Extra opgaver til ETL

Det her dokument indeholder lidt forskellige ting I kan se på, hvis I er ved at være færdige med ETL-opgaven. I har muligvis gjort nogle at tingene allerede, så brug det som inspiration og lav det I synes er mest spændende/relevant.

Små optimeringer

Her er lidt forskellige tips I kan prøve, som måske kan gøre jeres kode lidt mere overskuelig:

- Brug *match* statements i stedet for længere rækker af *if/else*. Se w3schools og pythons dokumentation.
- Csv.DictReader til at læse csv-filer. Se python docs
- Separér jeres credentials (user og password til database fx) ud fra jeres kode. Det kan fx være i en json fil, som i loader i jeres program. Json filen kan så tilføjes jeres gitignore, så den ikke ligger i et public git repo. Alternativ kan I bede brugeren indtaste det i command-line.

Fejlhåndtering med exceptions

De fleste af jer bruger allerede exceptions, men der er alligevel et par ting I kan overveje:

- Kan I finde på edgecases, hvor jeres program crasher? Hvor stort et problem er en given edgecase og hvordan ville I håndtere den?
- Kan I strukturere jeres fejlhåndtering bedre? Kan I finde en mere modulær tilgang til fejlhåndtering, så fejl bliver håndteret færre steder i jeres kode, men stadig håndterer alle de samme fejl?
- Definér jeres egne exception-classes. Ved at lave en class der arver fra Exception eller en af dens subclasses, kan man definere sine egne exceptions. Det kan nogen gange gøre det nemmere at forstå eventuelle fejl man får. Og det kan hjælpe med ovenstående struktur. Se evt design-patterns længere nede for mere om nedarvning (inheritance).

Tests

For at sikre at ens kode opfører sig, som man forventer, vil man ofte skrive tests der kører de funktioner man har defineret og sammenligner outputtet med et forventet resultat.

Python har et built-in testing framework, som bare hedder unittest.

- Se også listen af assert funktioner, som er funktioner der bruges til at sammenligne to ting i konteksten af test.
- Her er en intro til unittest https://www.geeksforgeeks.org/unit-testing-python-unittest/

Ting man kan teste er:

- Transformationerne.
  - Bliver de rigtige værdier transformeret som de skal?
  - Bliver de forkerte værdier håndteret korrekt?
- Extraction.
  - Bliver data hentet korrekt?
  - Håndterer systemet internet-problemer på en korrekt måde?

        - Er der timeout?
        - Stopper programmet hvis det ikke kan komme videre, eller forsøger det at arbejde videre?
    - Hvordan bliver systemet påvirket, hvis extraction sker fra et forkert sted?
- Load.
    - Bliver korrekt data indsat korrekt?
    - Bliver forkert data håndteret ordentligt?

Design-patterns

Hvis man vil arbejde lidt mere med overordnet kodestruktur, kan man se på nogle af de designmønstre man bruger i softwareudvikling. I har måske allerede brugt nogle af dem. De her mønstre bruges ofte i konteksten af Object-Orienteret-Programmering.

Overvej om I kan bruge nogle af dem til at:

- reducere kodegentagelse (kaldes ofte DRY-princippet).
- hjælpe med at løse nogen af de ovenstående ting med exceptions og tests.
- gøre jeres kode mere ekspressiv. Med det menes, at man kan forstå kodestrukturen ud fra navne på klasser, funktioner og variable, uden at skulle referere så meget til kommentarer eller et andet dokument.

Her er et par af dem, der virker mest relevante i forhold til ETL-opgaven:

- Inheritance
- Dependency Injection. Se også CodeAesthetic på youtube.
    - Kan være specielt relevant i forhold til at skrive tests, da det gør det nemmere at teste individuelle dele, ved at injecte mock objects.