Neo4j import and Queries

Abhimanyu Aryan
abhimanyu.chat@gmail.com

1. CSV exports

From SQL Oracle developer you can export data as CSVs which is what I'll use to import data in Neo4j

Department doesn't have any foreign key and dependency so I first inserted that in neo4j

```
// Create Department nodes
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-projeto/
neo4j/hospital-csv/department.csv/DEPARTMENT_DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWH
QLIVZMMEELIP5K6KZROTJXQ' AS row
CREATE (:Department {
    IDDEPARTMENT: row.IDDEPARTMENT,
    DEPT_HEAD: row.DEPT_HEAD,
    DEPT_NAME: row.DEPT_NAME,
    EMP_COUNT: toInteger(row.EMP_COUNT)
});
```

Staff Staff has relation with department with IDDepartment. So, we need. So, we will build a relation between both

```
// Load staff CSV data
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-
projeto/neo4j/hospital-csv/staff/STAFF_DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWHVG7N5L
5VNW4TAKXUZROTLDQ' AS row

// Create staff nodes
CREATE (:Staff {
    EMP_ID: toInteger(row.EMP_ID),
    EMP_FNAME: row.EMP_FNAME,
    EMP_LNAME: row.EMP_LNAME,
    DATE_JOINING: row.DATE_JOINING,
    DATE_SEPERATION: row.DATE_SEPERATION,
    EMAIL: row.EMAIL,
    ADDRESS: row.ADDRESS,
    SSN: toInteger(row.SSN),
    IS_ACTIVE_STATUS: row.IS_ACTIVE_STATUS,
    IDDEPARTMENT: row.IDDEPARTMENT
});
```

OK for some reason both department and staff nodes were missing `iddepartment` and I couldn't establish a relation between them. Something went wrong during insertion.

So, I'll drop both

```
MATCH (dept:Department)
DETACH delete dept;
```

remove all properties keys

CREATING RELATION BETWEEN DEPARTMENT AND STAFF

```
MATCH (dept:Department)
MATCH (staff:Staff)
WHERE staff.IDDEPARTMENT = dept.IDDEPARTMENT
```

```
MERGE (staff)-[:WORKS_IN]->(dept)
RETURN dept, staff;
```

CREATING DOCTOR NODE

doctor is connected to staff node

```
// Load doctor CSV data
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-projeto/
neo4j/hospital-csv/doctor.csv/DOCTOR_DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWHWUP7J2S2
VYWDATT4ZRU4DSA' AS row

// Create doctor nodes with EMP_ID and QUALIFICATIONS properties
CREATE (:Doctor {
    EMP_ID: toInteger(row.EMP_ID),
    QUALIFICATIONS: row.QUALIFICATIONS
});

// Create primary key constraint on EMP_ID
CREATE CONSTRAINT FOR (d:Doctor) REQUIRE d.emp_id IS UNIQUE;
```

relation between Doctor and Staff

```
MATCH (doc:Doctor)
MATCH (staff:Staff)
WHERE doc.EMP_ID = staff.EMP_ID
MERGE (doc)-[:MEMBER_OF]->(staff)
RETURN doc, staff;
```

CREATING TECHNICIAN NODE

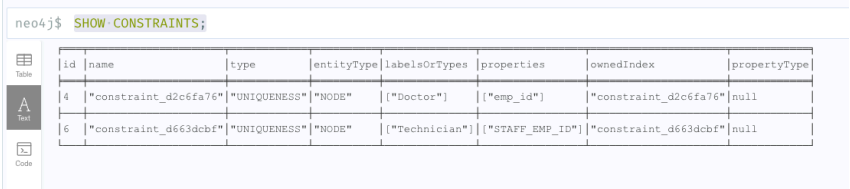
```
// Load doctor CSV data
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-projeto/
neo4j/hospital-csv/technician.csv/TECHNICIAN_DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWG
TNPF5OM5CUNFEP4EZSJAVZA' AS row

// Create technician nodes with STAFF_EMP_ID property
CREATE (:Technician {
    STAFF_EMP_ID: toInteger(row.STAFF_EMP_ID)
});

/* Primary  key for STAFF_EMP_ID */
CREATE CONSTRAINT FOR (t:Technician) REQUIRE t.STAFF_EMP_ID IS UNIQUE;
```

CREATING RELATION BETWEEN TECHNICIAN AND STAFF

```
MATCH (tech:Technician)
MATCH (staff:Staff)
WHERE tech.STAFF_EMP_ID = staff.EMP_ID
MERGE (tech)-[:IS_TECHNICIAN_FOR]->(staff)
RETURN tech, staff;
```

SHOW ALL UNIQUE CONSTRAINTS

```
SHOW CONSTRAINTS;
```



RENAMING RELATIONSHIP

```
MATCH (doc:Doctor)-[rel:MEMBER_OF]->(staff:Staff)
CREATE (doc)-[:IS_DOCTOR_FOR]->(staff)
DELETE rel
RETURN doc, staff;
```

Counting relationship MEMBER_OF to see if was deleted

```
MATCH ()-[rel:MEMBER_OF]->()
RETURN count(rel);
```

return 0

CREATING TECHNICIAN NODE

```
// Load doctor CSV data
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-
projeto/neo4j/hospital-csv/nurse.csv/NURSE_DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWGSL
BL6O77WIEF3SXAZSJY66Q' AS row

// Create technician nodes with STAFF_EMP_ID property
CREATE (:Nurse {
    STAFF_EMP_ID: toInteger(row.STAFF_EMP_ID)
});

/* Primary  key for STAFF_EMP_ID */
CREATE CONSTRAINT FOR (t:Nurse) REQUIRE t.STAFF_EMP_ID IS UNIQUE;
```

CREATE RELATION BETWEEN NURSE AND STAFF

```
MATCH (nur:Nurse)
MATCH (staff:Staff)
WHERE nur.STAFF_EMP_ID = staff.EMP_ID
MERGE (nur)-[:IS_NURSE_FOR]->(staff)
RETURN nur, staff;
```

CREATE HOSPITALIZATION NODE

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-projeto/
neo4j/hospital-csv/hospitalization.csv/HOSPITALIZATION_DATA_TABLE.csv?token=GHSAT0AAA
AAACPXCOWHINGK46EE3SPJ2JUSZSJZ4PQ' AS row

CREATE (:Hospitalization {
    ADMISSION_DATE: date(row.ADMISSION_DATE),
    DISCHARGE_DATE: date(row.DISCHARGE_DATE),
    ROOM_IDROOM: toInteger(row.ROOM_IDROOM),
    IDEPISODE: toInteger(row.IDEPISODE),
    RESPONSIBLE_NURSE: toInteger(row.RESPONSIBLE_NURSE)
});

CREATE CONSTRAINT FOR (h:Hospitalization) REQUIRE h.IDEPISODE IS UNIQUE;
```

Hospitalization

Important to use this we had to install a plugin APOC which is basically a jar file in our neo4j project. That can be done manually or automatically using neo4j browser

Neo4j is written in Java and Scala so most of the plugins are jar files. Apoc here provides important utility functions

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-projeto/
neo4j/hospital-csv/hospitalization.csv/HOSPITALIZATION_DATA_TABLE.csv?token=GHSAT0AAA
```

```
AAACPXCOWGBZHSCSGO4UBYP4X6ZSJ2BLA' AS row
WITH row,
     apoc.date.parse(row.ADMISSION_DATE, 'ms', 'dd-MM-yy') AS admissionMillis,
     apoc.date.parse(row.DISCHARGE_DATE, 'ms', 'dd-MM-yy') AS dischargeMillis
CREATE (:Hospitalization {
    ADMISSION_DATE: date(datetime({epochMillis: admissionMillis})),
           DISCHARGE_DATE:   CASE   WHEN   row.DISCHARGE_DATE   IS   NOT   NULL   THEN
date(datetime({epochMillis: dischargeMillis})) ELSE NULL END,
    ROOM_IDROOM: toInteger(row.ROOM_IDROOM),
    IDEPISODE: toInteger(row.IDEPISODE),
    RESPONSIBLE_NURSE: toInteger(row.RESPONSIBLE_NURSE)
});
```

Getting total number of Hospitalization nodes

```
MATCH (h:Hospitalization)
RETURN count(h) AS totalHospitalizations;
```

Which return 101 and that you can verify with the oracle db table

Hospitalization and Nurse Relation

Foreign key - `RESPONSIBLE_NURSE` is related to nurse table

```
MATCH (h:Hospitalization)
MATCH (n:Nurse)
WHERE h.RESPONSIBLE_NURSE = n.STAFF_EMP_ID
MERGE (h)-[:RESPONSIBLE_FOR]->(n);
```

Creating room node

```
LOAD   CSV   WITH   HEADERS   FROM   'https://raw.githubusercontent.com/anaritaasp/nosql-
projeto/neo4j/hospital-csv/room.csv/ROOM_DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWHHUME
JFS2C4YJIDX6ZSMCCKA' AS row

CREATE (:Room {
    IDROOM: toInteger(row.IDROOM),
    ROOM_TYPE: row.ROOM_TYPE,
    ROOM_COST: toInteger(row.ROOM_COST)
});

CREATE CONSTRAINT FOR (r:Room) REQUIRE r.IDROOM IS UNIQUE;
```

Relation between Hospitalization and Room

```
MATCH (h:Hospitalization)
MATCH (r:Room)
WHERE h.ROOM_IDROOM = r.IDROOM
MERGE (h)-[:ASSIGNED_TO]->(r);
```

Inserting EPISODE

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-projeto/
neo4j/hospital-csv/episode.csv/EPISODE_DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWGMOF5DS
2IRQLNIMXOZSMEXDA' AS row

CREATE (:Episode {
    IDEPISODE: toInteger(row.IDEPISODE),
    PATIENT_IDPATIENT: toInteger(row.PATIENT_IDPATIENT)
});

CREATE CONSTRAINT FOR (e:Episode) REQUIRE e.IDEPISODE IS UNIQUE;
```

Relation between Hospitalization and Episode

```
MATCH (h:Hospitalization)
MATCH (e:Episode)
WHERE h.IDEPISODE = e.IDEPISODE
MERGE (h)-[:INVOLVES]->(e);
```

Creating BILL NODE

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-projeto/
neo4j/hospital-csv/bill.csv/BILL_DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWGOCQIUBXCBIBM
46DWZSMFJAQ' AS row
WITH row,
          datetime({epochMillis:  apoc.date.parse(row.REGISTERED_AT,  'ms',  'yy-MM-dd
hh:mm:ss.SSSSSSSSS a')}) AS registeredAt

CREATE (:Bill {
    IDBILL: toInteger(row.IDBILL),
    ROOM_COST: toInteger(row.ROOM_COST),
    TEST_COST: toInteger(row.TEST_COST),
    OTHER_CHARGES: toInteger(row.OTHER_CHARGES),
    TOTAL: toInteger(row.TOTAL),
    IDEPISODE: toInteger(row.IDEPISODE),
    REGISTERED_AT: registeredAt,
    PAYMENT_STATUS: row.PAYMENT_STATUS
});

CREATE CONSTRAINT FOR (b:Bill) REQUIRE b.IDBILL IS UNIQUE;
```

Create relation between BILL AND

```
MATCH (b:Bill), (e:Episode)
WHERE b.IDEPISODE = e.IDEPISODE
MERGE (b)-[:BILLED_FOR]->(e);
```

CREATE PRESCRIPTION

```
LOAD  CSV  WITH  HEADERS  FROM  'https://raw.githubusercontent.com/anaritaasp/nosql-
projeto/neo4j/hospital-csv/prescription.csv/PRESCRIPTION_DATA_TABLE.csv?token=YOUR_
GITHUB_TOKEN' AS row
WITH    row,    apoc.date.parse(row.PRESCRIPTION_DATE,    'ms',    'dd-MM-yy')    AS
prescriptionMillis

CREATE (:Prescription {
    IDPRESCRIPTION: toInteger(row.IDPRESCRIPTION),
    PRESCRIPTION_DATE: date(datetime({epochMillis: prescriptionMillis})),
    DOSAGE: toInteger(row.DOSAGE),
    IDMEDICINE: toInteger(row.IDMEDICINE),
    IDEPISODE: toInteger(row.IDEPISODE)
});

CREATE CONSTRAINT FOR (p:Prescription) REQUIRE p.IDPRESCRIPTION IS UNIQUE;
```

Relation between prescription and episode

```
MATCH (p:Prescription), (e:Episode)
WHERE p.IDEPISODE = e.IDEPISODE
MERGE (p)-[:PRESCRIBED_FOR]->(e);
```

Medicine node

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-projeto/
neo4j/hospital-csv/medicine.csv/MEDICINE_DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWHZ3
HKJMTOYMTW242CZSMGNIQ' AS row

CREATE (:Medicine {
    IDMEDICINE: toInteger(row.IDMEDICINE),
    M_NAME: row.M_NAME,
    M_QUANTITY: toInteger(row.M_QUANTITY),
    M_COST: toFloat(row.M_COST)
});

CREATE CONSTRAINT FOR (m:Medicine) REQUIRE m.IDMEDICINE IS UNIQUE;


MATCH (p:Prescription), (m:Medicine)
WHERE p.IDMEDICINE = m.IDMEDICINE
MERGE (p)-[:PRESCRIBED_MEDICINE]->(m);
```

Appointment node

```
// Load appointment CSV data
OAD    CSV    WITH    HEADERS    FROM    'https://raw.githubusercontent.com/anaritaasp/
nosql-projeto/neo4j/hospital-csv/appointment.csv/APPOINTMENT_DATA_TABLE.csv?token=
YOUR_GITHUB_TOKEN' AS row
WITH row,
     apoc.date.parse(row.SCHEDULED_ON, 'ms', 'dd-MM-yy') AS scheduledMillis,
     apoc.date.parse(row.APPOINTMENT_DATE, 'ms', 'dd-MM-yy') AS appointmentMillis
CREATE (:Appointment {
    SCHEDULED_ON: apoc.date.convertFormat(row.SCHEDULED_ON, 'dd-MM-yy', 'yyyy-MM-dd'),
    APPOINTMENT_DATE: apoc.date.convertFormat(row.APPOINTMENT_DATE, 'dd-MM-yy', 'yyyy-
MM-dd'),
    APPOINTMENT_TIME: row.APPOINTMENT_TIME,
    IDDOCTOR: toInteger(row.IDDOCTOR),
    IDEPISODE: toInteger(row.IDEPISODE)
});


// Create a unique constraint on APPOINTMENT_DATE, APPOINTMENT_TIME, IDDOCTOR, and
IDEPISODE combination
CREATE CONSTRAINT FOR (a:Appointment) REQUIRE (a.APPOINTMENT_DATE, a.APPOINTMENT_TIME,
a.IDDOCTOR, a.IDEPISODE) IS UNIQUE;
```

Relation of appointment with doctor and appointment

```
// Match Appointment nodes with corresponding Doctor nodes and create relationships
MATCH (a:Appointment), (d:Doctor)
WHERE a.IDDOCTOR = d.EMP_ID
MERGE (a)-[:HAS_DOCTOR]->(d);

// Match Appointment nodes with corresponding Episode nodes and create relationships
MATCH (a:Appointment), (e:Episode)
WHERE a.IDEPISODE = e.IDEPISODE
MERGE (a)-[:BELONGS_TO_EPISODE]->(e);
```

Lab Screening

```
// Load Lab_Screening CSV data
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/anaritaasp/nosql-projeto/
neo4j/hospital-csv/lab_screening.csv/LAB_SCREENING_DATA_TABLE.csv?token=GHSAT0AAAAAAC
```

```
PXCOWHSGB3VCOCZZEY4XCIZSMHKTA' AS row
WITH row, apoc.date.parse(row.TEST_DATE, 'ms', 'dd-MM-yy') AS testMillis
WITH row, apoc.date.format(testMillis, 'ms', 'yyyy-MM-dd') AS testDate
CREATE (:Lab_Screening {
    LAB_ID: toInteger(row.LAB_ID),
    TEST_COST: toFloat(row.TEST_COST),
    TEST_DATE: date(testDate),
    IDTECHNICIAN: toInteger(row.IDTECHNICIAN),
    EPISODE_IDEPISODE: toInteger(row.EPISODE_IDEPISODE)
});

// Create a unique constraint on LAB_ID
CREATE CONSTRAINT FOR (ls:Lab_Screening) REQUIRE ls.LAB_ID IS UNIQUE;
```

Lab Screening relation with Episode and Technician

```
MATCH (ls:Lab_Screening), (ep:Episode)
WHERE ls.EPISODE_IDEPISODE = ep.IDEPISODE
MERGE (ls)-[:BELONGS_TO]->(ep)
RETURN ls, ep;

MATCH (ls:Lab_Screening), (tech:Technician)
WHERE ls.IDTECHNICIAN = tech.STAFF_EMP_ID
MERGE (ls)-[:PERFORMED_BY]->(tech)
RETURN ls, tech;
```

Adding PATIENT

```
// Load the CSV file and parse the BIRTHDAY date
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/AbhimanyuAryan/nosql-
database-evaluation/neo4j/hospital-csv/patient.csv/PATIENT_DATA_TABLE.csv?token=GHSAT
0AAAAAACPXCOWGURVLXYTTEKXC2PCKZSM6CWQ' AS row
WITH row, apoc.date.parse(row.BIRTHDAY, 'ms', 'dd-MM-yy') AS birthMillis
WITH row, apoc.date.format(birthMillis, 'ms', 'yyyy-MM-dd') AS birthDate
CREATE (:Patient {
    IDPATIENT: toInteger(row.IDPATIENT),
    PATIENT_FNAME: row.PATIENT_FNAME,
    PATIENT_LNAME: row.PATIENT_LNAME,
    BLOOD_TYPE: row.BLOOD_TYPE,
    PHONE: row.PHONE,
    EMAIL: row.EMAIL,
    GENDER: row.GENDER,
    POLICY_NUMBER: row.POLICY_NUMBER,
    BIRTHDAY: date(birthDate)
});

// Create a unique constraint on IDPATIENT
CREATE CONSTRAINT FOR (p:Patient) REQUIRE p.IDPATIENT IS UNIQUE;
```

Relation between patient and episode

```
// Create relationships between Patient and Episode based on the foreign key
PATIENT_IDPATIENT
MATCH (p:Patient), (e:Episode)
WHERE p.IDPATIENT = e.PATIENT_IDPATIENT
MERGE (p)-[:HAS_EPISODE]->(e)
RETURN p, e;
```

Create EMERGENCY_CONTACT

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/AbhimanyuAryan/nosql-
database-evaluation/neo4j/hospital-csv/emergency_contact.csv/EMERGENCY_CONTACT_DATA_
TABLE.csv?token=GHSAT0AAAAAACPXCOWGOXRE2DQ2DTIIOIBMZSM7AJQ' AS row
CREATE (:Emergency_Contact {
    CONTACT_NAME: row.CONTACT_NAME,
    PHONE: row.PHONE,
    RELATION: row.RELATION,
    IDPATIENT: toInteger(row.IDPATIENT)
});
```

IDPATIENT and Phone both primary key constraint for EMERGENCY_CONTACT

```
CREATE CONSTRAINT FOR (ec:Emergency_Contact) REQUIRE (ec.IDPATIENT, ec.PHONE) IS UNIQUE;
```

Relation between EMERGENCY_CONTACT and PATIENT

```
MATCH (p:Patient), (ec:Emergency_Contact)
WHERE p.IDPATIENT = ec.IDPATIENT
MERGE (ec)-[:CONTACT_FOR]->(p)
RETURN ec, p;
```

Insurance Node

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/AbhimanyuAryan/nosql-
database-evaluation/neo4j/hospital-csv/insurance.csv/INSURANCE_DATA_TABLE.csv?token=
GHSAT0AAAAAACPXCOWG4336DUF2BIN25QHEZSNB22A' AS row
CREATE (:Insurance {
    POLICY_NUMBER: row.POLICY_NUMBER,
    PROVIDER: row.PROVIDER,
    INSURANCE_PLAN: row.INSURANCE_PLAN,
    CO_PAY: toInteger(row.CO_PAY),
    COVERAGE: row.COVERAGE,
    MATERNITY: row.MATERNITY,
    DENTAL: row.DENTAL,
    OPTICAL: row.OPTICAL
});

// Create a unique constraint on POLICY_NUMBER
CREATE CONSTRAINT FOR (i:Insurance) REQUIRE i.POLICY_NUMBER IS UNIQUE;
```

Creating relation between insurance and patient

```
MATCH (p:Patient), (i:Insurance)
WHERE p.POLICY_NUMBER = i.POLICY_NUMBER
MERGE (p)-[:HAS_INSURANCE]->(i);
```

Check if there's a relationship between patient and emergency_contact

```
MATCH (p:Patient)-[r]->(ec:Emergency_Contact)
RETURN p.IDPATIENT, p.PATIENT_FNAME, p.PATIENT_LNAME, ec.CONTACT_NAME, ec.PHONE,
type(r) AS relationshipType, r;
```

Add node medical_history

```
// Load CSV and create Medical_History nodes with the correct date format
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/AbhimanyuAryan/
nosql-database-evaluation/neo4j/hospital-csv/medical_history.csv/MEDICAL_HISTORY_
DATA_TABLE.csv?token=GHSAT0AAAAAACPXCOWHLEZTFAHDBWD2RRLAZSNDICQ' AS row
WITH row,
     apoc.date.parse(row.RECORD_DATE, 'ms', 'dd-MM-yy') AS recordMillis
WITH row,
     apoc.date.format(recordMillis, 'ms', 'yyyy-MM-dd') AS recordDate
```

```
CREATE (:Medical_History {
    RECORD_ID: toInteger(row.RECORD_ID),
    CONDITION: row.CONDITION,
    RECORD_DATE: date(recordDate),
    IDPATIENT: toInteger(row.IDPATIENT)
});


// Create a unique constraint on RECORD_ID
CREATE CONSTRAINT FOR (mh:Medical_History) REQUIRE mh.RECORD_ID IS UNIQUE;
```

Patient has medical history

```
MATCH (p:Patient), (mh:Medical_History)
WHERE p.IDPATIENT = mh.IDPATIENT
CREATE (p)-[:HAS_MEDICAL_HISTORY]->(mh)
```

QUERIES

Get Patient by ID

```
MATCH(p:Patient {IDPATIENT: 1})
RETURN p
```

Get all Episodes for a specific Patient

```
MATCH(p:Patient {IDPATIENT: 1})-[:HAS_EPISODE]->(e:Episode)
RETURN e
```

Get all patients with specific blood type

```
MATCH(p:Patient {BLOOD_TYPE: 'O-'})
RETURN p
```

Find the average age of all Patients

```
MATCH (p:Patient)
RETURN avg(date().year - p.BIRTHDAY.year) AS averageAge
```

Get patient phone number and update it

```
MATCH (p:Patient {IDPATIENT:1})
return p.PHONE

MATCH (p:Patient {IDPATIENT:1})
SET p.PHONE = '555-6789'
RETURN p
```

See patients medical history and add new condition to medical history

```
MATCH (p:Patient {IDPATIENT:1})-[:HAS_MEDICAL_HISTORY]->(mh)
RETURN p, mh

MATCH (p:Patient {IDPATIENT: 1})
CREATE (mh:Medical_History {
    RECORD_ID: 47,
    CONDITION: 'Back Pain',
    RECORD_DATE: date('2024-05-20'),
    IDPATIENT: 1
})
CREATE (p)-[:HAS_MEDICAL_HISTORY]->(mh)
RETURN p, mh
```

Delete a Patient and all their Episodes

```
MATCH (p:Patient {IDPATIENT: 2}) - [:HAS_EPISODE] -> (e:Episode)
DETCH DELETE p, e
```

Remove a specific episode for a patient

```
MATCH (p:Patient {IDPATIENT: 1}) - [:HAS_EPISODE] -> (e: Episode {IDEPISODE: 180})
DETACH DELETE e
```

Find Patients who have been prescribed a specific medicine

```
MATCH (p:Patient)-[:HAS_EPISODE]->(e:Episode)<-[:PRESCRIBED_FOR]-(pr:Prescription)-
[:PRESCRIBED_MEDICINE]->(m:Medicine {M_NAME: 'Paracetamol'})
RETURN p
```

Get the total cost of all bills for a specific Patient

```
MATCH (p:Patient {IDPATIENT: 3})-[:HAS_EPISODE]->(e:Episode)<-[:BILLED_FOR]-(b:Bill)
RETURN sum(b.TOTAL) AS totalCost
```

Find all Patients who have an appointment on a specific date

```
/* Find relation between Episode and Appoitnment */
MATCH (:Episode)-[r]-(:Appointment)
RETURN type(r)

/* Once you find relation you use it */
MATCH (p:Patient)-[:HAS_EPISODE]->(e:Episode)<-[:BELONGS_TO_EPISODE]-(a:Appointment)
WHERE a.APPOINTMENT_DATE = datetime("2018-11-29T00:00:00Z")
RETURN p
```