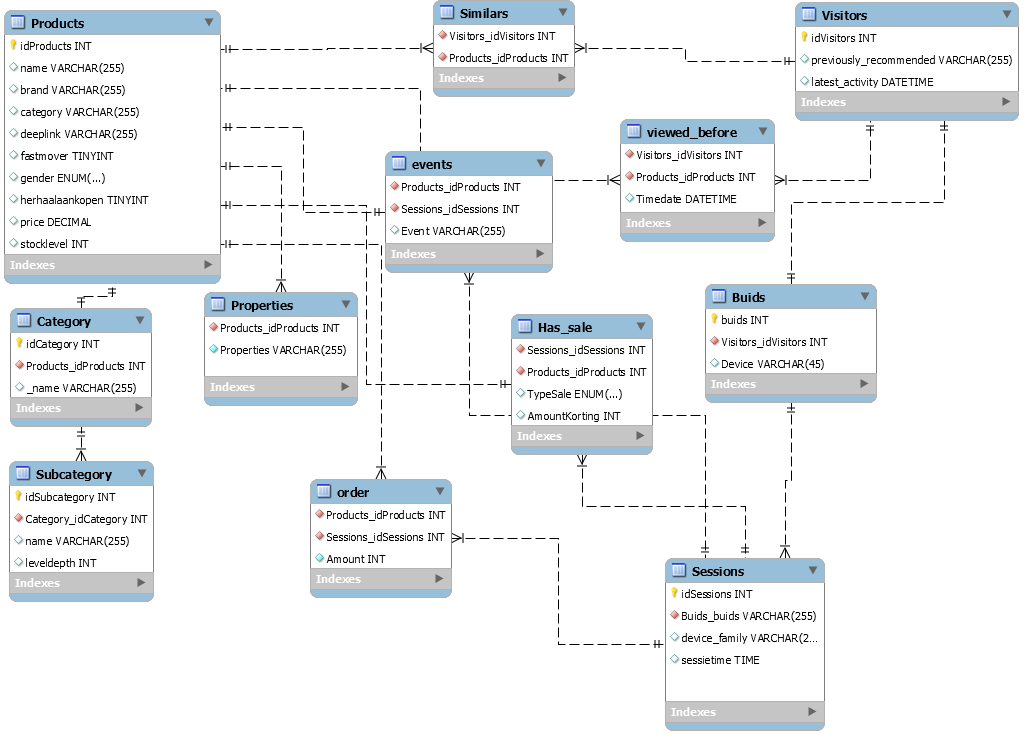
Het proces en de problemen van het maken van een database

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. De reden dat we niet vanuit het mongo compass de bestanden hebben geüpload is omdat dit het proces enorm zou vertragen. Daarnaast hebben we onder andere ook installaties gedaan van onder andere PostgreSQL en flask voor de eigenlijke webshop.

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 sowieso dat we deze gegevens eruit moesten gaan gooien, maar de oplossing daarvoor zouden we later nog bekijken.

Hierna besloten we de andere formatieve opdrachten over te slaan, voornamelijk door het gevoel dat deze opdrachten niet veel extra zouden opleveren voor de uiteindelijke summatieve opdracht, en zodat we meer onze aandacht konden focussen op het maken van de database. Als eerste stap hadden we een .txt 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 enige 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, subtabellen (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).



Ondertussen had de andere groep csv uitgevogeld ……

* ***Info over csv nodig***

Nadat we alles door hadden, besloten we de foutieve gegevens te droppen 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 voornamelijke 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 NODIG!!!!!***
* ***wellicht nog redenen waarom we hebben gekozen voor oplossingen***