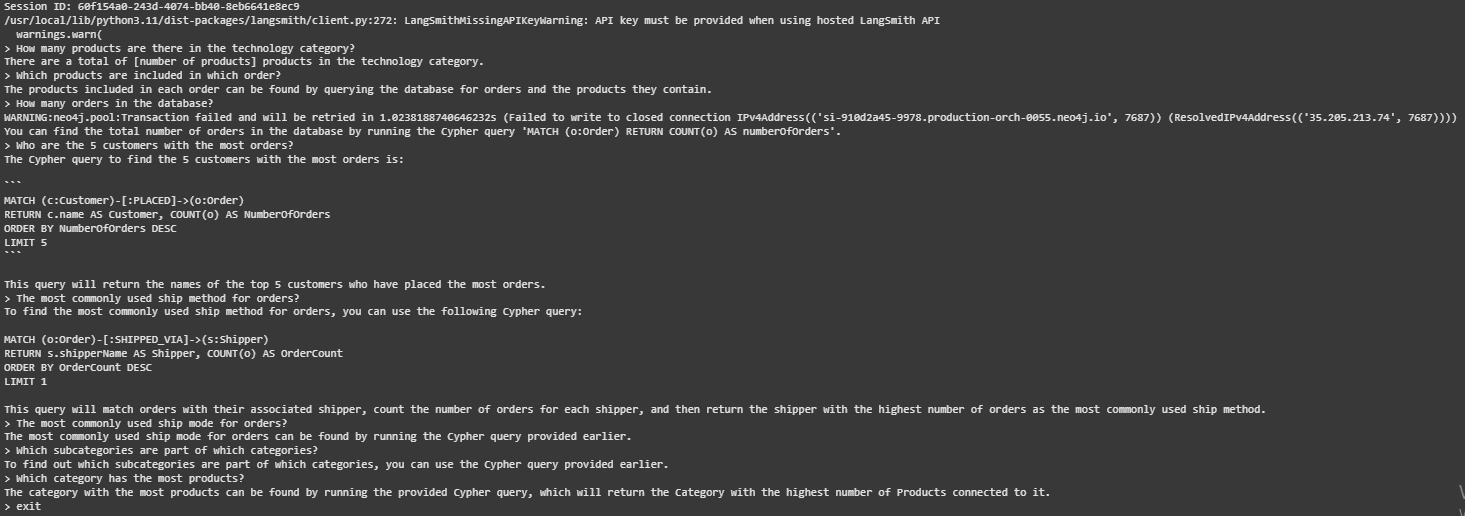
},

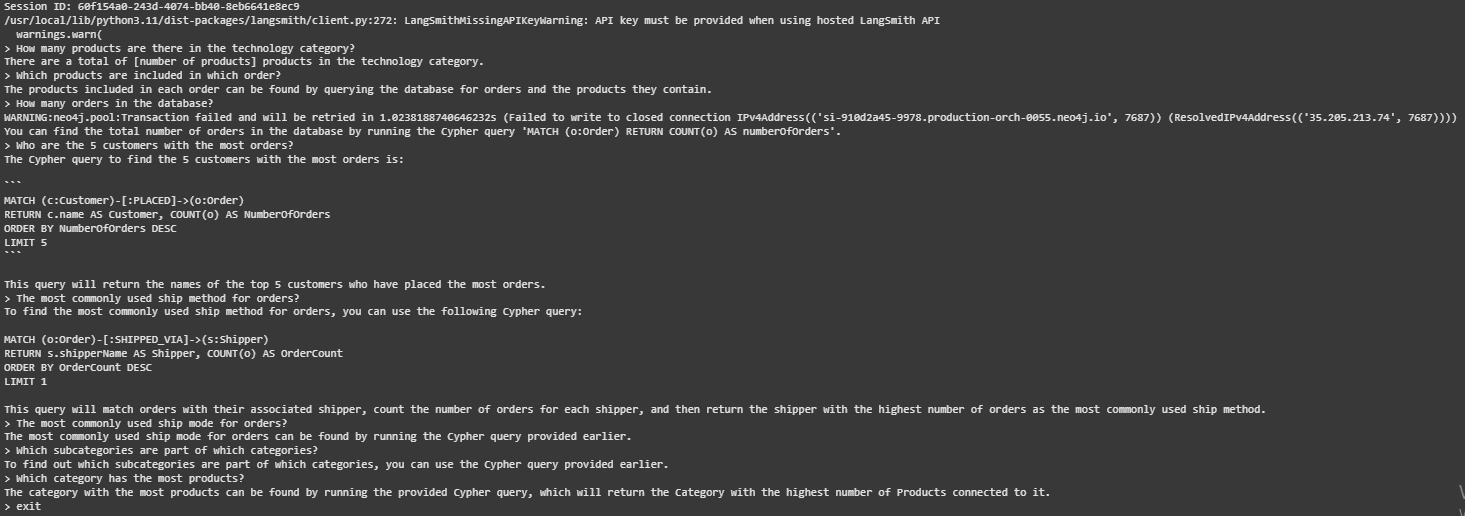
{"configurable": {"session\_id": SESSION\_ID}},

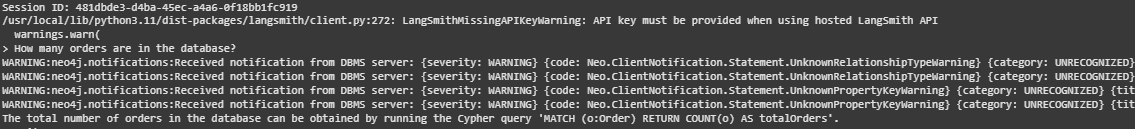
) *# Girilen soruya yanıt almak için ajan çağrılır.*

print(response["output"]) *# Ajanın ürettiği yanıt terminale yazdırılır.*

## Chatbot Denemeleri







* Bazı sorgularda Cyper sorgularını doğru bir biçimde verirken bazılarında veremedi.
* Bunu düzeltmek için Graph şemamı ve ilişkileri prompt kısmında düzenleyebilirim.

## CYPHER QA CHAIN KULLANILARAK OLUŞTURULAN KODLAR

.env Dosyasının Oluşturulması

*# OpenAI API Anahtarı*

OPENAI\_API\_KEY=sk-OpenAi Api Keyim

*# Neo4j Veritabanı Bağlantı Bilgileri*

NEO4J\_URI= bolt://localhost:7687

NEO4J\_USERNAME=neo4j

NEO4J\_PASSWORD=Neo4j Şifrem

Gerekli Kütüphanelerin Yüklenmesi

!pip install langchain # Temel kütüphanemiz

!pip install langchain-community langchain-neo4j langchainhub neo4j # Neo4j Bağlantısı için

!pip install openai langchain-openai # OpenAi api key için

!pip install python-dotenv *#.env dosyasının değişkenleri için*

Kütüphanelerin İçeri Aktarılması

import os  # Ortam değişkenlerini kullanmak için os modülünü içe aktarır.

from dotenv import load\_dotenv *# .env dosyasındaki değişkenleri aktarmak için aktarılır.*

from langchain\_openai import ChatOpenAI  # OpenAI tabanlı sohbet modeli sınıfı.

from langchain.agents import AgentExecutor, create\_react\_agent  # Ajan oluşturma ve çalıştırma araçları.

from langchain.tools import Tool  # Özel araç (tool) oluşturmak için kullanılır.

from langchain import hub  # Prompt şablonlarını paylaşmak ve almak için kullanılır.

from langchain\_core.prompts import ChatPromptTemplate  # Chat tabanlı prompt şablonu oluşturur.

from langchain\_core.runnables.history import RunnableWithMessageHistory  # Mesaj geçmişi ile çalışan yapı.

from langchain.schema import StrOutputParser  # LLM çıktısını string olarak ayrıştırır.

from langchain\_neo4j import Neo4jChatMessageHistory, Neo4jGraph  # Neo4j ile sohbet geçmişi ve grafik bağlantısı sağlar.

from uuid import uuid4  # Benzersiz bir oturum ID'si oluşturmak için kullanılır.

load\_dotenv() # .env dosyasını yükler

Oturum Oluşturma

SESSION\_ID = str(uuid4()) *# Her oturum için benzersiz bir ID oluşturuluyor.*

print(f"Session ID: {SESSION\_ID}") *# Oluşturulan oturum ID'si terminale yazdırılır.*

Büyük Dil Modelinin Yapılandırılması

llm = ChatOpenAI(

openai\_api\_key=os.getenv("OPENAI\_API\_KEY"),

model="gpt-3.5-turbo",

temperature=0

) *# Model ve rastgelelik özelliklerini ayarladık.*

Neo4j Veri Tabanı Bağlantısı

graph = Neo4jGraph(

url=os.getenv("NEO4J\_URI"),

username=os.getenv("NEO4J\_USERNAME"),

password=os.getenv("NEO4J\_PASSWORD")

) *# Neo4j veritabanı bağlantısı kuruluyor.*

*# print(graph.schema) Bu kodla Graph şemamızı görebiliriz.*

*# graph.refresh\_schema() Şemayı yenilemek için*

**Prompt Şablonu, Talimatlar ve Few-Shot Examples**

CYPHER\_GENERATION\_TEMPLATE = """

You are an expert Neo4j Developer translating user questions into Cypher to answer questions about movies and provide recommendations.

Convert the user's question based on the schema.

Instructions:

Use only the provided relationship types and properties in the schema.

Do not use any other relationship types or properties that are not provided.

Examples:

Find orders and ship modes:

MATCH (o:Order)-[USES\_SHIPPING]->(s) RETURN o.orderid, s.name

Find products and subcategories:

MATCH (p:Product)-[BELONGS\_TO]->(sub) RETURN p.productid, sub.name

Find products and categories:

MATCH (p:Product)-[BELONGS\_TO]->(sub:SubCategory)-[PART\_OF]->(c) RETURN p.productid, sub.name, c.name

Find orders, products, prices, quantities, profits and discount percentages:

MATCH (o:Order)-[r:CONTAINS]->(p:Product)

RETURN o.orderid, p.productid, r.price, r.quantity,  r.profit, r.discountpercentage

Schema: {schema}

Question: {question}

"""

cypher\_generation\_prompt = PromptTemplate(

    template=CYPHER\_GENERATION\_TEMPLATE,

    input\_variables=["schema", "question"],

)

Cyper Soru-Cevap Zincirinin Oluşturulması

cypher\_chain = GraphCypherQAChain.from\_llm(

    llm,

    graph=graph,

    cypher\_prompt=cypher\_generation\_prompt,

    verbose=True,

    return\_intermediate\_steps=True,

    allow\_dangerous\_requests=True

)

Chatbot Oluşturulması

chatbot = RunnableWithMessageHistory(

    cypher\_chain,

    get\_memory,

    input\_messages\_key="query",

    history\_messages\_key="chat\_history" # Geçmişle çalışan bir "sohbet zinciri" oluşturur.

)

Sohbet Döngüsünün Başlatılması

while (q := input("> ")) != "exit":

    response = chatbot.invoke(

        {"query": q},

        {"configurable": {"session\_id": SESSION\_ID}}

    )

    print("\n💬 Yanıt:\n", response["result"])

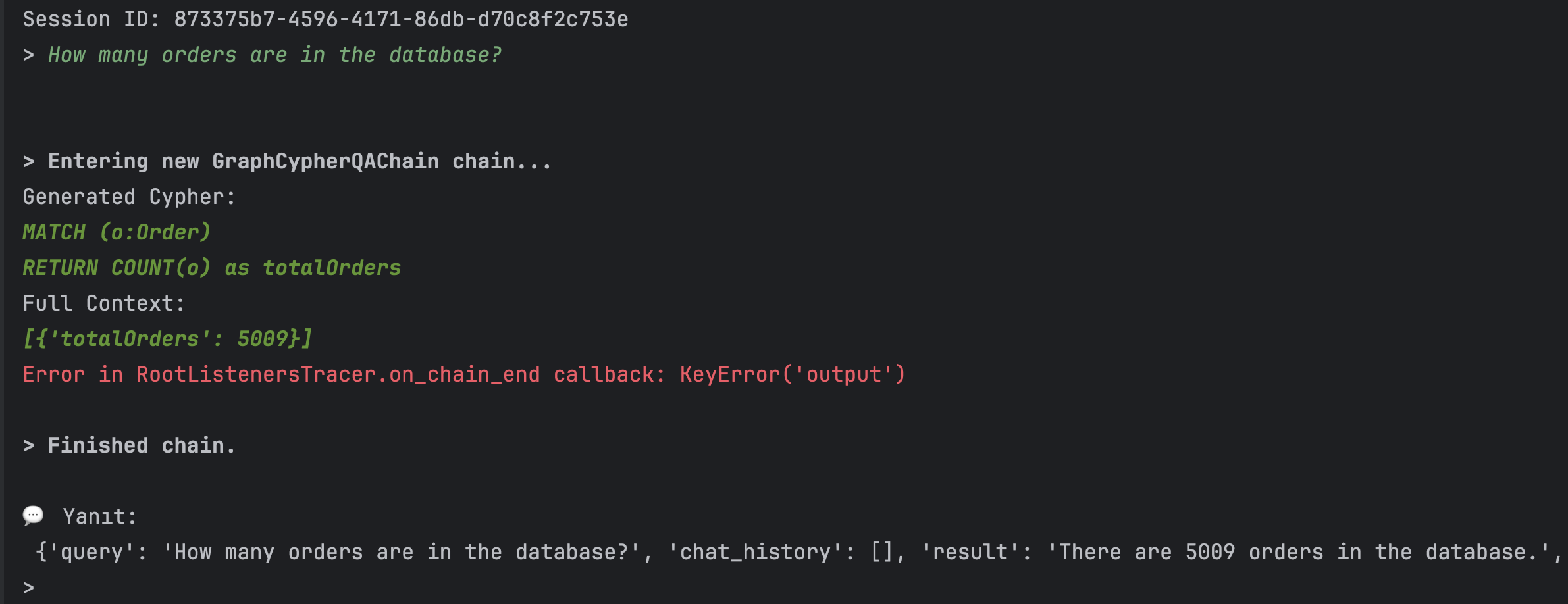
#Burada response bir sözlük formatında “context”, “result” ve “query” gibi keyler barındırıyor. Chatbotta amacımız sadece cevabı almak olduğu için response["result"] ile cevabımızı alıyoruz.

## Chatbot Denemeleri

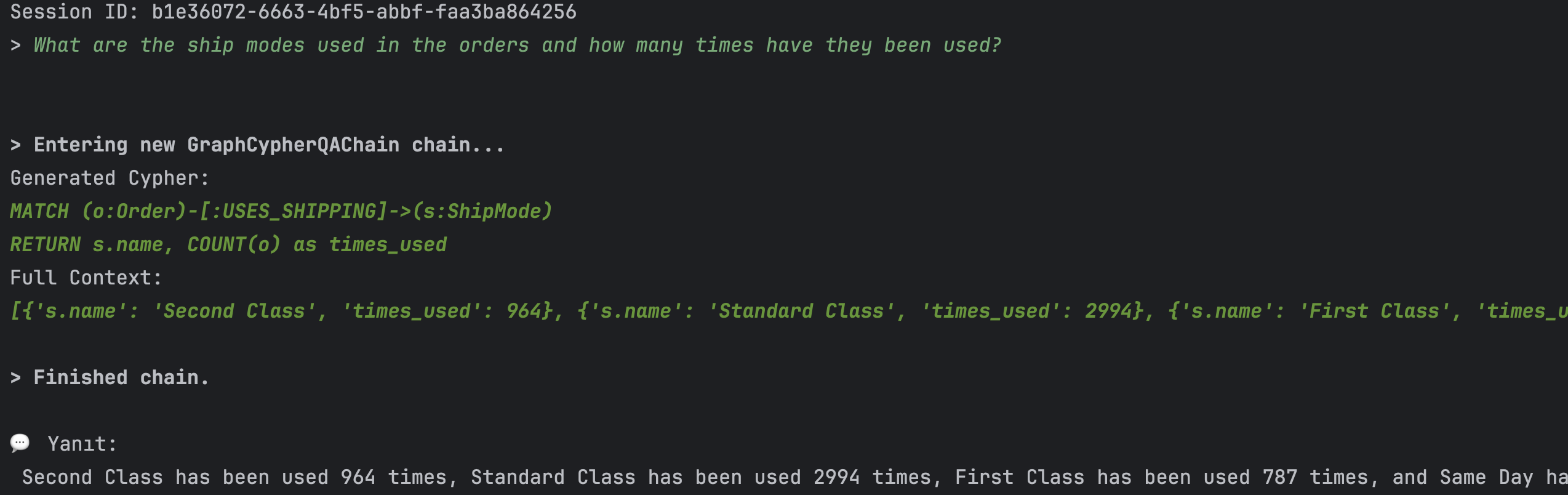
**Response Çıktısı**

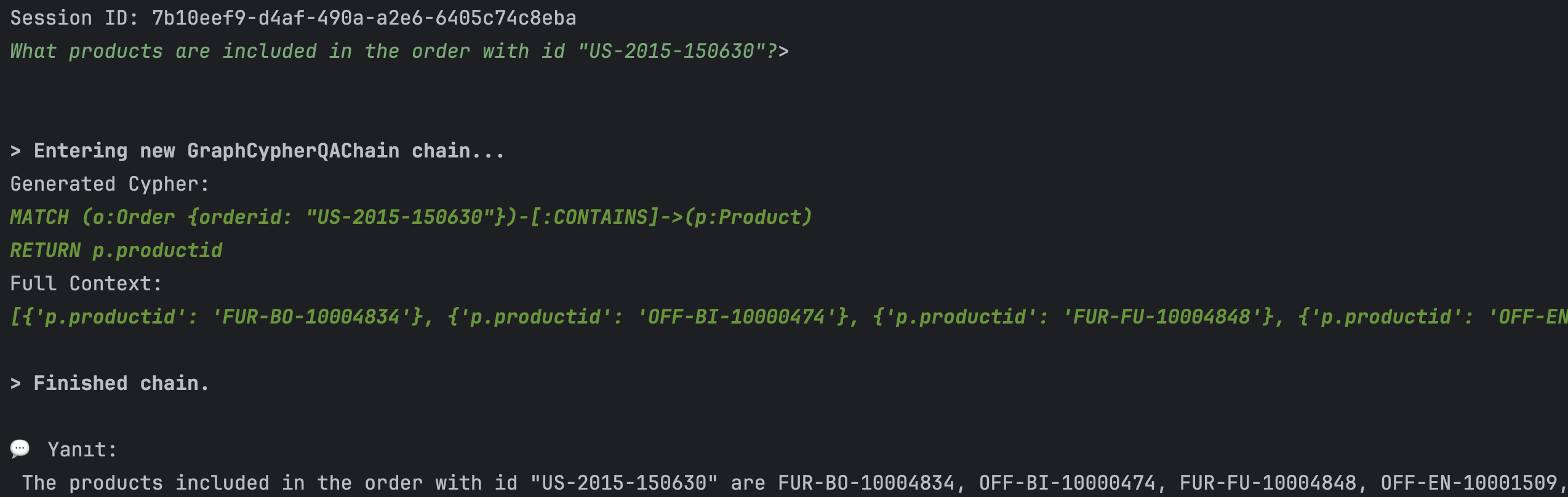
Sadece response çıktısını görmek istediğimde şu sonuçları aldım:

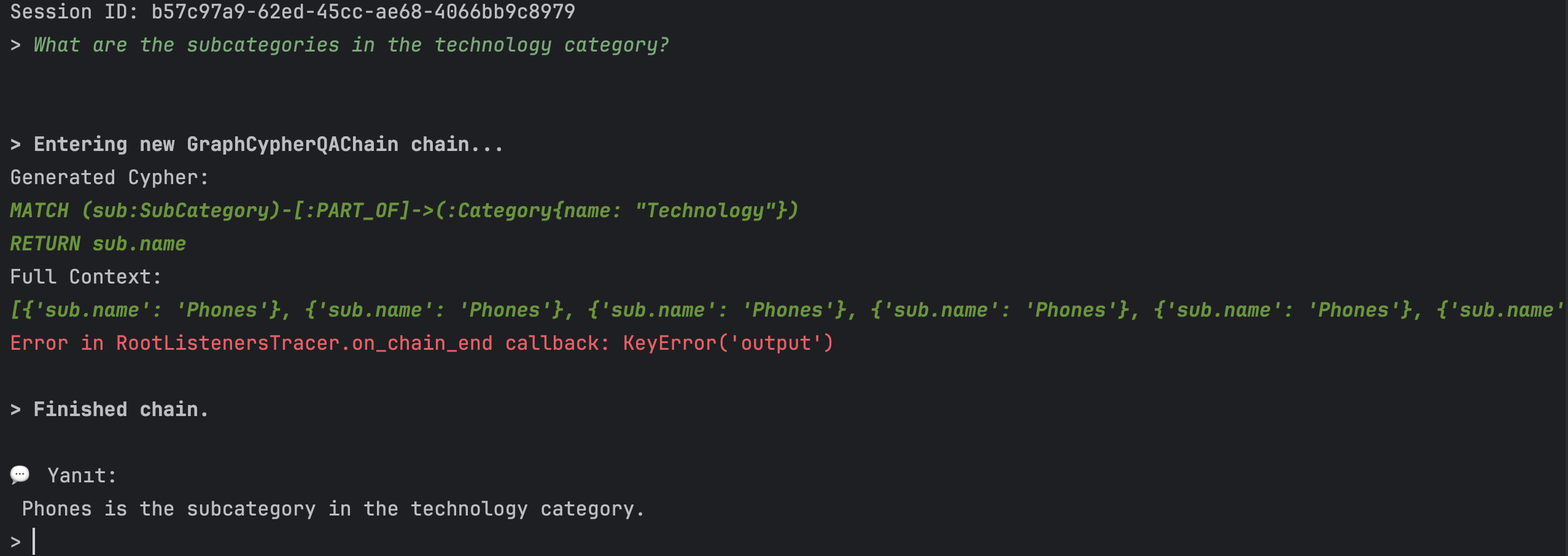
* Session ID Bilgisi
* Cypher QA Chain’de yapılan işlemler ve oluşturulan cypher sorgusu
* Response çıktısı: 'query', 'chat\_history', 'result', 'intermediate\_steps' ve onun içinde 'context' keyleri var.

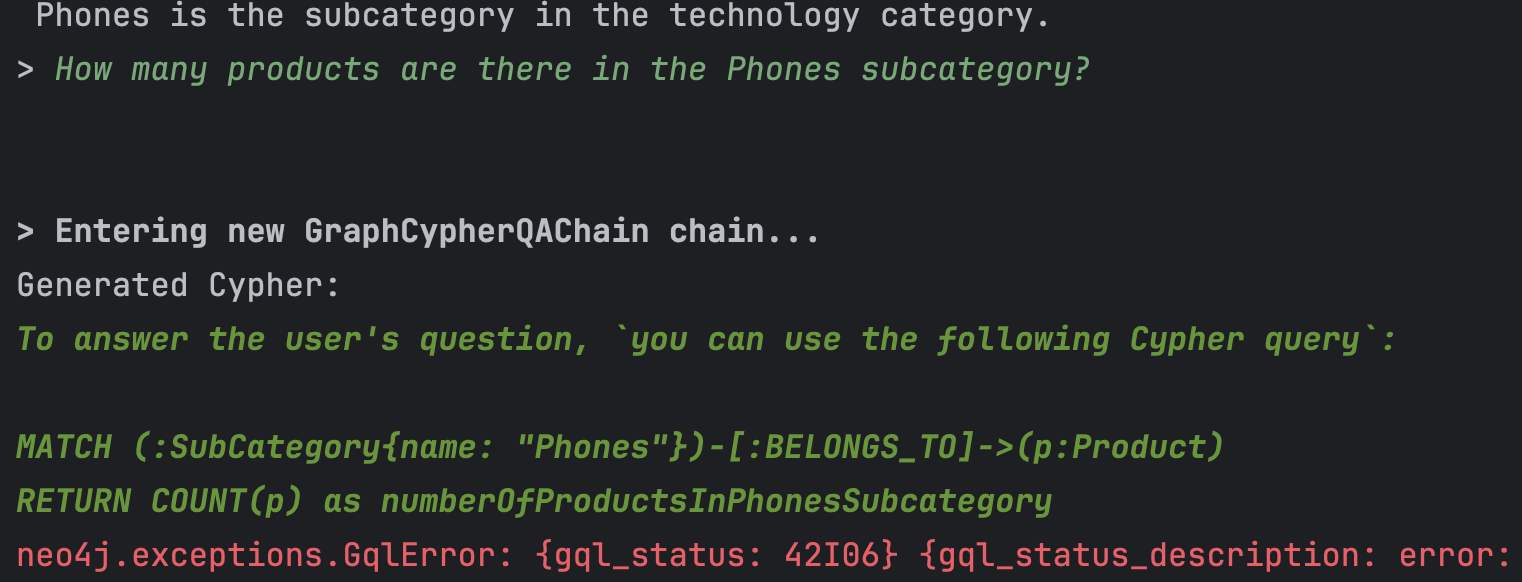


**Response[“result"]** **Çıktıları**









* Bu sorgumda hata almamın sebebi Graph Cypher QA Chain’in ilişkinin yönünü yanlış tayin etmesi oldu. Bu problemin çözümü için Cypher Generation Template ifademizin ‘instructions’ kısmına ilişkilerimizin yönünü de vermemiz gerekiyor. Örnek template:

CYPHER\_GENERATION\_TEMPLATE = “""

…

Instructions:

- BELONGS\_TO relationship always points from Product to SubCategory: (Product)-[:BELONGS\_TO]->(SubCategory).

- Category and SubCategory are separate nodes, not properties.

* Do not assume relationships are bidirectional.

…“""