LAB 3 REPORT

1) TASK1

a) File importation

After lunching the server with the command **(<path>/neo4j console)** , we use a browser to connect a client to the server using the following **URL :localhost:7474.** From the command prompt in the browser's interface, we enter the following command to load the file **:**
LOAD CSV WITH HEADERS FROM "file:///boson_crime_file" AS row CREATE (:BostonCrime {INCIDENT_NUMBER: row.INCIDENT_NUMBER, OFFENSE_CODE_GROUP: OFFENSE_CODE_GROUP}); we repeat the same procedure for the lookup file
NB: this command works only if the file is in the IMPORT directory of neo4j and it's a practice recommended in the documentation.
b) Modelisation of data as property graph:

After the importation, each element in the table will corresponds to a node where the label is associated to the table it comes from; the attributes are stored in the node as properties.
We added relationship between nodes using the following commands:
MATCH(b_c:BostonCrime),(b_l:BostonLookup)
WHERE b_c.OFFENSE_CODE = b_l.CODE
CREATE(boston_crime) –[:HAS_NAME]->(boston_lookup)

2) TASK2

a) To solve this question, we select from the nodes belonging to the label "BostonCrime" the ones where the property "OFFENSE_CODE_GROUP" is equal to "Drug Violation". The query used is the following:

MATCH (Boston_crime : bostonCrime)
WHERE boston_crime.OFFENSE_CODE_GROUP = "Drug Violation"
Return boston_crime.OFFENSE_CODE_GROUP
,count(boston_crime.OFFENSE_CODE_GROUP) as number_of_incidents. The related picture is **task2_1**

b) This question is solved selecting from the nodes belonging to bostonCrime the ones where the attribute OFFENSE:CODE:GROUP = "Investigate Person", then making a join with the nodes of "boston_lookup". The query used is the following:

MATCH (b_c:BostonCrime{OFFENSE_CODE_GROUP:"Investigate Person"})-[:HAS_NAME]->(b_l:BostonLookup)
Return b_c.INCIDENT_NUMBER,b_c.OFFENSE_CODE_GROUP ,b_l.NAME as NAME; The related picture is **task2_2.PNG**.
c) The pictures related to this task are **task2_eplain_Query1.PNG, task2_profile_Query1.PNG** for the first query, and **task2_eplain_Query2, task2_profile_Query2** for query 2.

d) To add indexes, I used the following commands **CREATE INDEX ON: BostonCrime(OFFCENSE_CODE_GROUP), CREATE INDEX ON Bostonlookup:(CODE),CREATE INDEX ON :BostonCrime(OFFENSE_CODE).**

1. CHANGES IN QUERY 1
   They are some important changes in the execution plan. First of the the strategy used for scanning is no more nodeByLabelScan but nodeIndexSeek which is faster. Moreover, the number of stages in the execution plan is reduced because the filter stage has been removed, and finally the number of rows at the exit of each stage is considerably reduced. The related pictures are **task2_profile_query1_index.PNG and task2_explain_query1_index.PNG.**
2. CHANGES IN QUERY 2
   We can notice that the execution plan of query 2 has also changed a lot. First of all the scan strategy is no more NodebyLabelScan but NodeIndexSeek as for query 1, also the filter related to the where condition has been removed because the first stage has already selected data satisfying the where condition, and finally the total number of rows at the exit of each stage in the plan has been reduced. The related pictures are **task2_profile_query2_index.PNG and task2_explain_query2_index.PNG.**

e) The chosen query is the total number of crime per group ordered in ascending order.

```
MATCH(b_c:bostonCrime)
Return b_c.OFFENSE_CODE_GROUP as OFFENSE_CODE_GROUP,count(*) as
tot_crime_per_group
```
Order by tot_crime__per_group. The related picture is **task2_5**

3 Task3

We can notice that since there are too many nodes, the obtained graph is not quite easy to read.But in general, if a graph contains few nodes visualizing it allows to better understand relationship between data. The related picture is **tak3.png**.