Pistes à explorer pour une optimisation future

Le code pour l'extraction consomme beaucoup de mémoire RAM. Cela est dû à la quantité de donnée présente dans les bases de données.

Le code dans opt_extraction.py est plus optimisé que celui de extraction.py.

Pistes à explorer pour optimiser l'extraction et la sauvegarde des données au format ndjson provenant de bases de données nécessitant des jointures.

Points de vigilances :

ATTENTION → On pourrait parcourir le dataframe par lots et supprimer les lots au fur et à mesure qu'ils sont sauvegardés mais cela est risqué car cela implique de modifier le dataframe alors même qu'on itère dessus. Cela n'est pas une bonne pratique et pourrait occasionner des bugs.

Alternatives :

- Group BY sur le timestamp dans la query SQL et LIMIT i ; i + batch_size

Dans la fonction save_with_join() et dans la façon d'implémenter les codes pour joindre les bases de données l'optimisation la plus simple serait de mettre en place une boucle qui run n fois la query sql :

SELECT * FROM table GROUP BY timestamp LIMIT i, i + batch_size

Cela permettrait de générer de plus petit dataframe et de les traiter les uns après les autres.