Anpassung der Daten, um sie nutzbar zu machen. Es wurde dafür ein Python Skript geschrieben, welches die Yelp Daten mit den Inspektionsdaten verknüpft hat. Dafür wurden allen eindeutig zuordenbaren Restaurants in dem Inspektionsdatensatz eines der meistbewerteten Businesses aus Yelp mit der Kategorie Restaurant zugeordnet und in einer mapping Tabelle dargestellt.

fakingScript.py

Danach müssen die Daten bereinigt werden. Lokale ohne eindeutigen INSPECTION DATE (hier 01/01/1900) müssen entfernt werden, sowie die den zugehörigen

Database-Name: PortfolioPruefung

Yelp: yelp\_business

Inspektionen: inspection\_results

1. fakingScript (ändert yelp daten und verbindet yelp und inspections)
2. consistencyCheck (prüft ob alle Datensätze dieselben Datenfelder hat)
3. consistencyEnforcement (fügt fehlende Datensätze hinzu)
4. removeMissingGeoData (löscht die in consistencyEnforcement erstellten Geodaten mit Null)
5. deleteRestaurantsWithoutInspections (Restaurants ohne Inspection werden gelöscht)
6. dataTypesCheck (checkt aus einheitlichen Datentypen -> BUILDING inkonsistent, aber irrelevant, da wir es nicht in SQL übernehmen + PHONE inkonsistent )
7. searchDuplicates (sucht nach Duplikate -> findet keine)
8. weitere Skripte notwendig: daten ohne geodaten löschen, consistency check für andere datentypen

121 daten ohne geodaten sind statistisch irelevant -> gelöscht

Zwei SQL-Tabellen benötigt

1. Yelp Daten
2. Inspection results

Yelp Daten: Object ID = Primärschlüssel

Business Id, Name, Address, Postal Code, Borough, latitude, longitude, Phone, Stars, Reviews, Cuisine Description

Inspection results: Inspection (Object) ID = Primärschlüssel, KAMI (Business ID) = Fremdschlüssel

Inspection Date, Action, Violation Code, Violation Description, Critical Flag, Score, Grade, Inspection Type, Community Board, Council District, Census Tract, BIN, BBL, NTA

Daniel: SQL-Abfragen in MySQL

Here’s a brief explanation of the improvements and their impact on performance:

1. **Indexing (CAMIS and business\_id)**: Creating indexes on frequently queried fields speeds up search operations by allowing MongoDB to quickly locate relevant documents instead of scanning the entire collection. This reduces query time significantly.
2. **Cursor Iteration**: Instead of converting query results into a list (which can be slow for large datasets), using a cursor allows documents to be processed one at a time, reducing memory usage and improving efficiency.
3. **Batch Inserts (insert\_many)**: Inserting documents in batches (instead of one at a time) minimizes the number of database operations, reducing overhead and speeding up the process of adding large amounts of data.

Together, these optimizations improve query speed, reduce memory consumption, and enhance overall performance when handling large datasets.

Niklas: Präsentation

Silas: Python-Skripte