Ocaml ETL

Alexandre Magno Maciel dos Santos

1. Introdução

O objetivo deste projeto é realizar um processo completo de ETL - Extract, Transform e Load a partir de dados sobre pedidos e produtos. Dessa forma, gerar as saídas esperadas seguindo as três etapas citadas acima. Primeiramente, extrair dados vindos de CSVs que simulam tabelas de um banco de dados, em seguida, fazer as transformações necessárias e concluir salvando esses dados em novos CSVs.

Ao longo deste relatório serão discutidas a fundo as etapas, dificuldades e soluções para conclusão do ETL.

2. Leitura dos Dados

Nesta etapa serão abordados os processos de extração de dados, passando por abertura de documentos, tipagem e escrita de informações numa estrutura de dados adequada.

Para leitura de dados, foi necessário realizar uma análise dos datasets recebidos:

- order.csv - documento que contém id de um pedido e informações sobre cliente, data e status;
- order_item.csv - documento com os produtos de cada compra. Possui informações como ID do pedido, quantidade, preço e taxa.

Com essas informações foram criados os tipos order e item, com seus respectivos campos e tipagens.

Com os dados devidamente categorizados, foi possível passar para a próxima etapa, instalar a biblioteca CSV e escrever as funções: read_csv e parse_row, seguindo a estrutura main e helper functions. A primeira delas, read_csv, abre um arquivo, transforma as linhas do csv em lista e aplica parse_row em cada uma delas. Parse_row funciona como uma função auxiliar que mapeia os itens vindos da linha do csv em um record, utilizando tipagem para cada campo. Exemplo order_id: int.

- Esse processo foi realizado de forma igual para orders e items;
- Escritos em arquivos separados item_reader, order_reader;
- Processos impuros, como abertura de documentos e exibição de erros, ficaram isolados devido ao uso de funções auxiliares.
- Durante essa fase, foram testadas algumas abordagens de organização de projeto. A proposta inicial foi criar uma lib para cada tipo de funcionalidade, contudo, gerou problemas de importação dos módulos. A solução encontrada foi inserir todas as implementações na pasta bin, junto com o arquivo main. Além disso, criar um arquivo types.ml para armazenar de forma única todos tipos criados e importa-los quando necessário.

Como conclusão do processo de leitura, foi possível converter todos os dados da entrada em listas de records - order e item, para seguir para o processamento.

3. Processamento e Transformação dos Dados

Como requisitos, o cliente solicitou uma saída que apresentava o valor final e valor dos impostos de uma compra agrupando os itens de um determinado pedido. Dessa forma, foram criadas uma série de funções para juntar, agrupar, filtrar e processar valores.

3.1 Junção

Para essa etapa a proposta foi realizar um inner join entre os dados. Dessa forma, foi necessária a criação do type 'combined'. Ele consiste em um Joined entre order e item, pois dessa forma evita repetição de código e contorna uma limitação do Ocaml em lidar com tipos que tenham campos de mesmos nomes.

Essa solução surgiu após problemas na abordagem inicial de criar um record copiando todos os campos de order e item. Além da repetição de código, o Ocaml apresenta dificuldade inferir corretamente o tipo a ser usado caso haja nomes repetidos entre os tipos, mesmo aplicando tipagem nas definições das funções.

Internamente, o processo foi mapear a lista de orders em uma tupla: (order_id, order) e realizar um filter_map em item list. Para cada item, encontrar seu respectivo order usando assoc_opt em order_id e criar um Joined entre eles.

O resultado, por fim, foi uma nova lista de records que combina ambas as entradas.

3.2 Filtragem

Outra necessidade do cliente era parametrizar a saída de acordo com o status e origem específicos dos pedidos. Assim, foi criada a função filter_by que recebe as entradas pedidas, e realiza um filtro na lista de dados combinados. Para essa função a passagem de valores status e origin são opcionais, para os casos em que a filtragem não é necessária.

3.3 Processamento de Quantidade

Nesta etapa, foi necessário calcular o valor total e o montante de imposto para cada item. Para isso, a função items_amount_processing realiza um map em combined e, para cada item, multiplica sua quantidade pelo preço unitário, gerando o valor total. Da mesma forma, o imposto é calculado multiplicando o valor total pela taxa de imposto do item.

3.4 Agrupamento por ID do Pedido

Para a etapa final do processamento, foi necessário realizar a operação de agrupamento entre todos os Joined { order, item } da lista de combined. O processo escolhido foi utilizar o método fold_left com um acumulador - uma lista de tuplas (order.id, combined). Para cada linha de combined busca-se a presença do order.id atual no acumulador, caso encontrado seleciona o order correspondente, cria-se um novo record somando os dados de price e tax. Caso não encontrado, adiciona no contador a tupla (order.id, combined).

- O retorno dessa função é um combined para manter a consistência entre funções;
- Os valores originais de product_id e quantity referentes aos itens foram zerados por não serem considerados após agrupamento;
- Essa implementação não é a mais eficiente em relação a complexidade de tempo, porque a busca no acumulador ocorre em O(n). Assim, uma sugestão de melhoria seria implementar usando um hashmap, que realizar as buscas em O(1);

- A função se encerra com um map que aplica o método snd para descartar a tupla e devolver apenas os itens combined. snd (int,combined) -> combined;
- Um processo similar de agrupamento foi realizado em mean_by_month_and_year para juntar os itens de combined por data e aplicar média price e tax.

4. Geração e Salvamento dos Resultados

Após o processamento dos dados, acontece a transformação da lista de combined em um formato adequado para a saída. Para isso, as funções generate_output e generate_monthly_output convertem os dados combinados para o tipo output, contendo apenas as informações essenciais: ID do pedido, valor total e montante do imposto. Esses dados são então exportados para arquivos CSV por meio das funções to_csv e to_csv_monthly, que formatam e escrevem os resultados em arquivos estruturados, garantindo que as informações processadas possam ser utilizadas para análises futuras.

5. Requisitos Extras Atingidos

- Projeto realizado utilizando DUNE;
- Todas as funções foram documentadas seguindo o formato docstring (feito com auxílio de AI);
- Saída adicional que contém a média de receita e impostos pagos agrupados por mês e ano - salvo no arquivo: monthly_output.csv;
- Tratamento dos dados conjuntamente via operação de inner join pré etapa de Transform.
- Outputs salvos em SQLite3 (feito com auxílio de AI);
- Leitura dos dados de entrada em um arquivo estático na internet (exposto via http).

6. Conclusão

O projeto desenvolvido conseguiu implementar com sucesso uma pipeline completa de ETL utilizando OCaml, desde a extração dos dados de arquivos CSV até sua transformação e posterior armazenamento em novos arquivos de saída.

Durante o desenvolvimento, foram enfrentados desafios como a manipulação de tipos com campos de nomes repetidos, organização modular do código e otimização do agrupamento de records. As soluções aplicadas, como a criação do tipo combined, a separação do código em módulos específicos e o uso de operações como fold_left para agregação de dados, possibilitaram um código mais organizado e reutilizável.

Como possíveis melhorias futuras, poderiam ser exploradas otimizações no agrupamento de dados para reduzir a complexidade computacional, bem como a utilização de estruturas de dados mais eficientes, como Hashtbl, para melhorar o desempenho.

Em conclusão, o projeto cumpriu seus objetivos ao demonstrar uma abordagem funcional para ETL utilizando OCaml, apresentando uma solução flexível e bem estruturada para processamento de dados tabulares.