Cypher and SQL and Js

Farnoosh Koleini

2023-03-18

1. Relational database schema design (10 pts)

creating a table for accidents

CREATE TABLE IF NOT EXISTS public.accidents (“Accident_Index” character varying(255) COLLATE pg_catalog.”default”, “Accident_Severity” character varying(255) COLLATE pg_catalog.”default”)

Query Query History

```
1  SELECT * FROM public.accidents
2
```

Data Output Messages Notifications

	Accident_Index character varying (255)	Accident_Severity character varying (255)
1	Accident_Index	Accident_Severity
2	201501BS70001	3
3	201501BS70002	3
4	201501BS70004	3
5	201501BS70005	3
6	201501BS70008	2
7	201501BS70009	3
8	201501BS70010	3
9	201501BS70011	3
10	201501BS70012	3
11	201501BS70013	3
12	201501BS70014	3
13	201501BS70015	3
14	201501BS70016	3
15	201501BS70017	3
16	201501BS70018	3
17	201501BS70019	3
18	201501BS70020	3
19	201501BS70022	3

Total rows: 1000 of 140057 Query complete 00:00:00.259

creating a table for model

CREATE TABLE IF NOT EXISTS public.model (“Accident_Index” character varying(255) COLLATE pg_catalog.”default”, “Vehicle_Type” character varying(255) COLLATE pg_catalog.”default”)

Query Query History

```
1  SELECT * FROM public.model
2
```

Data Output Messages Notifications

	Accident_Index character varying (255)	Vehicle_Type character varying (255)
1	Accident_Index	Vehicle_Type
2	201501BS70001	19
3	201501BS70002	9
4	201501BS70004	9
5	201501BS70005	9
6	201501BS70008	1
7	201501BS70008	9
8	201501BS70009	3
9	201501BS70009	19
10	201501BS70010	9
11	201501BS70010	1
12	201501BS70011	9
13	201501BS70011	9
14	201501BS70012	9
15	201501BS70012	5
16	201501BS70013	8
17	201501BS70013	8
18	201501BS70014	9

Total rows: 1000 of 257846 Query complete 00:00:00.236

creating a table for vehicle_type

CREATE TABLE IF NOT EXISTS public.vehicle_type (code character varying(255) COLLATE pg_catalog."default", label character varying(255) COLLATE pg_catalog."default")

Query Query History

```
1  SELECT * FROM public.vehicle_type
2
```

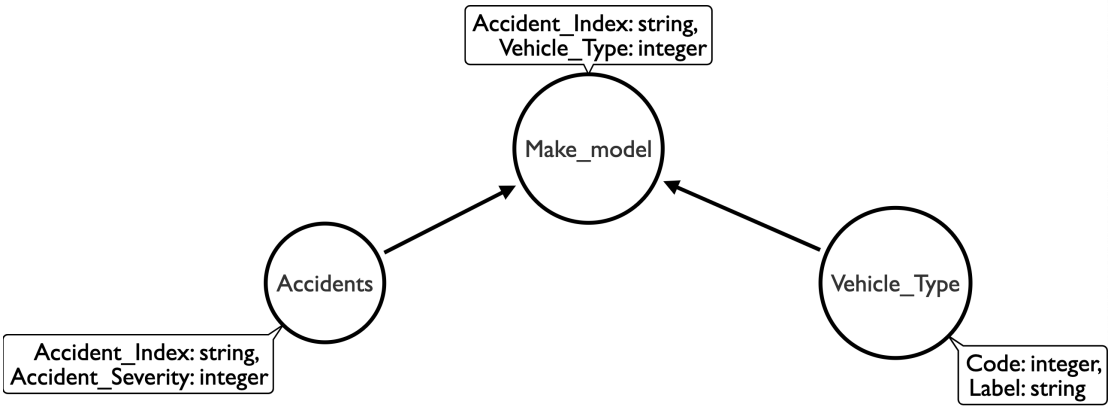
Data Output Messages Notifications

	code character varying (255)	label character varying (255)
1	code	label
2	0	Pedestrian
3	1	Cyclist
4	2	Motorcycle 50cc and under rider or passenger
5	3	Motorcycle 125cc and under rider or passenger
6	4	Motorcycle over 125cc and up to 500cc rider or passenger
7	5	Motorcycle over 500cc rider or passenger
8	8	Taxi/Private hire car occupant
9	9	Car occupant
10	10	Minibus (8 - 16 passenger seats) occupant
11	11	Bus or coach occupant (17 or more pass seats)
12	16	Horse rider
13	17	Agricultural vehicle occupant
14	18	Tram occupant
15	19	Van / Goods vehicle (3.5 tonnes mgw or under) occupant
16	20	Goods vehicle (over 3.5t. and under 7.5t.) occupant
17	21	Goods vehicle (7.5 tonnes mgw and over) occupant

Total rows: 22 of 22 Query complete 00:00:00.125

2. Neo4j schema design and data load (35 pts)

2.1 Schema design (10 pts) Draw the database schema for the graph database. You can use http://www.apcjones.com/ (http://www.apcjones.com/) arrows, or draw it on paper and add a picture of it.



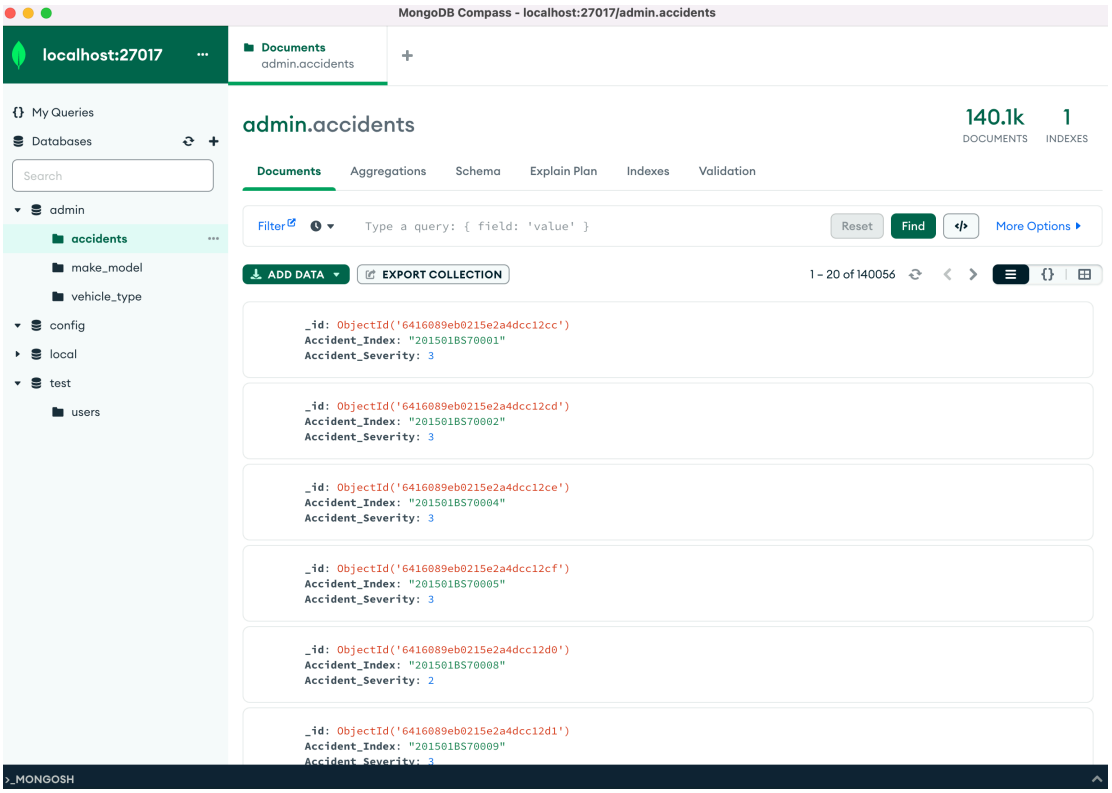
2.2 Data load (25 pts) List the Cypher queries to load the data in the database.

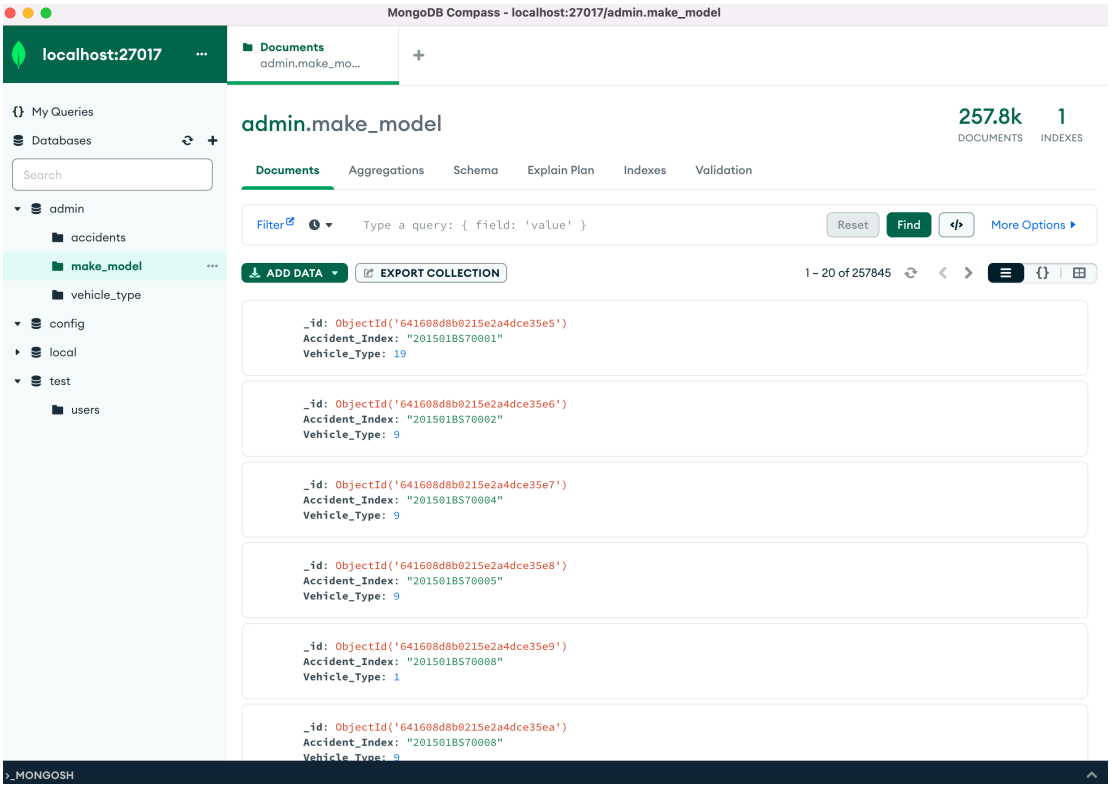
CREATE (:Accidents (‘Accident Index:’‘, ’Accident Severity’: 0}); CREATE (:Make Model { ‘Accident Index’: ’‘, ’Vehicle Type’: 0}); CREATE (:Vehicle_Type {Code: 0, Label:”});

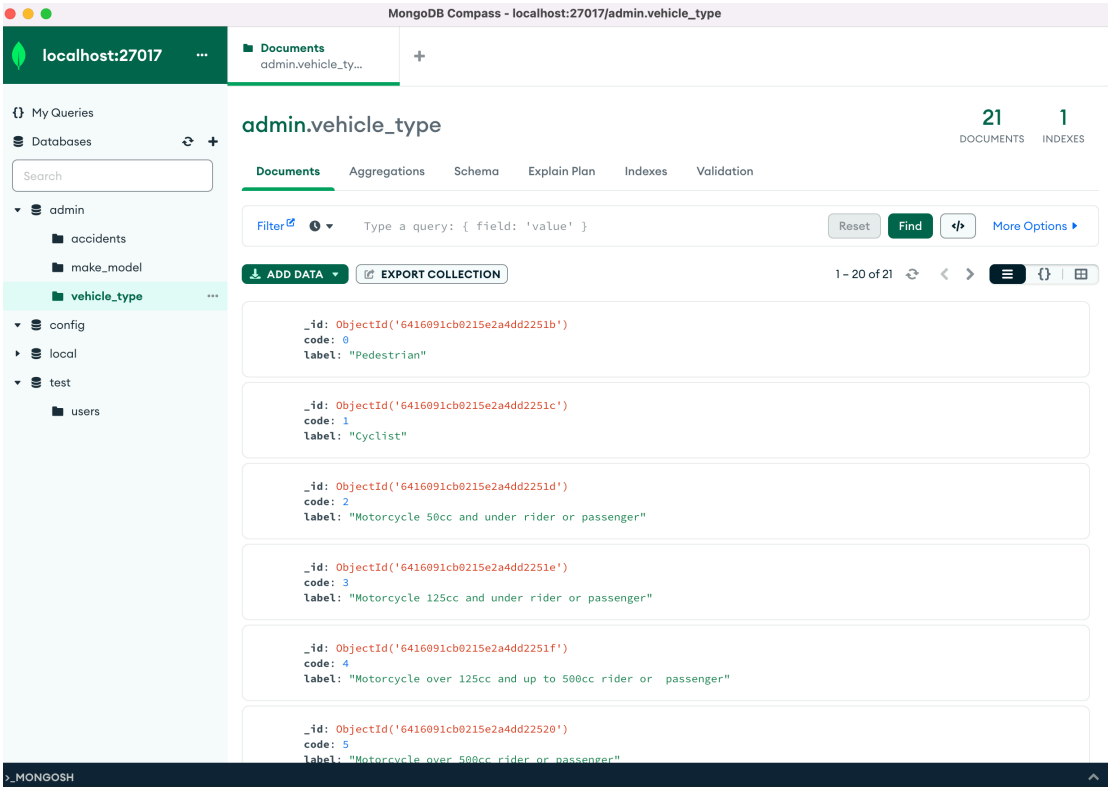
LOAD CSV WITH HEADERS FROM ‘file:///Accidents.csv (file:///Accidents.csv)’ AS row CREATE (:Accidents { ‘Accident Index’: row. ‘Accident Index’ , ‘Accident Severity’: toInteger (row. ‘Accident Severity’) }) LOAD CSV WITH HEADERS FROM file:///Make_Model.csv (file:///Make_Model.csv)’ AS row CREATE (:Make_Model { ‘Accident Index’: row. ‘Accident Index’ , ‘Vehicle Type’: toInteger (row. ‘Vehicle Type’) }) LOAD CSV WITH HEADERS FROM ‘file:///Vehicle_Type.csv (file:///Vehicle_Type.csv)’ AS row MERGE (:Vehicle Type { Code: toInteger (row. code), Label: row.label })

3. MongoDB data load (5 pts)

Follow the instructions from MongoDB to import data with MongoDB Compass. Include a screenshot of the loaded collection(s).


MongoDB Compass - localhost:27017/admin.accidents
localhost:27017
Documents
admin.accidents
My Queries
Databases
Search
admin
accidents
make_model
vehicle_type
config
local
test
users
admin.accidents
140.1k
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Filter
Type a query: { field: 'value' }
Reset
Find
More Options
ADD DATA
EXPORT COLLECTION
1 - 20 of 140056
_id: ObjectId('6416089eb0215e2a4dcc12cc')
Accident_Index: "201501BS70001"
Accident_Severity: 3
_id: ObjectId('6416089eb0215e2a4dcc12cd')
Accident_Index: "201501BS70002"
Accident_Severity: 3
_id: ObjectId('6416089eb0215e2a4dcc12ce')
Accident_Index: "201501BS70004"
Accident_Severity: 3
_id: ObjectId('6416089eb0215e2a4dcc12cf')
Accident_Index: "201501BS70005"
Accident_Severity: 3
_id: ObjectId('6416089eb0215e2a4dcc12d0')
Accident_Index: "201501BS70008"
Accident_Severity: 2
_id: ObjectId('6416089eb0215e2a4dcc12d1')
Accident_Index: "201501BS70009"
Accident_Severity: 3
>_MONGOSH


MongoDB Compass - localhost:27017/admin.make_model
localhost:27017
Documents
admin.make_mo...
My Queries
Databases
Search
admin
accidents
make_model
vehicle_type
config
local
test
users
admin.make_model
257.8k
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Filter
Type a query: { field: 'value' }
Reset
Find
More Options
ADD DATA
EXPORT COLLECTION
1 - 20 of 257845
_id: ObjectId('641608d8b0215e2a4dce35e5')
Accident_Index: "201501BS70001"
Vehicle_Type: 19
_id: ObjectId('641608d8b0215e2a4dce35e6')
Accident_Index: "201501BS70002"
Vehicle_Type: 9
_id: ObjectId('641608d8b0215e2a4dce35e7')
Accident_Index: "201501BS70004"
Vehicle_Type: 9
_id: ObjectId('641608d8b0215e2a4dce35e8')
Accident_Index: "201501BS70005"
Vehicle_Type: 9
_id: ObjectId('641608d8b0215e2a4dce35e9')
Accident_Index: "201501BS70008"
Vehicle_Type: 1
_id: ObjectId('641608d8b0215e2a4dce35ea')
Accident_Index: "201501BS70008"
Vehicle_Type: 9
>_MONGOSH



4. Data analysis (60 pts: 20 pts each per SQL, Cypher, and MongoDB queries)

SQL query

SELECT vehicle_type.code, AVG(accidents.accident_severity) AS average_accident_severity, percentile_cont(0.5) WITHIN GROUP (ORDER BY accidents.accident_severity) AS median_accident_severity FROM accidents JOIN make_model ON accidents.accident_index = make_model.accident_index JOIN vehicle_type ON make_model.vehicle_type = vehicle_type.code WHERE vehicle_type.label ILIKE '%motorcycle%' GROUP BY vehicle_type.code;

Cypher query

MATCH (a:accidents)-[:accident_index]->(mm:make_model)-[:code]->(vt:vehicle_type) WHERE tolower(vt.label) CONTAINS "motorcycle" RETURN vt.label, AVG(a.Accident_Severity) as avg_severity MATCH (mm:make_model)-[:code]->(vt:vehicle_type) WHERE toLower(vt.label) CONTAINS 'motorcycle' WITH vt.label AS label, COLLECT(mm.Accident_Severity) AS severities WITH label, apoc.coll.sort(severities) AS sortedSeverities WITH label, CASE WHEN size(sortedSeverities) % 2 = 0 THEN (sortedSeverities[size(sortedSeverities) / 2] + sortedSeverities[(size(sortedSeverities) / 2) - 1]) / 2.0 ELSE sortedSeverities[(size(sortedSeverities) - 1) / 2] END AS medianSeverity RETURN label, medianSeverity

MongoDB query

Create intermediary collection with initial aggregation results db.createCollection(“intermediary”)

db.accident.aggregate([{ lookup: {from: "model", localField: "accident_index", foreignField: "accident_index", as: "model"}}, { unwind: “model"}, { lookup: {from:” vehicle_type”, localField: “model.vehicle_type”, foreignField: “code”, as: “vehicle_type”}}, { *unwind* :"vehicle_type”}, { match: {"vehicle_type.label": /motorcycle/i}}, { group: {_id: “vehicle_type.code", avg_accident_severity: { avg:” accident_severity"}}}, { out: “intermediary”}])

Calculate median using additional queries on intermediary collection db.intermediary.aggregate([{ group: {_id: " _id”, avg_accident_severity: {*first* :"avg_accident_severity”}, sorted_accident_severity: { *push* :"avg_accident_severity”}}}, { project: {median_accident_severity: { arrayElemAt: [“ sorted_accident_severity", { floor: { divide: [{ size:”$sorted_accident_severity”}, 2]} }] } } }])

this approach requires creating a new collection to store the intermediary results, and may not be the most efficient way to calculate the median in MongoDB. Another option would be to use the bucket aggregation stage to bin the accident_severity values and then calculate the median based on the bin boundaries. Using the bucket aggregation stage in MongoDB, we can group the accident severity values into separate buckets based on their value, and then find the bucket that contains the median value. Here’s the code for that:

db.accident.aggregate([{ $lookup: { from: “model”, localField: “accident_index”, foreignField: “accident_index”, as: “model” } }, { *unwind* :"model” }, { $lookup: { from: “vehicle_type”, localField: “model.vehicle_type”, foreignField: “code”, as: “vehicle_type” } }, { *unwind* :"vehicle_type” }, { $match: { “vehicle_type.label”: /motorcycle/i } }, { group: { _id: "vehicle_type.code”, average_accident_severity: { *avg* :"accident_severity” }, buckets: { *push* :"accident_severity” } } }, { $project: { _id: 1, average_accident_severity: 1, median_accident_severity: { $let: { vars: { count: { *size* :"buckets” }, middle: { $trunc: { $divide: [{ *size* :"buckets” }, 2] } } }, in: { $cond: { if: { $eq: [{ $mod: [{ *size* :"buckets” }, 2] }, 0] }, then: { $avg: [{ *arrayElemAt* : ["buckets”, “$middle”] }, { *arrayElemAt* : ["buckets”, { *subtract* : ["middle”, 1] }] }] }, else: { *arrayElemAt* : ["buckets”, “$middle”] } } } } } } }])

Results:

	code integer	average_accident_severity numeric	median_accident_severity double precision
1	2	2.8265534197586053	3
2	3	2.7807017543859649	3
3	4	2.6904435299497028	3
4	5	2.5849163595123334	3
5	23	2.4444444444444444	2
6	97	2.6945454545454545	3