Evreka Back-End Developer Evaluation Questions

Abdullah Taha

GitHub repository: https://github.com/abdullah-taha/Evreka-Evaluation-Questions

Question1:

1.Function to get a list of last points per vehicle for the last 48 hour:

For this question I focused on the point that the NavigationRecord table is very big and consuming in term of performance. For this, I did the followings:

- I used the filter method so the filtering process would happen in the database server. Because the database servers are always much faster computers and it is always a good idea to put the work as much as possible to the server rather than getting all the data and filtering it on python side.
- I used the select_related() method so it would join the vehicle table in the same database hit without requiring a second hit. (if I had accessed the vehicle plate in a separate line it would have made another database hit)
- When building the models, I put an index on the datetime column (db.index=True) to make the filtering speed on that column faster.

With those two points in mind, the function would make only one hit to the database joining the required table, filtering and getting the required data in an efficient way.

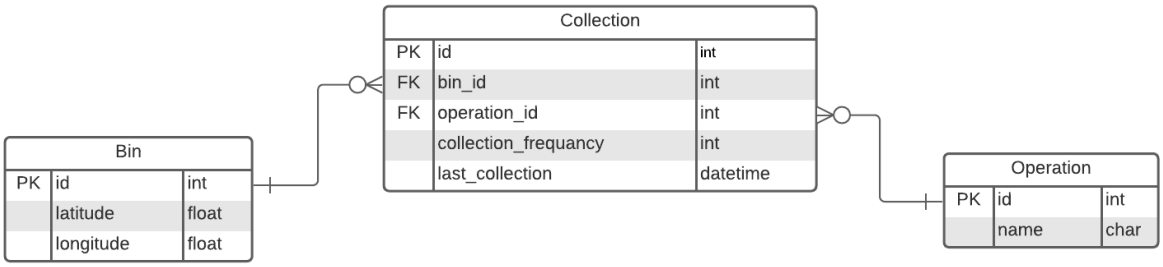
note: I used Django debug tool to investigate some query performances.

2.Suggestion for getting the data in a more efficient way.

- First things I would suggest is to index the datetime columns if it is not already indexed.
- Other suggestion is to cache the results or use a temporary table but that totally depends on the frequency of data insertion and the frequency of querying.
- Another thing I could suggest is to make a caching table that would holds the data for the last 48 hours. When new data arrives, it would be inserted to both tables. We can then fetch the required data in a much faster way by querying only the small table. We would also schedule a daily job on the database to delete all the old data.

Question2:

1. Database model design:



Here we notice a many-to-many relationship between the Bin and the Operation entities. Every bin has multiple operations, and every operation is applied on many bins. We make a third table representing the Collection entity which will have a bin-operation information per record. This table will also hold the collection_frequency and last_collection information. Beforehand when there was only one operation, the collection_frequency and last_collection attributes were attributes that described the Bin entity. But when different operations came into the design. those two attributes are no longer describing the bin itself. they are now describing the Collection entity.

2. Function to get a list of collection_frequency values for all bin-operation pairs:

Here, because we need the information of only the collection_frequency column, I used the value_list method to get only the required column's values without the need to build a python object of the whole table.