* Preprocessing (MySQL Workbench):

First, we decided to aggregate all the tables we needed from MySQL Workbench (collisions, victims, parties), by using the script below:

|  |
| --- |
| CREATE TABLE main AS  SELECT   table1.case\_id AS case\_id,  table1.killed\_victims AS killed\_victims,  table1.injured\_victims AS injured\_victims,  table1.primary\_collision\_factor AS primary\_collision\_factor,  table1.pcf\_violation\_category AS pcf\_violation\_category,  table1.pedestrian\_action AS pedestrian\_action,  table1.lighting AS lighting\_condition,  table1.alcohol\_involved AS alcohol\_involved,  table1.collision\_date AS collision\_date,  table2.party\_type AS party\_type,  table2.party\_race AS party\_race,  table2.party\_sex AS party\_sex,  table2.party\_sobriety AS party\_sobriety,  table2.party\_drug\_physical AS party\_drug\_physical,  table2.under\_influence\_ethnicity AS under\_influence\_ethnicity,  table2.vehicle\_year AS vehicle\_year,  table2.vehicle\_make AS vehicle\_make,  table3.victim\_degree\_of\_injury AS victim\_degree\_of\_injury,  table3.victim\_role AS victim\_role,  table3.victim\_sex AS victim\_sex,  table3.victim\_age AS victim\_age FROM  (  SELECT   \*  FROM  collisions  WHERE  (primary\_collision\_factor IS NOT NULL) AND (primary\_collision\_factor != 2) AND (primary\_collision\_factor != 'unknown')  AND (pcf\_violation\_category IS NOT NULL) AND (pcf\_violation\_category != 21804) AND (pcf\_violation\_category != 'unknown')  AND (pedestrian\_action IS NOT NULL)  AND (lighting IS NOT NULL)  ) AS table1    INNER JOIN    (  SELECT   \*  FROM  parties  WHERE  (vehicle\_year > 1929 OR vehicle\_year < 2020)  AND (vehicle\_make IS NOT NULL)  AND (party\_drug\_physical IS NOT NULL)  AND (party\_race IS NOT NULL)  AND (under\_influence\_ethnicity IS NOT NULL)  ) AS table2 ON table1.case\_id = table2.case\_id    INNER JOIN  (  SELECT   \*  FROM  victims  WHERE  (victim\_degree\_of\_injury IS NOT NULL) AND (victim\_degree\_of\_injury != '7') AND (victim\_degree\_of\_injury != '6') AND (victim\_degree\_of\_injury != '5') AND (victim\_degree\_of\_injury != 'M')  AND (victim\_age > -1 OR victim\_age < 100)  AND (victim\_sex = 'male' OR victim\_sex = 'female')   AND (victim\_role != 'l') AND (victim\_role != 'm') AND (victim\_role IS NOT NULL)  ) AS table3 ON table1.case\_id = table3.case\_id |
|  |

We took this chance to also clean our dataset from some incorrect values and mistakes that have been made by police officers when populating the [SWITRS](https://tims.berkeley.edu/help/SWITRS.php) .

Subsequently, we have used the “Table Data Export Wizard” function available in MySQL Workbench to export the .json file.

* Preprocessing (Python):

In this phase, we opened the .json file in a Jupyter Notebook to create a better and more purposeful file that could fulfil the design of a truly non-relational database schema.

|  |
| --- |
| import json with open('./json/main.json') as m:  database = json.load(m) |

We checked again for any unwanted or misleading data, using the script below:

|  |
| --- |
| database = [  x for x in database if   (x['party\_sobriety'] != '')  and ( x['under\_influence\_ethnicity'] != '' )  and ( x['party\_drug\_physical'] != '' )  and ( x['vehicle\_make'] != '' )  and ( x['party\_race'] != '' )  and ( x['vehicle\_year'] > 1929 or x['vehicle\_year'] < 2020 )  and ( x['victim\_role'] != '' )   and ( x['victim\_role'] != 'm' )   and ( x['victim\_role'] != 'l' )  and ( (x['victim\_sex'] == 'male') or (x['victim\_sex'] == 'female') )  and ( x['victim\_age'] > -1 or x['victim\_age'] < 100 )  and ( x['victim\_degree\_of\_injury'] != '6' )  and ( x['victim\_degree\_of\_injury'] != '7' )  and ( x['victim\_degree\_of\_injury'] != '5' )  and ( x['victim\_degree\_of\_injury'] != 'M' )  and ( x['victim\_degree\_of\_injury'] != '')  and ( x['primary\_collision\_factor'] != '' )   and ( x['primary\_collision\_factor'] != 2 )   and ( x['primary\_collision\_factor'] != 'unknown' )  and ( x['pcf\_violation\_category'] != '' )  and ( x['pcf\_violation\_category'] != 21804 )  and ( x['pcf\_violation\_category'] != 'unknown' )  and ( x['pedestrian\_action'] != '')  and ( x['lighting\_condition'] != '') ] |

We set the case\_id field as our main identifier in order to combine/group the information more effectively by using the script below:

|  |
| --- |
| for item in database:  item['case\_id'] = int(item['case\_id'])  if item['under\_influence\_ethnicity'] == '0':  item['under\_influence\_ethnicity'] = False  else:  item['under\_influence\_ethnicity'] = True  if item['alcohol\_involved'] == 0:  item['alcohol\_involved'] = False  else:  item['alcohol\_involved'] = True   guids = set([x['case\_id'] for x in database]) result = [{'case\_id': x} for x in guids] |

For this reason, we inserted all the fields with more than one value per case\_id in arrays (see below):

|  |
| --- |
| counter = len(result)  for i in result:  case\_id = i['case\_id']    counter -= 1  if counter % 1000 == 0:  print(counter)    for j in database:  if j['case\_id'] == case\_id:  if 'killed\_victims' not in i:  i['killed\_victims'] = j['killed\_victims']  i['injured\_victims'] = j['injured\_victims']  i['primary\_collision\_factor'] = j['primary\_collision\_factor']  i['pcf\_violation\_category'] = j['pcf\_violation\_category']  i['pedestrian\_action'] = j['pedestrian\_action']  i['lighting\_condition'] = j['lighting\_condition']  i['alcohol\_involved'] = j['alcohol\_involved']  i['collision\_date'] = j['collision\_date']  i['party\_type'] = [j['party\_type']]  i['party\_race'] = [j['party\_race']]  i['party\_sex'] = [j['party\_sex']]  i['party\_sobriety'] = [j['party\_sobriety']]  i['party\_drug\_physical'] = [j['party\_drug\_physical']]  i['vehicle\_year'] = [j['vehicle\_year']]  i['vehicle\_make'] = [j['vehicle\_make']]  i['victim\_degree\_of\_injury'] = [j['victim\_degree\_of\_injury']]  i['victim\_role'] = [j['victim\_role']]  i['victim\_sex'] = [j['victim\_sex']]  i['victim\_age'] = [j['victim\_age']]  else:  i['party\_type'].append(j['party\_type'])  i['party\_race'].append(j['party\_race'])  i['party\_sex'].append(j['party\_sex'])  i['party\_sobriety'].append(j['party\_sobriety'])  i['party\_drug\_physical'].append(j['party\_drug\_physical'])  i['vehicle\_year'].append(j['vehicle\_year'])  i['vehicle\_make'].append(j['vehicle\_make'])  i['victim\_degree\_of\_injury'].append(j['victim\_degree\_of\_injury'])  i['victim\_role'].append(j['victim\_role'])  i['victim\_sex'].append(j['victim\_sex'])  i['victim\_age'].append(j['victim\_age']) |

At this point, we were ready to dump the .json file using the following code...

|  |
| --- |
| with open('./json/opt\_main.json', 'w') as m:  json.dump(result, m) |

...and import it into our MongoDB environment, by using this Windows PowerShell command (after having set the MongoDB environment path as global):

|  |
| --- |
| PS C:\Users\giogi> mongoimport --db traffic --collection california --drop --jsonArray --batchSize 1 --file opt\_main.json |

Our final preprocessing step was to convert all the date fields to the ISODate format, so as to be able to easily use conditional statements over dates in our queries. Here is the function we used to perform, and save permanently, such modifications.

|  |
| --- |
| db.california.find().forEach( function(doc) {  doc['collision\_date'] = new Date(doc['collision\_date']);  db.california.save(doc); }) |