Het proces en de obstakels van het maken van een database

Onze opdracht was het omzetten van ‘Document Store’ naar een ‘Relationele Database’. Voor deze opdracht kregen we data van de opisopvoordeelshop website in handen. Drie datasets gevuld met de producten, sessies en bezoekers (geanonimiseerd). Deze bestanden werden omgezet naar .json (als ze dat al niet stonden) en door middel van de command prompt in de database gezet. Wij hebben gebruik gemaakt van de PostgresSQL, Flask, Psycopg2, Pymongo, CSV, Pandas, Numpymodules.

Met de bestanden in de database konden we eindelijk onze eerste blik erop werpen en aan de slag gaan met de formatieve opdracht 2a. Door hier queries voor op te zetten merkten we al gauw dat er een aantal dingen mis waren, zoals onder andere producten waar de prijs 0 van was, gegevens waar dingen van misten en ga zo maar door. We wisten nog niet dat we met de gegevens aan moesten. De oplossingen zouden later in het proces blijken.

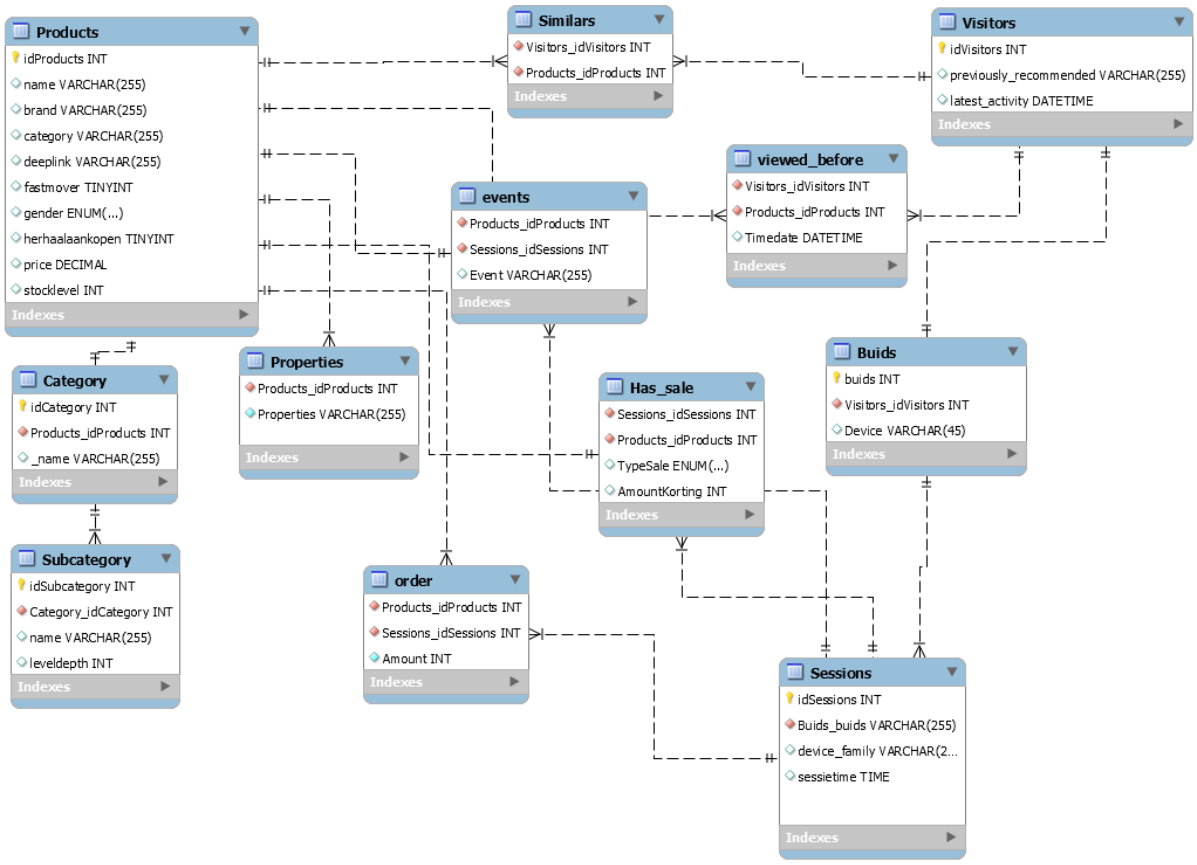
Hierna hebben wij besloten om een begin te maken aan de formatieve opdrachten. Hierdoor hebben wij een beter beeld gekregen van de dataset en wat de beste oplossingen zijn om met de dataset om te gaan. Als eerste stap hadden we een kladversie bestandje aangemaakt om alle mogelijk bruikbare gegevens in op te schrijven. We hebben rechtstreeks in de MongoDB gekeken en de gegevens hebben we vooral gekozen vanwege de mogelijkheden, zowel overduidelijk als theoretisch, die we erin zagen. Daarnaast gingen we ook kijken naar manieren om de data vanuit MongoDB naar PostgreSQL te zetten. De beste manier die we vonden was om de data eerst in een .csv (comma-separated values) bestand te zetten, waarna het makkelijker zou zijn om de data in sql queries om te zetten. Op dit punt hebben we de groep tijdelijk in tweeën gesplitst. De ene groep zou aan de slag gaan met csv, de anderen zouden de database in elkaar gaan zetten aan de hand van een entity-relation diagram.

Het entity-relation diagram (afgekort ERD) is onderverdeeld in drie hoofdtabellen, sub tabellen (zoals properties), die grote data van de hoofdtabellen in kleinere stukken opdeelt, en meerdere koppeltabellen. het gebruik van de koppeltabellen is om makkelijker data op te halen en op te slaan voor onze recommendation engine. Daarnaast is in deze tabel makkelijk te zien wat voor type relaties qua aantallen bestaan. de relaties in dit diagram zijn 1 (gesignificeerd door de twee streepjes aan het uiteinde) op 1, of 1 op ‘1 of meer’ (een streepje en een kraaienpoot).

Wij hebben alle mogelijke kolommen sterk onderzocht en selectie gemaakt van de meest interessante kolommen. Deze kolommen zijn in onze beleving van belang voor een werkende aanbevelingsmachine. De tabellen “Products”, “Sessions” en “visitors” zijn het meest belangrijk. Onze aandacht is voornamelijk uitgegaan naar deze tabellen. Waarna deze tabellen succesvol zijn ingeladen en opgevuld zijn wij overgegaan tot het opzetten van de andere tabellen.

Voor iedere tabel wordt een kolom aangemaakt die alle identificatienummers van alle rijen bevat. Deze nummers zijn allemaal uniek van elkaar. De dataset is vastgezet in de encoding “UTF-8”. Gedurende het onderzoeken van de inhoud is gebleken dat de dataset velen oneffenheden bevat. Voor dit probleem hebben wij verschillende functies gemaakt die stapsgewijs alle problemen verhelpen.

Sam van Roosmalen
Stan Haakman
Youssef Boulfiham
Laurens Lancel
Peter Brouwer
Luuk Bouwman



Zoals eerder vermeld is de datset vooraf overgezet naar verschillende csv-bestanden. Dit hebben wij mogelijk gemaakt met behulp van Python classes die bestaan uit twee primaire functies. De class is makkelijk aan te roepen, leesbaar en flexibel aanpasbaar. In eerste instantie wordt bepaald welke collectie moet worden ingelezen uit de MongoDB en welke dictionaire-waardes gewenst zijn. Deze wordt vervolgens middels een slimme recursie-algoritme uitgelezen uit de database. Gedurende het proces bleek dat de dictionaires zelf ook dictionaires bevat. Hier hebben wij gelukkig ook een oplossing voor gevonden. Wanneer de gewenste waardes zijn ingelezen en vastgehouden, opent het Python script de module CSV. Middels deze module word een csv-bestand en gemaakt en wordt er per regel genoteerd welke rijwaardes zijn opgehaald. Dit proces hebben wij toegepast voor alle gewenste tabellen.

Nadat we alles door hadden, besloten we de foutieve gegevens aangepast en verbeterd in de sql queries in python. De efficiëntie zou hierdoor iets minder goed kunnen worden, maar de groep besloot dat, in ieder geval voor onszelf, we het programma liever makkelijk wilde houden dan het mogelijk 1 seconde (zo niet minder) sneller laten werken.

Zo begonnen we de code te schrijven, vooral in de bestanden _sql_aanmaken.py en huwebshop.sql. Deze huwebshop.sql is gemaakt met behulp van mysql workbench en de ERD. Al gauw liepen we tegen wat problemen aan, de eerste meest voornamelijk waren dat een comment in sql werd geïnterpreteerd als uitvoerbare code in python en een enter in sql werd gezien als \n. Dit hebben we uiteindelijk weten te omzeilen door in python een extra functie te maken die deze tekst filtert en ervoor zorgt dat als het ware de code juist zou worden geïnterpreteerd. Een ander probleem was dat python de sql code als losse regels pakte en dus onlogisch werd gelezen. Als voorbeeld, het blok voor het aanmaken van ‘products’ begint met “CREATE TABLE IF NOT EXISTS Products (“ gevolgd door alles dat in de tabel komt. Python las dit echter als een op zichzelf staande regel. Dit hebben we opgelost

door alle code eerst samen te voegen, dit opsplitsen op de puntkomma's en daarna de puntkomma's weer toevoegen, aangezien deze bij het splitsen verloren gingen. Dit resulteerde in dat de sql nu goed gelezen kon worden.

We konden eindelijk beginnen met testen, en daar kwamen meteen al problemen naar voren. Een van de problemen was dat de 'drop table' niet werkte omdat deze tabel blijkbaar nog actief was. Dit konden we gelukkig simpel oplossen door overal goed de connectie te sluiten. Een ander probleem die we wel zagen aankomen was een mogelijke crash als de tabel niet bestond, en inderdaad, dit gebeurde zoals verwacht. Ook dit is redelijk makkelijk opgelost met een try en except. Wordt de tabel gevonden, dan wordt die verwijdert, zo niet dan slaat het programma deze stap over. De reden dat we de tabellen weg wilde kunnen gooien was zodat aanpassingen in de huwebshop.sql makkelijk uitvoerbaar waren zonder extra te moeten rotzooien binnen het python script. Nadat de tabellen dan al wel of niet zijn verwijderd, wordt alles aangemaakt en opgevuld met de gewenste gegevens

Pseudocode

1. Modules en programma's installeren: pymongo, MongoDB, MongoCompass, CSV, Pandas, Numpy, SQL, PostgreSQL, PgAdmin4, Flask
2. Dataset van huwebshop downloaden
3. Dataset middels cmd overzetten naar MongoDB
4. entity-relation diagram uitschrijven

1. Python script maken bestaande uit alle classes en functies om volledige proces aan te sturen.
2. Middels Python verbinding maken met localhost van MongoDB
3. Verbinding maken met gewenste database
4. Verbinding maken met gewenste collecties. Dit zijn products, visitors en sessions
5. Gewenste data uit dataset lezen
6. Data overschrijven naar csv
7. Data bewerken, opschonen, masseren en controleren.
8. Duplicaten uniek maken, nan omzetten naar bruikbare waarde, geschikt maken voor gewenste kolom-datatype.
9. Verbinding maken met PostgreSQL
10. Controleer of gewenste database bestaat, zowel verwijder database/ database legen is ook mogelijk
11. Creëer nieuwe database
12. Controleer of gewenste tabellen bestaan, zowel verwijder of leeg tabellen
13. Creëer nieuwe tabellen
14. Controleer of gewenste kolomnamen bestaan, zowel verwijder of leeg kolommen.
15. Creëer kolommen en bepaal kolomnamen met gewenste eigenschappen.
16. Vul database op met dataset uit csv-bestanden.
17. Controleer of alle processen geslaagd zijn en of er fouten zijn gemaakt.
18. Documenteer alle bevindingen en belangrijke opmerkingen.
19. Archiveer en back-up