Rodrigo Anciães Patelli

Projeto de ETL em Ocaml

URL DO REPOSITÓRIO: https://github.com/RodrigoAnciaes/ETL_Ocaml.git

RESUMO

O objetivo deste projeto é realizar o processamento de dados de arquivos csv contendo dados de pedidos e itens relacionados aos pedidos correspondentes, através do uso de linguagens de programação incluídas no paradigma de programação funcional, realizando filtragem mapeamento e redução de dados, para obter medidas como as taxas totais aplicadas a cada pedido.

Como objetivos do projeto, as funções impuras foram separadas das funções puras das impuras criadas em diversos arquivos e o projeto e seus arquivos organizados usando o gerenciador de projetos DUNE e os dados são organizados em forma de records durante a execução do código.

Nota sobre pureza das funções por arquivo:

Arquivos completamente compostos por funções puras

lib/Process_data.ml

Arquivos que contém funções impuras

lib/Read_data.ml e lib/Save_data.ml fazem leitura e escrita de arquivos.

Bin/main.ml recebe argumentos do usuário, faz uso de prints e chama as funções de leitura e escrita de arquivos

Requisitos do projeto:

1- Projeto ser feito em OCaml.
2- Para calcular a saída, é necessário utilizar map , reduce e filter
3- O código deve conter funções para leitura e escrita de arquivos CSV.
4- Separar as funções impuras das funções puras nos arquivos do projeto.
5- A entrada precisa ser carregada em uma estrutura de lista de Records
6- É obrigatório o uso de Records. Helper Functions para carregar os campos em um Record.

Requisitos Opcionais **IMPLEMENTADOS:**

1 - Salvar os dados de saída em um Banco de Dados SQLite.

2 - Organizar o projeto ETL utilizando dune.

3 - Documentar todas as funções geradas via formato docstring.

4- Ler os dados de entrada em um arquivo estático na internet (exposto via http).

Requisitos Opcionais **Parcialmente i**mplementados:

1- Gerar arquivos de testes completos para as funções puras (existem testes em uma Branch separada do repositório chamada de testes que podem ser executados através do dune test, porém eles não foram revisados e foram em boa parte gerados pela IA Claude, então optei por não os adicionar no projeto principal.)

Ferramentas utilizadas

Docker + devcontainer: https://code.visualstudio.com/docs/devcontainers/containers

Para o desenvolvimento deste projeto foi utilizado um devcontainer contendo dependências básicas para a linguagem Ocaml, como seu compilador e o Opam. Este container foi desenvolvido prréviamente pelo professor da disciplina Raul Ikeda.

O devcontainer agiliza o processo de desenvolvimento pela sua fácil integração com VS-Code, permimitindo que este seja aberto com facilidade no ambiente do container.

Opam: https://opam.ocaml.org/

Opam é um gerenciador de pacotes para Ocaml e é essencial para intalação e uso das outras bibliotecas e ferramehtas do projeto.

Dune: https://dune.readthedocs.io/en/stable/

Dune é um sistema de build para projetos Ocaml, ele facilita todo o processo de desenvolvimento, permitindo a fácil criação de executável, testes e documentação.

Bibliotecas auxiliares usadas pelo projeto (devem ser instaladas antes de sua execução):

- Sqlite3

- Csv

- Ounit2

- Odoc

- Ocurl (binding para Curl)

USO DO PROJETO:

Os passos para a execução do projeto estão mais bem detalhados no README do projeto, porém aqui está um pequeno resumo:

Dentro da pasta do projeto Dune, assegure que esse está com a build realizada correntamente com o comando:

dune build

E então execute o projeto:

dune exec ETL_project -- [OPTIONS]

Exemplos de commandos:

dune exec ETL_project -- --status "Pending" --origin "P"

dune exec ETL_project --input-order "custom_order.csv" --input-order-item "custom_items.csv" --output-csv "results_2025.csv"

Também é possível gerar a documentação do código do projeto atavés de:

Dune build @doc

Desenvolvimento do projeto:

Antes de mencionar o desenvolvimento e os arquivos gerados, é necessário comentar que para cada um dos arquivos do projeto, exceto main.ml e test.ml, foram criados arquivos .mli que tem a mesma função de arquivos header em C, mapeando as funções do arquivo e permitindo seu uso por outras partes do projeto.

Criando ambiente:

A primeira fase do projeto é a inicialização do ambiente, adicionando o container de desenvolvimento no projeto e o abrindo/executando pela primeira vez. Ao fazer isso ele instalará o container, as dependências desejadas e executará os comandos de criação necessários. Não sei o motivo, mas as vezes os comandos de criação necessários não são executados corretamente, caso isso ocorra, execute os comandos do arquivo. devcontainer/ postCreate.sh manualmente após o fim do processo de inicialização do container.

Após isso, para começar um projeto dune do zero, execute dune init. Este comando gerará a infraestrutura básica para o projeto dune. Se já não tiver é necessário criar também uma pasta /data, onde devem ser colocados os arquivos para processamento.

Ao iniciar o projeto o dune também cria automaticamente um arquivo bin/main.ml e diversos arquivos de configuração dune onde são declarads informações como nomes de bibliotecas, dependências necessárias e informações necessárias para a criação automática de documentação.

Iniciando o código:

A primeira etapa deste projeto foi criar as funções resposáveis pela leitura dos arquivos csv e criação dos records relacionados. Como funções de leitura de arquivos em Ocaml são impuras, elas foram separadas em um arquivo lib/Read_data.ml. As funções utilizam da biblioteca csv para extrair os dados e transformalos em um array que pode ser processado em um record definido

A segunda etapa do projeto é o processamento em sí, este que apesar de ser formados por funções puras também foi separado em um arquivo separado lib/Process_data.ml, pensando em uma estrutura que para a execução final do código todas as funções serão chamadas por um arquivo principal bin/main.ml.

Para o processamento foram criadas três funções principais:

origin_status_filter:

Resposável em filtrar o record recebido por Read_data.ml de acordo com a origem e estatus pedida pelo usuário. Está função recebe, além dos records a serem filtrados, uma função transformer que aplicará o restante das transformações do processamento.

transform_to_result:

Esta é a função transformer recebida pela função origin_status_filter, o objetivo desta função é aplicar as transformações pedidas ao record, como calcular a taxa total de cada pedido e salvar todos os resultados em um único record que será enviado posteriormente a uma função que o salvará em csv ou sql, como as funções necessárias para o salvamento do arquivo são impuras, elas foram separadas em outro arquivo lib/Save_data.ml.

A função transform_to_result possui quatro partes principais, sendo elas:

- Filtrar quais são as pedidos que estão em ambos os conjuntos de dados de input (orders e order_items). Esta operação não só é importante para garantir a consistência dos dados após a passagem do filtro (origin_status_filter), mas também para garantir que não ocorram casos em que um dos datasets está mais completo ou incompleto que outro.

- Calcular as medidas de resumo para cada item (total_amount e total_tax).

- Agrupar as medidas calculadas pelo order_id.

- Fazer a junção e cálculo final dos resultados.

transform_to_result_with_filter:

Esta última função gfaz a aplicação parcial da função origin_status_filter, ao passar transform_to_result como argumento, possibilitando o caso de uso que o usuário passa apenas quais valores ele deseja filtrar para a função e ela já retorna os resultados completos.

Após a leitura e processamento dos dados, a próxima etapa foi criar funções capaz de salvar os resultados tanto em formato csv tanto em um baco de dados sqlite, para isso, no arquivo lib/Save_data.ml foram criadas quatro funções principais:

add_header_to_csv:

Esta função tem como objetivo de criar o arquivo adicionar os headers ao arquivo csv.

helper:

Esta função tem como objetivo realizar a operação de salvar o record dos resultados no arquivo

save_to_csv:

Esta função faz a aplicação parcial das funções add_header_to_csv e helper para simplificar e ocultar esta parte do usuário.

save_to_sqlite:

Esta função faz a criação do banco de dados se ele ainda não existe e insere os dados dos resultados.

Juntando as operações:

Após criadas todas as funções necessárias para o projeto restavam criar uma função main.ml para o projeto e fazer com que ela recebesse os argumentos necessários para o filtro por linha de comando, também foram adicionados argumentos para definir nomes e locais para onde salvar cada um dos resultados e receber as entradas.

Para que esta etapa funcionasse corretamente foi vital organizar os arquivos conforme os padrões de um projeto dune e atualizar os arquivos dune de cada diretório do projeto com informações sobre bibliotecas criadas eplos diretórios e outras dependências externas.

Criando documentação:

Após finalizada a parte principal do projeto, para eu fosse possível gerar documentação através do comando dune build @doc, foram escritas docstrings para todas as funções, assim como os comentários do código forma reescritos para melhor compreensão.

Ler os dados de entrada através de uma requisição http:

Após finalizadas todas as etapas anteriores, optei por trocar o uso de arquivos csv locais por csv armazenados em um endereço http.

Para armazenar os arquivos, uma vez que eles eram pequenos eu optei pela solução mais fácil e sem custo adicional, de criar um repositório no github para estes arquivos e obte-los de maneira crua através de uma requisição direto ao endereço dos arquivos, desta maneira os endereços foram:

Endereço do repositório:

https://github.com/RodrigoAnciaes/Data_for_ETL_Ocaml

Arquivos:

github://RodrigoAnciaes/Data_for_ETL_Ocaml/main/order.csv
github://RodrigoAnciaes/Data_for_ETL_Ocaml/main/order_item.csv

Após isso, para ser capaz de realizar requisições web através do código Ocaml foi escolhido utilizar o pacote ocurl por familiaridade com o uso do curl em outras situações, para isso foi necessário instalar o pacote através do opam e atualizar as dependências dos arquivos dune.

Para que o código de leitura de dados fosse capaz de fazer a requisição e leitura dos arquivos, foi necessária a criação de uma função auxiliar (get_file_content) em lib/Read_data.ml, esta que é responsável por fazer a requisição e baixar os dados como arquivos temporários, O restante do código de Read_data.ml ficou majoritariamente inalterado com exceção de que as outras funções agora utilizam o arquivo gerado pela função auxiliar.

Por fim foi necessário alterar bin/main.ml para que este tivesse a capacidade de receber os endereços corretos dos arquivos e mostrasse feedback adequado para o usuário.

Uso de IA durante o desenvolvimento do projeto:

Github copilot:

Foi mantido ativo no editor durante todo o desenvolvimento, gerando múltiplas sugestões que foram aceitas em diversos momentos. A sua maior contribuição foi geralmente nas operações que escreviam nos records, com a lógica feita ele sugeria códigos coerentes de como passar os resultados para os records e dos records para outras estruturas.

Claude AI:

Foi utilizado para revisão do código, geração de doscstrings, reformulação dos cometários e me ajudou a fazer os argumentos de linha de comando funcionarem corretamente na main.

Ele também foi utilizado para gerar testes com base nos códigos e como afirmado no início deste relatório, eu os deixei de fora do resultado, por terem sido pouco modificados após sua geração, sendo escritos em sua maioria por IA e não por mim.