Please indicate below the tool you have analyzed for Homework 3, and past the code you have realized.
Your code must include all queries and indexes (if any), and possibly the script you have used to populate the database (in case you used a tool for this, e.g., compass for MongDB, please specify). If you have interacted with the database system through an external programming language, e.g., Python, insert your functions/program.

==========

We had 4 tables, each one refers to users, vehicles involved in accidents, characteristics and places where accidents took place in.
To retrieve efficiently the several actors of each accident and to fast research through nodes in our queries on Neo4J, we've just divided the Users table into 4 types of nodes, Drivers, Passengers, Pedestrians and Riders.
First step was to create all the primary keys and indexes to traverses graph more quickly and efficiently.

Primary keys on Acc property for characteristics and places, on ID for all type of nodes:

```
//A_primary key on acc for characteristics
CREATE CONSTRAINT FOR (d:Characteristics) REQUIRE d.Acc IS UNIQUE

//A_primary key on acc for places
CREATE CONSTRAINT FOR (p:Places) REQUIRE p.Acc IS UNIQUE

//A_primary key on id for characteristics
CREATE CONSTRAINT FOR (c:Characteristics) REQUIRE c.ID IS UNIQUE

//A_primary key on id for drivers
CREATE CONSTRAINT FOR (d:Drivers) REQUIRE d.ID IS UNIQUE

//A_primary key on id for passengers
CREATE CONSTRAINT FOR (p:Passengers) REQUIRE p.ID IS UNIQUE

//A_primary key on id for pedestrians
CREATE CONSTRAINT FOR (p:Pedestrians) REQUIRE p.ID IS UNIQUE

//A_primary key on id for places
CREATE CONSTRAINT FOR (p:Places) REQUIRE p.ID IS UNIQUE

//A_primary key on id for riders
CREATE CONSTRAINT FOR (r:Riders) REQUIRE r.ID IS UNIQUE

//A_primary key on id for vehicles
CREATE CONSTRAINT FOR (p:Vehicles) REQUIRE p.ID IS UNIQUE
```

Indexes on Acc property for nodes where it wasn't PK and on Safety in Drivers and on Surface in Places:

```
//B_index on acc for drivers
CREATE INDEX IndAccD FOR (d:Drivers) ON (d.Acc)

//B_index on acc for passengers
CREATE INDEX IndAccP FOR (p:Passengers) ON (p.Acc)

//B_index on acc for pedestrians
CREATE INDEX IndAccP2 FOR (p:Pedestrians) ON (p.Acc)

//B_index on acc for riders
CREATE INDEX IndAccR FOR (r:Riders) ON (r.Acc)

//B_index on acc for vehicles
CREATE INDEX IndAccV FOR (v:Vehicles) ON (v.Acc)

//B_index on safety for drivers
CREATE INDEX IndSafety FOR (d:Drivers) ON (d.Safety)

//B_index on surface for places
CREATE INDEX IndSurf FOR (p:Places) ON (p.surface_condition)
```

After splitting Users into 4 categories as described before: Drivers, Passengers, Riders, Pedestrians, we proceed to import the data using APOC plugin of Neo4J and the CALL 'apoc.periodic.iterate' construct for all nodes (import using batches of 1000):

```
//import characteristics
CALL apoc.periodic.iterate(
  'LOAD CSV FROM "file:///cleaned_characteristics.csv" AS row RETURN
row',
  'CREATE (:Characteristics {ID: toInteger(row[1]), Acc:
toInteger(row[2]), Year: toInteger(row[3]), Month: toInteger(row[4]),
Day: toInteger(row[5]), HourMin: toInteger(row[6]), Lighting:
toInteger(row[7]), Builtup: toInteger(row[8]), Atm: toInteger(row[9]),
Collision: toInteger(row[10]) })',
  { batchSize: 1000, parallel: false }
)


//import drivers
CALL apoc.periodic.iterate(
  'LOAD CSV FROM "file:///drivers.csv" AS row RETURN row',
  'CREATE (:Drivers {ID: toInteger(row[2]), Acc: toInteger(row[3]),
User: toInteger(row[4]), Severity: toInteger(row[5]), Sex:
toInteger(row[6]), Trip: toInteger(row[7]), Safety: toInteger(row[8]),
Pedmove: toInteger(row[9]), Birth: toInteger(row[10]), Plate:
row[11]})',
```

```
  { batchSize: 1000, parallel: false }
)


//import passengers
CALL apoc.periodic.iterate(
  'LOAD CSV FROM "file:///passengers.csv" AS row RETURN row',
  'CREATE (:Passengers {ID: toInteger(row[2]), Acc: toInteger(row[3]),
User: toInteger(row[4]), Severity: toInteger(row[5]), Sex:
toInteger(row[6]), Trip: toInteger(row[7]), Safety: toInteger(row[8]),
Pedmove: toInteger(row[9]), Birth: toInteger(row[10]), Plate:
row[11]})',
  { batchSize: 1000, parallel: false }
)


//import pedestrians
CALL apoc.periodic.iterate(
  'LOAD CSV FROM "file:///pedestrians.csv" AS row RETURN row',
  'CREATE (:Pedestrians{ID: toInteger(row[2]), Acc: toInteger(row[3]),
User: toInteger(row[4]), Severity: toInteger(row[5]), Sex:
toInteger(row[6]), Trip: toInteger(row[7]), Safety: toInteger(row[8]),
Pedmove: toInteger(row[9]), Birth: toInteger(row[10]), Plate:
row[11]})',
  { batchSize: 1000, parallel: false }
)


//import places
CALL apoc.periodic.iterate(
  'LOAD CSV FROM "file:///cleaned_places.csv" AS row RETURN row',
  'CREATE (:Places {ID: toInteger(row[1]), Acc: toInteger(row[2]),
type_road: toInteger(row[3]), traffic_regime: toInteger(row[4]),
road_profile: toInteger(row[5]), road_plan: toInteger(row[6]),
road_width: toInteger(row[7]), surface_condition: toInteger(row[8]),
road_infrastructure: toInteger(row[9]), place_accident:
toInteger(row[10]) })',
  { batchSize: 10000, parallel: false }
)


//import riders
CALL apoc.periodic.iterate(
  'LOAD CSV FROM "file:///riders.csv" AS row RETURN row',
  'CREATE (:Riders {ID: toInteger(row[2]), Acc: toInteger(row[3]),
User: toInteger(row[4]), Severity: toInteger(row[5]), Sex:
toInteger(row[6]), Trip: toInteger(row[7]), Safety: toInteger(row[8]),
Pedmove: toInteger(row[9]), Birth: toInteger(row[10]), Plate:
row[11]})',
```

```
  { batchSize: 1000, parallel: false }
)


//import vehicles
CALL apoc.periodic.iterate(
  'LOAD CSV FROM "file:///cleaned_vehicles.csv" AS row RETURN row',
  'CREATE (:Vehicles {ID: toInteger(row[1]), Acc: toInteger(row[2]),
Category: toInteger(row[3]), Plate: row[4] })',
  { batchSize: 10000, parallel: false }
)
```

Then we proceed to create the relationships, mostly bidirectional except two of them those going from 'Characteristics' nodes to 'Places' nodes and from 'Vehicles' to ‘Characteristics’:

```
//Relationship Driver/Vehicles
CALL apoc.periodic.iterate(
  'MATCH (d:Drivers) RETURN d',
  'MATCH (d:Drivers)
   MATCH (v:Vehicles { Acc: d.Acc, Plate: d.Plate })
   CREATE (d)-[:DRIVES { Acc: d.Acc, Plate: d.Plate }]->(v)
   CREATE (v)-[:DRIVEN_BY { Acc: v.Acc, Plate: v.Plate }]->(d)',
  { batchSize: 1000, parallel: false }
)


//Relationship Pedestrians/Vehicles
CALL apoc.periodic.iterate(
  'MATCH (p:Pedestrians) RETURN p',
  'MATCH (p:Pedestrians)
   MATCH (v:Vehicles { Acc: p.Acc, Plate: p.Plate })
   CREATE (p)-[:HIT_BY { Acc: p.Acc, Plate: p.Plate }]->(v)
   CREATE (v)-[:RAN_OVER { Acc: v.Acc, Plate: v.Plate }]->(p)',
  { batchSize: 1000, parallel: false }
)

//Relationships Drivers/Charact
CALL apoc.periodic.iterate(
  'MATCH (d:Drivers) RETURN d',
  'MATCH (d:Drivers)
   MATCH (c:Characteristics { Acc: d.Acc })
   CREATE (d)-[:INVOLVED_IN2 { Acc: d.Acc, Severity: d.Severity,
Lighting:c.Lighting, Atm: c.Atm  }]->(c)
   CREATE (c)-[:INVOLVES2 { Acc: d.Acc, Severity: d.Severity,
Lighting:c.Lighting, Atm: c.Atm  }]->(d)',
  { batchSize: 1000, parallel: false }
)
```

```
//Relationships Passengers/Charact
CALL apoc.periodic.iterate(
  'MATCH (p:Passengers) RETURN p',
  'MATCH (p:Passengers)
   MATCH (c:Characteristics { Acc: p.Acc })
   CREATE (p)-[:INVOLVED_IN3 { Acc: p.Acc, Severity: p.Severity,
Lighting:c.Lighting, Atm: c.Atm  }]->(c)
   CREATE (c)-[:INVOLVES3 { Acc: p.Acc, Severity: p.Severity,
Lighting:c.Lighting, Atm: c.Atm  }]->(p)',
  { batchSize: 1000, parallel: false }
)

//Relationships Passengers/Vehicles
CALL apoc.periodic.iterate(
  'MATCH (p:Passengers) RETURN p',
  'MATCH (p:Passengers)
   MATCH (v:Vehicles { Acc: p.Acc, Plate: p.Plate })
   CREATE (p)-[:INVOLVED_WITH { Acc: p.Acc, Plate: p.Plate }]->(v)
   CREATE (v)-[:CARRYING { Acc: v.Acc, Plate: v.Plate }]->(p)',
  { batchSize: 1000, parallel: false }
)


//Relationships Pedestrians/Charact
CALL apoc.periodic.iterate(
  'MATCH (p:Pedestrians) RETURN p',
  'MATCH (p:Pedestrians)
   MATCH (c:Characteristics { Acc: p.Acc })
   CREATE (p)-[:INVOLVED_IN5 { Acc: p.Acc, Severity: p.Severity,
Lighting:c.Lighting, Atm: c.Atm  }]->(c)
   CREATE (c)-[:INVOLVES5 { Acc: p.Acc, Severity: p.Severity,
Lighting:c.Lighting, Atm: c.Atm  }]->(p)',
  { batchSize: 1000, parallel: false }
)

//Relationships Places/Charact
CALL apoc.periodic.iterate(
  'MATCH (p:Places) RETURN p',
  'MATCH (p:Places)
   MATCH (c:Characteristics { Acc: p.Acc })
   CREATE (c)-[:TOOK_PLACE_IN{ Acc: p.Acc, type_road:p.type_road,
surface_condition:p.surface_condition}]->(p)',
  { batchSize: 1000, parallel: false }
)
```

```
//Relationships Riders/Charact
CALL apoc.periodic.iterate(
  'MATCH (r:Riders) RETURN r',
  'MATCH (r:Riders)
   MATCH (c:Characteristics { Acc: r.Acc })
   CREATE (r)-[:INVOLVED_IN4 { Acc: r.Acc, Severity: r.Severity,
Lighting:c.Lighting, Atm: c.Atm  }]->(c)
   CREATE (c)-[:INVOLVES4 { Acc: r.Acc, Severity: r.Severity,
Lighting:c.Lighting, Atm: c.Atm  }]->(r)',
  { batchSize: 1000, parallel: false }
)



//Relationships Riders/Vehicles
CALL apoc.periodic.iterate(
  'MATCH (r:Riders) RETURN r',
  'MATCH (r:Riders)
   MATCH (v:Vehicles { Acc: r.Acc, Plate: r.Plate })
   CREATE (r)-[:HIT_BY { Acc: r.Acc, Plate: r.Plate }]->(v)
   CREATE (v)-[:RAN_OVER { Acc: v.Acc, Plate: v.Plate }]->(r)',
  { batchSize: 1000, parallel: false }
)

//Relationships Vehicles/Charact
CALL apoc.periodic.iterate(
  'MATCH (v:Vehicles) RETURN v',
  'MATCH (v:Vehicles)
   MATCH (c:Characteristics { Acc: v.Acc })
   CREATE (v)-[:USED_IN { Acc: v.Acc, Category:v.Category }]->(c)',
  { batchSize: 1000, parallel: false }
)
```

Finally, we create all the corresponding queries we have done for HW1 and HW2, we've inserted explanation of numbers related to properties used in all queries to better understand results:

```
//-- QUERY 1. 'NUMERO DI INCIDENTI NOTTURNI E NON IN BASE ALLE
CONDIZIONI ATMOSFERICHE'

//-- atmospheric condition :
//-- 1 normal, 2 soft_rain, 3 heavy_rain, 4 snow, 5 fog, 6 heavy_wind
//-- lighting condition:
//-- 3,4,5 accidents by night in different situations (based on public
```

```
illumination // or not)

MATCH (c:Characteristics)
      WHERE  NOT c.Atm  IN [7,8,9]
      WITH c.Atm as atmospheric_conditions,
      SUM(CASE WHEN c.Lighting IN [3, 4, 5] THEN 1 ELSE 0 END) as
night,
      SUM(CASE WHEN NOT c.Lighting  IN [3, 4, 5] THEN 1 ELSE 0 END) as
not_night
RETURN atmospheric_conditions, night, not_night



//-- QUERY 2. 'CONDUCENTI GRAVEMENTE FERITI O MORTI COINVOLTI IN
INCIDENTI SU MANTO STRADALE NORMALE'

//-- severity: 2 dead user, 3 heavily injured user
//-- type_user: 1 driver
//-- surface_condition: 1 normal

MATCH (d:Drivers)-[:INVOLVED_IN2]->(c:Characteristics)
MATCH (c)-[:TOOK_PLACE_IN]->(p:Places)
WHERE d.Severity IN [2, 3]
  AND p.surface_condition = 1
RETURN DISTINCT d.ID, d.Acc, d.Severity



//-- QUERY 3. 'NUMERO DI INCIDENTI DIVISI PER CATEGORIA DI VEICOLO'

//-- category of vehicles: 1 bycycle, 2 scooter, 3 quad, 7 cars, 13
camion
//-- we have take into account only these five types of vehicles
beacuse we //didn't have an explanation for the others

MATCH (v:Vehicles)
WHERE v.Category IN [1, 2, 3, 7, 13]
WITH v.Category AS cat, COUNT(DISTINCT v.Acc) AS accidents
RETURN cat, accidents



//-- QUERY 4. 'VISUALIZZA IL NUMERO DI INCIDENTI CON COINVOLGIMENTO DI
TRE O PIU' VEICOLI  (4, 5, 6)
//suddivisi per tipologia di strada
//-- (autostrada(1), superstrada(2), extraurbane(3), urbane(4),
campagna(5))
//-- in condizioni atmosferiche non buone (forte pioggia(3), neve(4),
forte //vento(6))

//-- collision: three or more vehicles involved
```

```
//-- atm: 3 heavy rain, 4 snow, 6 heavy wind
//-- type_road: 1 highway, 2 expressway, 3 suburban, 4 urban, 5 country
roads

MATCH (c:Characteristics)-[:TOOK_PLACE_IN]->(p:Places)
WHERE c.Collision IN [4, 5, 6] AND c.Atm IN [3, 4, 6]
WITH p.type_road AS type_road, COUNT(DISTINCT c.Acc) AS accidents
WHERE NOT type_road  IN [6, 9]
RETURN type_road, accidents
ORDER BY accidents DESC



//-- QUERY 5. Visualizza il numero di incidenti con solo due veicoli o
senza collisioni in cui il conducente  ha un età inferiore ai 35 anni,
suddivisi in quelli notturni o all'alba durante il rientro da feste o
eventi (5 o 9) e quelli diurni durante spostamenti per lavoro o scuola
(1,2,4) e il numero di incidenti in altre condizioni

//- collision: in (1,2,3,7) only two vehicles involved or no
collision(7)
//- lighting: by night is <> 1, on day = 1
//- trip: reason for journey (5,9) free-time or party, (1,2,4) from
home to //school or work

MATCH (d:Drivers) - [:INVOLVED_IN2] -> (c:Characteristics)
WHERE c.Collision in [1,2,3,7] AND c.Year - d.Birth < 35
WITH CASE
      WHEN c.Lighting <> 1 AND d.Trip IN [5,9] THEN 'Notturni o
all\'alba durante rientro da feste o eventi'
      WHEN c.Lighting <> 1 AND d.Trip IN [1,2,4] THEN 'Diurni durante
spostamenti per lavoro o scuola'
      ELSE 'Altre condizioni'
      END AS type_accident, COUNT(DISTINCT c.Acc) as accidents
RETURN type_accident, accidents



//-- QUERY 6. Visualizza il numero di incidenti in ordine decrescente
su vari tipi di manto stradale  -- (normale, bagnato, con pozzanghere,
innevato, ghiacciato) in cui tutti i conducenti  -- portavano la
cintura di sicurezza


//-- surface condition: 1 normal, 2 wet, 4 flooded, 5 snowy, 7 iced)
//-- safety equipment: drivers with belt is equal to 11

MATCH (d:Drivers)
WHERE d.Severity <> 11
WITH  COLLECT(d) AS excludedDrivers
```

```
MATCH (c:Characteristics)
WHERE NOT EXISTS {
      MATCH(x:Drivers) -[:INVOLVED_IN2]->(c)
      WHERE x  in excludedDrivers
}
WITH c

MATCH (c)-[:TOOK_PLACE_IN]->(p:Places)
WHERE p.surface_condition IN [1, 2, 4, 5, 7]
RETURN p.surface_condition, COUNT(*) as tot
ORDER BY tot DESC



//-- QUERY 7. Per ogni incidente con almeno un ferito grave o mortale ,
visualizza il numero totale di utenti, la tipologia dell'utente,  -- il
numero totale di veicoli e la tipologia di veicolo  coinvolti



//-- type user: 1 drivers, 2 passengers, 3 pedestrains, 4 riders
//-- category  of vehicles: 1 bycycle, 2 scooter, 3 quad, 7 cars, 13
camion
//-- severity of users in the accident: 2 dead, 3 heavily injured

MATCH (v:Vehicles)
WHERE v.Category in [1,2,3,7,13]
 MATCH (v)-[:DRIVEN_BY|:RAN_OVER|:INVOLVED_WITH|:RAN_OVER2]->(e)
WHERE e.Severity IN [2,3]
RETURN v.Acc,COUNT(v.Acc) as TotalUsers,SUM(CASE WHEN e.User = 1 THEN 1
ELSE 0 END) as Driver,SUM(CASE WHEN e.User = 2 THEN 1 ELSE 0 END) as
Passengers,SUM(CASE WHEN e.User = 3 THEN 1 ELSE 0 END) as
Pedestrians,COUNT(DISTINCT v.target) as Totalvehicles,SUM(CASE WHEN
e.User = 4 THEN 1 ELSE 0 END) as Riders,SUM(CASE WHEN v.Category = 1
THEN 1 ELSE 0 END) as Bycicles,SUM(CASE WHEN v.Category = 2 THEN 1 ELSE
0 END) as Scooter, SUM(CASE WHEN v.Category = 3 THEN 1 ELSE 0 END) as
Quads, SUM(CASE WHEN v.Category = 7 THEN 1 ELSE 0 END) as Cars,
SUM(CASE WHEN v.Category = 13 THEN 1 ELSE 0 END) as Trucks



//-- QUERY  8. Numero di incidenti in cui non sono coinvolti pedoni o
ciclisti (3 o 4) distinguendo tra quelli avvenuti in aree urbane e non,
suddivisi per fasce orarie (diurni, notturni, tramonto o alba)

//-- lighting: several band_time of the day
//-- type user: not pedestrains or riders( not in 3,4)
//-- areas of accident: 1 suburbs, 2 built-up

MATCH (c:Characteristics)
WHERE NOT (c)<-[:INVOLVED_IN5]-(:Pedestrians) AND NOT
```

```
(c)<-[:INVOLVED_IN4]-(:Riders)
WITH c, CASE c.Builtup
  WHEN 1 THEN 'periferia'
  WHEN 2 THEN 'urbana'
END as urban_area,
CASE
  WHEN c.Lighting = 1 THEN 'diurno'
  WHEN c.Lighting = 2 THEN 'tramonto o alba'
  WHEN NOT c.Lighting IN [1, 2] THEN 'notturno'
END as time_day
RETURN urban_area, time_day, COUNT(*) as accidents



//-- QUERY 9. Visualizza il numero di incidenti che hanno coinvolto tre
o pi√π veicoli con feriti in ordine decrescente  -- gravi o mortali
suddivisi nelle varie infrastrutture stradali  -- (tunnel o gallerie 1
, ponti 2 , ferrovie 4, zone pedonali 6)

//-- road infrastructure of accident: (1 tunnel or galleries, 2
bridges, 4 //rialways, 6 walkways)
//-- colllision: 4,5 to take accidents with three or more vehicles
(road //traffic disasters)
//-- severity: 2 user dead, 3 user heavily injured

CALL{
      MATCH (:Drivers)-[d:INVOLVED_IN2]->(c)
      WHERE d.Severity IN [2,3]
      RETURN c
      UNION
      MATCH (:Riders)-[r:INVOLVED_IN4]->(c)
      WHERE r.Severity IN [2,3]
      RETURN c
      UNION
      MATCH (:Passengers)-[s:INVOLVED_IN3]->(c)
      WHERE s.Severity IN [2,3]
      RETURN c
      UNION
      MATCH (:Pedestrians)-[t:INVOLVED_IN5]->(c)
      WHERE t.Severity IN [2,3]
      RETURN c
}
WITH c
MATCH (c)-[:TOOK_PLACE_IN]->(p:Places)
WHERE c.Collision IN [4, 5] AND p.road_infrastructure IN [1,2,4,6]
RETURN p.road_infrastructure as roadInfrastructure, COUNT(*) as
accidents
ORDER BY accidents DESC
```

```
//-- QUERY 10. Visualizza il numero di incidenti notturni o all'alba
-- durante il rientro da feste o eventi (5 o 9)  -- suddivisi per
fascia di et√† di tutte le persone coinvolte (under 30, over 30)
MATCH (c)-[r:INVOLVES2|INVOLVES3|INVOLVES5|INVOLVES4]->(u)
WHERE u.Trip IN [5, 9] AND c.Lighting <> 1
WITH c.Acc as id , COUNT(u.Acc) AS TotUsers, SUM(CASE WHEN c.Year +
2000 - u.Birth < 30  THEN 1 ELSE 0 END) AS Under30, SUM(CASE WHEN
c.Year + 2000 - u.Birth  >= 30  THEN 1 ELSE 0 END) AS Over30
RETURN SUM( CASE WHEN TotUsers = Under30 THEN 1 ELSE 0 END) as
TOTUnder30, SUM( CASE WHEN TotUsers = Over30 THEN 1 ELSE 0 END) as
TOTOver30
```