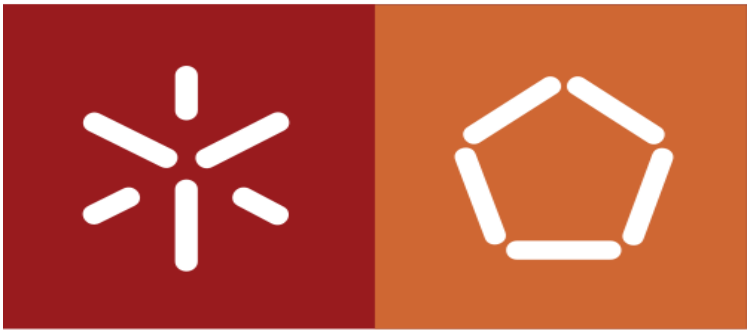
Laboratórios de Informática III – 2023/2024

Fase 1

João Pinto(a104270), Diogo Silva(a104183), Miguel Barrocas(a104272)

-- Grupo 75 --

Licenciatura em Engenharia Informática



Universidade do Minho
Escola de Engenharia

Departamento de Informática

Universidade do Minho

Índice

Introdução

Este projeto está a ser desenvolvido no âmbito da UC de Laboratórios de Informática III do ano 2023/2024, cujos objetivos de aprendizagem estão relacionados com o processo de programação, desenvolvimento de projetos e trabalho em equipa.

O projeto está dividido em duas fases, e esta primeira fase consiste em implementar o parsing (leitura dos ficheiros de entrada CSV) dos dados e o modo batch. Este modo consiste em executar várias queries(interrogações).

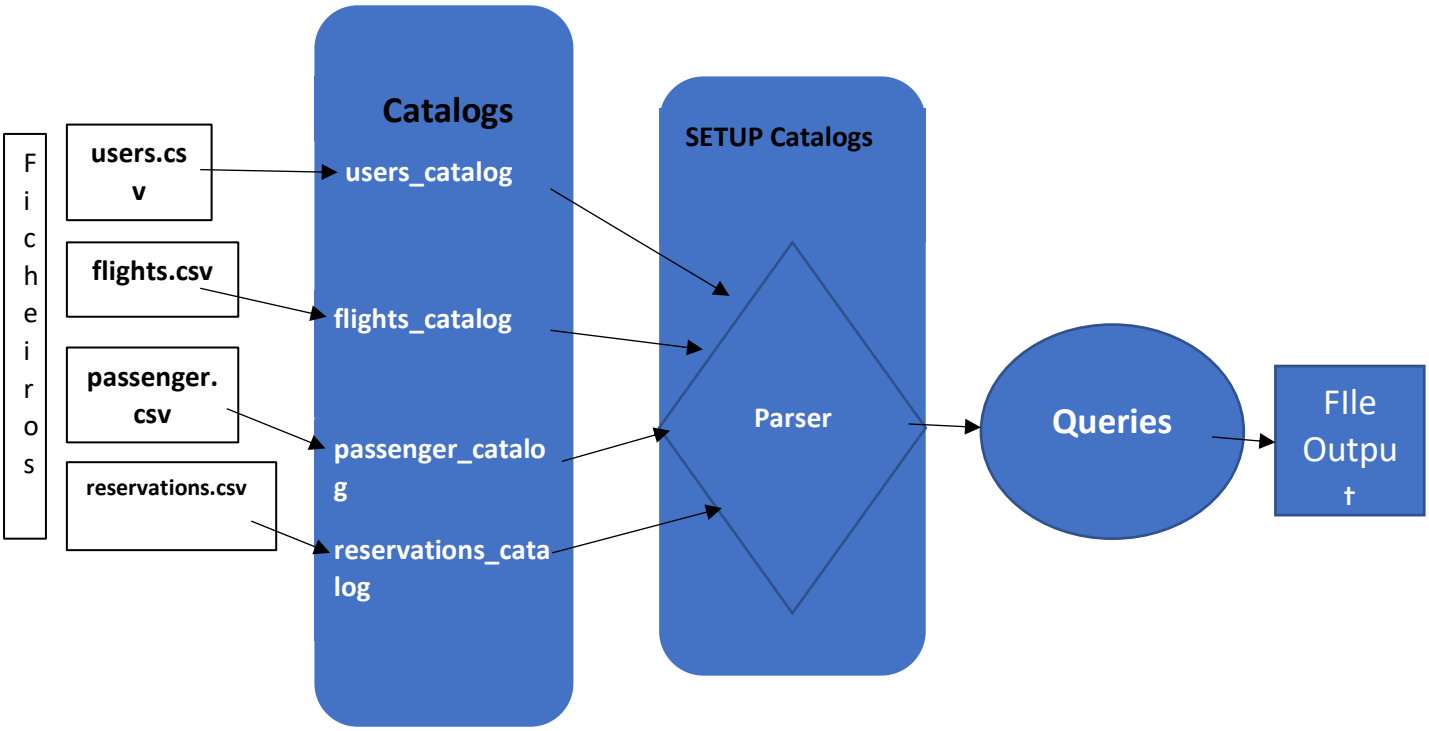
Sendo assim, o processo desta fase não implica só a leitura e interpretação dos dados, mas também o seu armazenamento em estruturas versáteis e de rápido acesso.

Uma das vantagens deste projeto é termos acesso à biblioteca glib que já possui implementações de várias estruturas úteis o que simplifica um bocado as coisas.

Desenvolvimento

Antes de partir para o código em si, analisamos bem o enunciado e começamos por destacar as coisas que mereciam especial atenção, como o formato das datas, arredondamentos, as descrições dos ficheiros de entrada, unidades, tipos, e por aí vai.

Depois disto, começamos a pensar em como seria a ligação entre todos os módulos e como se iam conectar. A arquitetura a que chegamos foi relativamente parecida à idealizada pelos docentes da UC.



Depois disto chegou a hora, então, de partir para a ação. O desafio inicial foi perceber quais as estruturas ideais para armazenar todos os dados. Concluímos que hash tables eram uma boa forma para as estruturas principais. Criamos, portanto, quatro: uma para os users, uma para os flights, uma para as reservations e outra para os passengers . De modo a assemelharem-se ao máximo a uma base de dados, decidimos (seguindo a arquitetura descrita em cima) agrupar cada uma numa estrutura mãe, o catálogo, onde cada uma das hash tables seria uma tabela da base de dados. Após isto passamos ao parsing dos dados. Aqui cada um dos catálogos tem o seu próprio parser pois do nosso ponto vista foi a forma mais fácil de implementar visto que cada um dos parsers recebe elementos diferentes isto pois para verificar os catálogos por vezes é necessário a verificação de outro, por exemplo

ao fazer o parse das reservations será necessário o catálogo dos users para verificar se o user é válido logo a reserva do mesmo também é válida, no entanto no parse dos flights não é necessário o catálogo dos users mas o do passengers, posto isto, foi esta a forma que achamos mais adequada e que fez mais sentido para nós.

Finalmente, de forma a finalizar o solicitado nesta primeira fase, optamos por escolher responder a estas seis queries. Pois, no nosso ponto de vista, eram aquelas cuja implementação era mais simples. As seis escolhidas foram:

Query 1: Esta primeira query consiste em "Listar o resumo de um utilizador, voo, ou reserva, consoante o identificador recebido por argumento", onde os identificadores são o "user_id" para os utilizadores, "flight_id" para os voos e "reservation_id" para as reservas. Tomemos como exemplo as seguintes linhas:

1. 1 DGarcia429
2. 1 0000000029
3. 1 Book0000000048

Decidimos então implementar uma função '**query1**' que começa por verificar o tipo de consulta e, em seguida, procura a entidade correspondente (usuário, voo ou reserva) nos catálogos fornecidos. Dependendo do tipo de entidade encontrada e das suas propriedades, ela constrói e retorna uma estrutura query1_result que contém informações formatadas sobre o usuário, voo ou reserva. Há algumas verificações adicionais, como o status do usuário (ativo ou inativo). Resumidamente, a função 'query1' interpreta comandos, busca informações no sistema com base nesses comandos e nos catálogos fornecidos, e retorna um resultado estruturado com as informações solicitadas. As funções auxiliares são responsáveis por gerenciar a alocação e "desalocação" de memória para evitar vazamentos de memória.

Query 3: A terceira query consiste em "Apresentar a classificação média de um hotel, a partir do seu identificador" (hotel_id). Assim sendo, o que nos pareceu fazer mais sentido foi implementar uma função '**query3**' que processa consultas relacionadas a reservas de hotéis. Ela recebe comandos e um catálogo de reservas como entrada. O objetivo principal é calcular a média das avaliações de hotéis associadas a reservas específicas, usando o ID da entidade fornecido nos comandos. A função verifica se o ID da entidade é válido, calcula a média das avaliações e, se a média for não negativa, cria uma estrutura de resultado contendo essa informação. A formatação do resultado é determinada pelo tipo de comando. A função `free_query3_result` é responsável por liberar a memória alocada para a estrutura de resultado quando necessário.

Query 4: A quarta query consiste em "Listar as reservas de um hotel, ordenadas por data de início" em que o critério de desempate para duas reservas que tenham a mesma data é o identificador da reserva (de forma crescente). Então, criamos a função ´**query4**` que

processa consultas em um sistema de reservas de hotéis. Ela recebe comandos e um catálogo de reservas, filtrando as reservas associadas a um hotel específico (determinado

pelo ID fornecido nos comandos). Em seguida, ordena essas reservas por data de início (da mais recente para a mais antiga) e ID. O resultado é formatado de acordo com o tipo de comando e inclui a lista ordenada de reservas. A função free_query4_result é responsável por liberar a memória alocada durante esse processo. Resumidamente, a função facilita a obtenção e ordenação de reservas de um hotel em um sistema de reservas de hotéis.

Query 6: A sexta query consiste em "Listar o top N aeroportos com mais passageiros, para um dado ano". Por isso utilizamos a função ´**query6**` que realiza uma consulta estatística em um catálogo de voos, agregando o número total de passageiros por aeroporto para um ano específico. Ao receber comandos, a função itera sobre a lista de voos, identifica aqueles que correspondem ao ano indicado e acumula o número de passageiros para cada origem e destino. Os resultados são ordenados pela quantidade de passageiros e, em caso de empate, pelo código do aeroporto.

Query 7: A sétima query consiste em "Listar o top N aeroportos com a maior mediana de atrasos". Com isto construímos a função ´**query7**` que processa consultas em um sistema de gerenciamento de voos. Ela recebe comandos e um catálogo de voos, identifica a mediana dos atrasos para cada aeroporto de origem, e retorna uma lista de aeroportos ordenada com base nessas medianas e nos códigos de origem. A função free_query7_result é responsável por liberar a memória alocada durante esse processo.

Query 9: A nona query consiste em" Listar todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome (de forma crescente)", onde utilizadores inativos não são considerados pela pesquisa e em caso de "empate" por dois utilizadores com o mesmo nome, os seus identificadores (de forma crescente) são usados como resolução do problema. Com isto tudo, decidimos então aplicar a função ´**query9**` que recebe comandos e um catálogo de usuários, realizando consultas a procura de usuários cujos nomes têm um prefixo específico (entity_id). A condição da consulta inclui verificar se o status da conta do usuário é ativo. A função utiliza funções auxiliares para remover acentos, hífens e espaços de modo que a ordenação seja feita sem erros, e implementa uma lógica de comparação personalizada para ordenar os resultados. A lista de usuários filtrada e ordenada é então retornada e a liberação de memória é realizada no final do processo.

Dificuldades Sentidas

Como é óbvio ao longo do projeto sentimos diversas dificuldades, que se foram resolvendo com várias tentativas e pesquisas. Listaremos então a seguir algumas das mais presentes e marcantes:

- Ao usar a biblioteca glib, no princípio era difícil entender algumas das suas funções o que tornou complicado os avanços do projeto, mas após melhor perceção, tornou-se útil;
- Datas: não sabíamos como íamos comparar as datas então decidimos passar as datas para epoch time e inicialmente não estava a funcionar, mas depois conseguimos
- escrever um algoritmo com a ajuda do link que esta no código, conseguindo assim verificar se as mesmas iam de acordo com o esperado;
- Na query 9 que consiste em listar todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome (de forma crescente). Tivemos problemas na ordenação dos outputs pois ao tentar ordenar com a função strcmp (que compara os valores ASCII de cada elemento da string) aqueles que tinham acentos eram colocados no final, então como solução decidimos criar um algoritmo que procura caracteres com acentos na string e estes são substituídos pelo equivalente sem acento, não interferindo assim na ordenação. A mesma situação funcionou para os hífens e os espaços;
- Por último, a implementação cuidadosa do encapsulamento e da modularidade. Percebemos que estes são fundamentais para um programa de qualidade, mas em C a sua aplicação não é a mais fácil, tendo em conta a linguagem. De qualquer das formas, achamos que acabamos por conseguir implementar os mesmos em grande parte do projeto.

Conclusão

Concluindo, achamos que conseguimos responder bem aos desafios implementados pela primeira fase do projeto, construindo uma boa base que vai dar jeito para as restantes queries que vão ser abordadas na segunda fase. Sendo assim, sentimo-nos confiantes e

com vontade de concluir o projeto que só nesta primeira fase já nos desafiou e ensinou tanto sobre alguns aspetos.