UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Astrix

Gustavo Alves da Silva Souza nºUSP 13727485

Felipe de Souza Melo Siqueira nºUSP 11218598

Guilherme Henrique Galdini Tosi n°USP 11781587

Lucas Silva Neves n°USP 10288633

Nelson Luiz Serpa de Oliveira nºUSP 9793502

São Carlos, 2023

Bases de Dados

Exercício apresentado ao Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação.

Área de Concentração: Introdução a Sistemas de Informação.

Orientador: Profª. Drª Elaine Parros Machado de Sousa

São Carlos, 2023

SUMÁRIO

1) Descrição do Problema e dos Requisitos de Dados

O ano é 3337, em um mundo onde a exploração espacial atingiu níveis inimagináveis, com **viagens** entre a estação de cada **planeta** se concretizando em minutos. É nesse contexto que surge a Astrix, cujo nome tem origem no deus grego Astraeus, o guardião das estrelas e do crepúsculo estrelado, como uma homenagem à conexão entre o céu noturno e a velocidade do espaço.

Astrix dispõe de sua própria frota de **espaçonaves**, possuindo números de séries únicos, para atender a uma ampla gama de clientes, esses clientes possuem um identificador único chamado CC (Cadastro de Cosmonauta), que se dividem em duas categorias:

- **Cosmonautas Físicos (CF)**: Este termo é utilizado para classificar qualquer indivíduo que viaje pelo espaço. Cada um deles possui um identificador único intergaláctico conhecido como CCF (Cadastro de Cosmonauta Físico) e pode reservar uma **viagem** intergaláctica a qualquer momento. As reservas são realizadas com base nos assentos disponíveis na aeronave de transporte. Um mesmo CF pode realizar diversas viagens diferentes.

- **Cosmonautas Jurídicos (CJ)**: Este termo é empregado para identificar empresas, independentemente de seu porte. Cada empresa possui um identificador único denominado CCJ (Cadastro de Cosmonauta Jurídico). Os CJ têm a opção de contratar os serviços da Astrix apenas para o **transporte** de cargas interplanetárias, ao qual um mesmo CJ pode contratar o serviço de transporte de cargas várias vezes, mas para cada transporte é gerado um identificador único.

O serviço de **transporte de carga** tem como origem apenas um **Planeta**, possuindo **nome** único em toda galáxia e seu código **CG** (Coordenada Galáctica), mas pode fazer a entrega dessa mesma carga em diversos planetas diferentes. As

informações do produto que está sendo transportado não ficam armazenadas. O valor do transporte varia de acordo com a distância entre os Planetas e Modelo da espaçonave escolhida, variando sua capacidade de carga.

Já o **transporte de cosmonautas** tem como origem e destino apenas um Planeta, mas uma mesma viagem pode ter diversas paradas em outros planetas. O valor da viagem é calculado a partir da distância entre os planetas e o modelo da espaçonave escolhida.

As espaçonaves da Astrix incorporam a mais avançada tecnologia galáctica e contam com sistemas de navegação de última geração. Isso garante **viagens** e **transportes** seguros e eficientes entre os **planetas**, minimizando o tempo de deslocamento. Além disso, a Astrix oferece quatro modelos distintos de **espaçonaves** para atender melhor às necessidades de seus clientes, divididos entre dois tipos:

1. **Transporte:**
    a. **Spacebus A380**: Trata-se de um ônibus espacial com capacidade para transportar até 50 Cosmonautas, oferecendo o melhor custo-benefício.
    b. **Spacecar A770**: Esta é uma espaçonave privativa espacial, ideal para aqueles que desejam se locomover apenas com cosmonautas conhecidos, desfrutando de maior conforto e segurança. A capacidade é limitada a 4 pessoas.
2. **Carga:**
    a. **Spacetrain J1000**: Um trem espacial com capacidade para transportar até 1 tonelada de carga.
    b. **Spacetran F2000**: Outro trem espacial, porém com capacidade para transportar até 1000 toneladas de carga.

O sistema também armazena os dados dos **funcionários** da Astrix, possuindo um código de cadastro único, chamado **CF** (Cadastro Funcionário). Além de armazenar outras informações, como **nome** e **endereço**, contendo o armazenamento do **nome do planeta**, **coordenada galáctica (CG)** e **número da**

moradia. Os funcionários são segregados em três tipos, de acordo com sua atuação dentro da empresa, sendo eles:

1. **Administrador**: responsável por administrar todo banco de dados da empresa;
2. **Empregado**: realiza serviços de manutenção nas espaçonaves, ao qual cada manutenção possui um código único de identificação (número sequencial) e, para cada serviço, são registrados a data, os problemas encontrados e as respectivas soluções. Um mesmo empregado pode trabalhar na mesma espaçonave várias vezes, mas para cada uma delas é gerado um novo código de serviço.
3. **Piloto**: possui um identificador único chamado CP (Cadastro Piloto), sendo necessário para que possa pilotar as espaçonaves.

A Astrix está pronta para atender às necessidades de transporte intergaláctico de seus clientes, seja qual for sua categoria e demanda específica.

1.1) Principais operações (funcionalidades)

- **Cosmonautas Físicos (CF)**:
  - Criar, buscar, editar e deletar seu perfil;
  - Buscar por planeta;
  - Buscar por viagens do Spacebus A380;
  - Buscar por viagens do Spacecar A770;
  - Consultar os dados do piloto;
  - Consultar os dados da Espaçonave;
  - Consultar os dados de uma viagem, incluindo quantidade de assentos disponíveis, planetas em que possuí alguma parada e o valor;

- **Cosmonautas Jurídicos (CJ)**:
  - Criar, buscar, editar e deletar seu perfil;
  - Buscar por planeta;
  - Buscar por viagens do Spacetrain J1000;
  - Buscar por viagens do Spacetran F2000;
  - Consultar os dados do piloto;
  - Consultar os dados da Espaçonave;
  - Consultar os dados de um transporte, incluindo o valor e planetas de destino;

- **Piloto**:
  - Buscar por Cosmonauta Físico;
  - Buscar por Cosmonauta Jurídico:
  - Buscar por transporte;
  - Busca por viagem;
  - Buscar por planeta;
  - Busca por espaçonave;

- **Administrador**:
  - Criar, buscar, editar e deletar seu perfil;
  - Criar, buscar, editar e deletar perfil do Piloto;
  - Criar, buscar, editar e deletar perfil do Empregado;;
  - Criar, buscar, editar e deletar perfil do Cosmonauta Físico;
  - Criar, buscar, editar e deletar perfil do Cosmonauta Jurídico;
  - Criar, buscar, editar e deletar Espaçonave;
  - Criar, buscar, editar e deletar Planeta;

1.2) Principais consultas não triviais

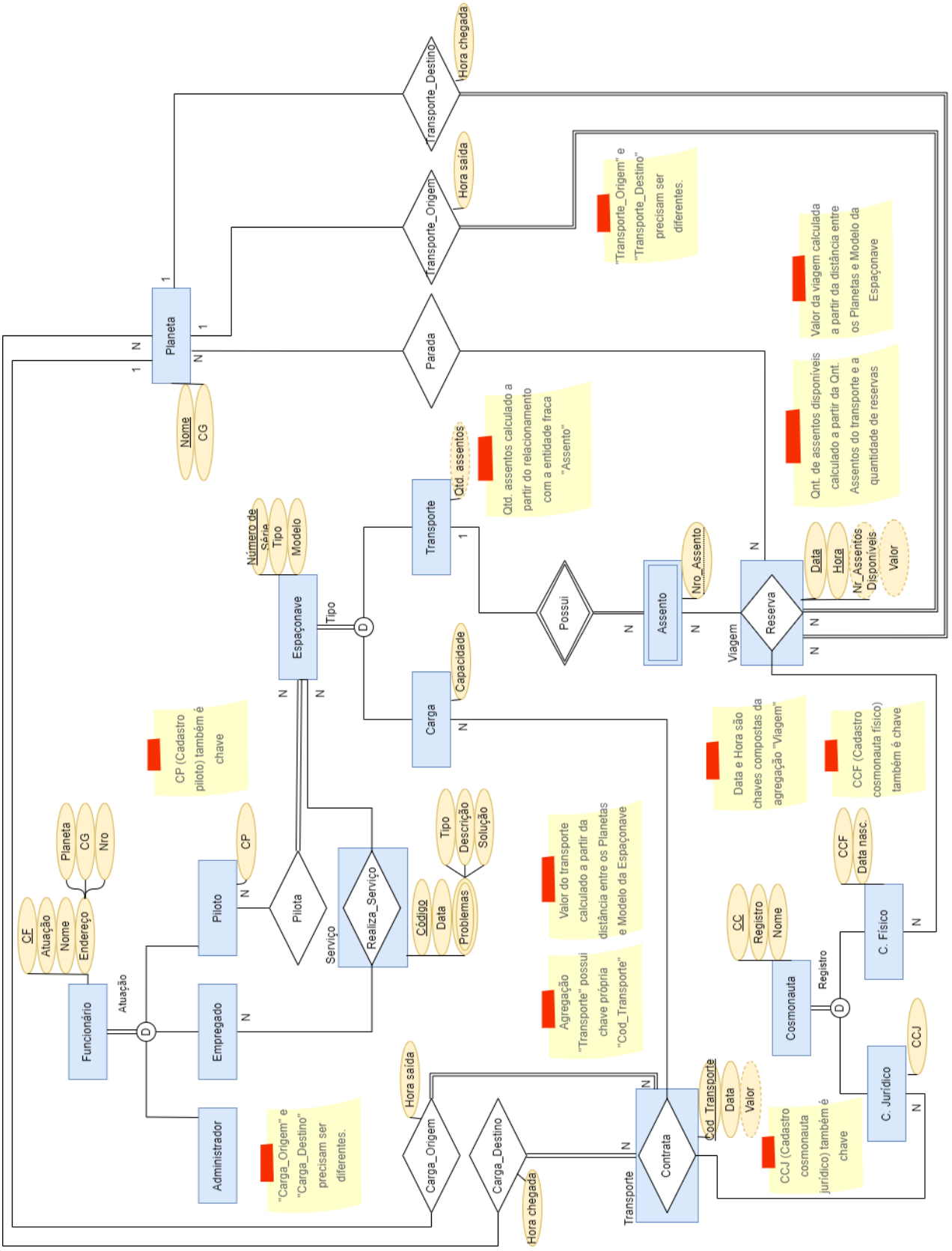
- **Relatório de Desempenho do Piloto**: Para os Pilotos, gerar um relatório de desempenho que inclua estatísticas sobre o número de viagens concluídas.

- **Consulta Planetas Visitados**: Permitir que Cosmonautas Físicos e Cosmonautas Jurídicos consultem uma lista de planetas que já visitaram em suas viagens intergalácticas.

- **Gestão de Manutenção de Espaçonaves**: Implementar um sistema de acompanhamento da manutenção das Espaçonaves, agendando manutenções preventivas e gerando alertas quando uma nave precisar de manutenção.

2) Projeto Conceitual

2.1) Diagrama do Modelo Entidade-Relacionamento



2.2) Possíveis problemas/inconsistências do MER

- **Ciclo: Viagem → Planeta**: Há a presença de dois ciclos entre as entidades “Viagem” e “Planeta”, uma considerando os relacionamentos “Parada” e “Transporte_Origem” e outra “Parada” e “Transporte_Destino”, ao qual apresentam a mesma fragilidade: possibilidade de inconsistências entre “Origem”/”Destino” e “Parada”, especialmente sobre a existência de tuplas dos primeiros que não pertencem a esse último relacionamento. É apropriado tratar essa vulnerabilidade em nível de aplicação.

- **Ciclo: Transporte→ Planeta:** Assim como no ciclo descrito acima, existe a mesma vulnerabilidade neste ciclo entre as entidades “Transporte” e “Planeta”.

- **Possível inconsistência entre “Carga_Origem” e “Carga_Destino”:** A origem e destino precisam ser planetas diferentes, sendo necessário atender esse requisito no SQL ou aplicação.

- **Possível inconsistência entre “Transporte_Origem” e “Transporte_Destino”:** A origem e destino precisam ser planetas diferentes, sendo necessário atender esse requisito no SQL ou aplicação.

- **Possível inconsistência entre o “Modelo” da Espaçonave e o “Tipo” dela:** sendo necessário atender os requisitos descritos no problema no SQL ou aplicação.

- **Possível inconsistência entre o número de assentos da viagem:** não dá para garantir no esquema que um cosmonauta não consiga reservar uma viagem mesmo que a espaçonave já esteja lotada, sendo necessário tratar no SQL ou aplicação.

2.3) Mudanças relacionadas à Primeira Entrega

- **Descrição completa do MER:** foi incrementado o descritivo do sistema em termos de entidades, relacionamentos e atributos. Assim como o ajuste de alguns requisitos, que serão descritos abaixo, a fim de facilitar a visualização do MER.

- **Escopo da Astrix:** as entidades “Objeto Celeste”, “Sistema Estelar” e “Galáxia” não estavam sendo relevantes para o sistema. Dessa forma, foi definido alterar o escopo da Astrix para somente Planetas, considerando que existe apenas uma estação em cada uma delas.

- **Definição dos tipos de espaçonaves:** transporte e carga.

- **Descrição e funcionalidades dos funcionários da Astrix.**

- **Inserção da agregação “Serviço”:** com o descritivo completo do MER, ficou definido que o relacionamento “Realiza_Serviço”, entre as entidades “Empregado” e “Espaçonave”, pode ser realizado por um empregado na mesma espaçonave várias vezes, mas para cada uma delas é gerado um novo código de serviço, sendo necessário a agregação.

- **Inserção da generalização “Espaçonave”:** a generalização estava modelada como entidade única, não atendendo as restrições do tipo de cada espaçonave e os serviços que cada uma realiza. Dessa forma, foram criadas as especializações “Carga”, serviço de transporte de carga oferecido para os Cosmonautas Jurídicos, e “Transporte”, serviço de viagem de Cosmonautas Físicos. Assim como foram criados relacionamentos próprios para cada uma das especializações.

- **Remoção da generalização “Objeto Celeste”:** como comentado acima, foi alterado o escopo da Astrix. Dessa forma, toda generalização foi removida do sistema e a entidade fraca modelada como “Estação” foi substituída pela entidade forte “Planeta”.

- **Remoção da agregação “Ponto de Parada”:** já que não possuía chave própria e a instância agregada não se relacionava com nenhuma entidade, então o relacionamento “Parada” já é o suficiente.

- **Inserção da agregação “Transporte”:** por conta da criação da entidade específica “Carga” e seu relacionamento com a entidade “C. Jurídico” foi necessário a criação da agregação “Transporte”, já que para cada par do relacionamento é gerado um novo serviço de transporte com código de transporte único.

- **Inserção da agregação “Viagem”:** por conta de um mesmo “C. Físico” poder reservar diferentes “Assentos” no transporte em dia/horário diferente resultando em uma nova viagem, sendo necessário a utilização da agregação.

- **Inserção da entidade fraca “Assento”:** como cada aeronave possui um número máximo de assentos, cada viagem possui um número máximo de reservas. Sendo assim, foi necessário criar a entidade fraca “Assento” da especificação “Transporte”, para que assim o sistema consiga fazer o controle das reservas.

- **Remoção da generalização “Passagem”:** com a criação das especificações “Carga”, “Transporte” e seus respectivos relacionamentos, não foi mais necessário a utilização da generalização “Passagem”.

- **Indicação de como o atributo derivado “Qtd_assentos” é calculado:** a partir do relacionamento com a entidade fraca “Assento”.

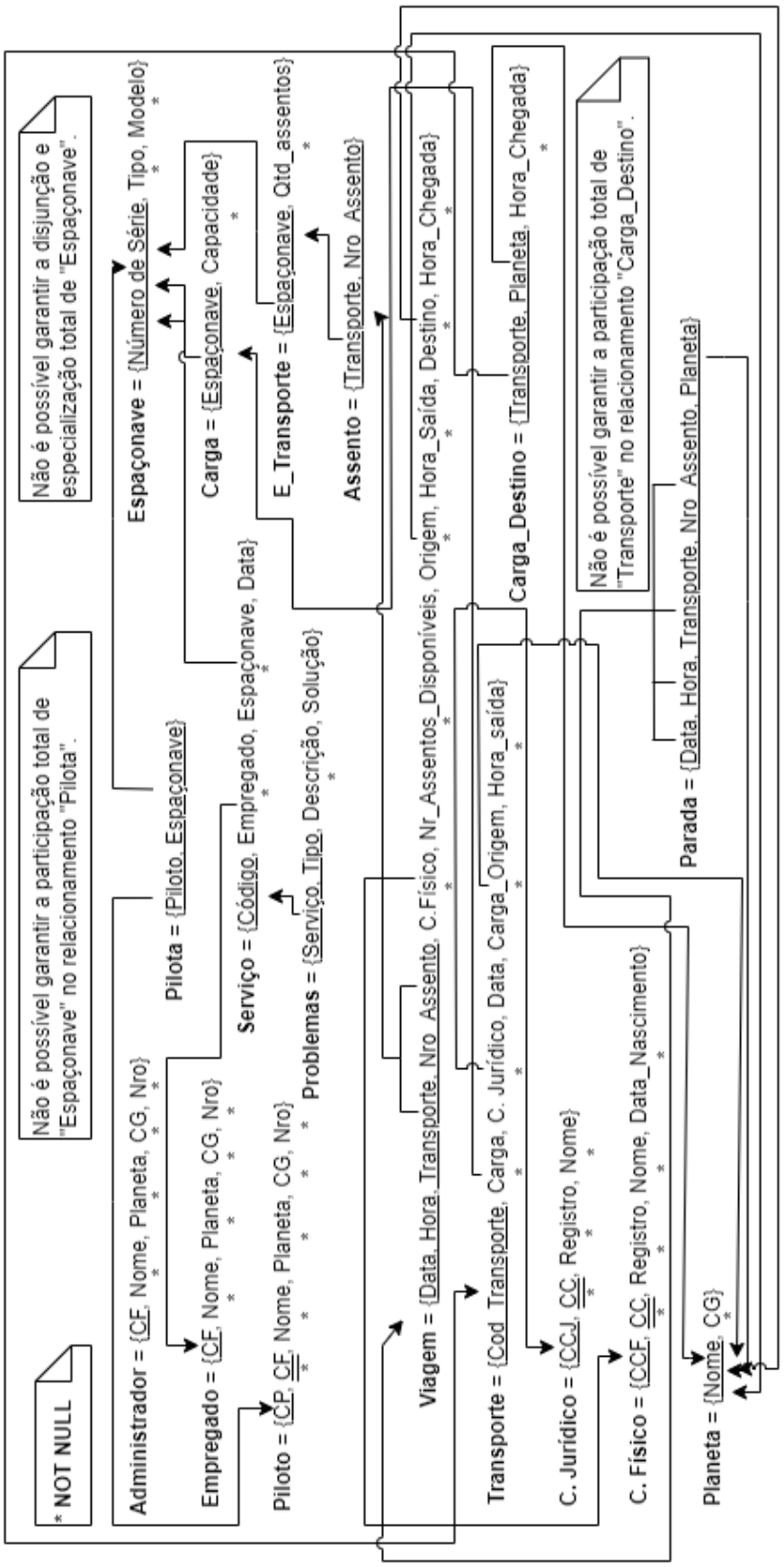
- **Indicação de como o atributo derivado “Nr_Assentos” é calculado:** a partir da quantidade de assentos do transporte e a quantidade de reservas para o mesmo.

- **Indicação de como o atributo derivado “Valor” é calculado:** a partir da distância entre os planetas e o modelo da espaçonave.

- **Alteração do nome da chave “Código” da entidade “Espaçonave” para “Número de Série”:** a fim de ser mais entendível em alto nível

3) Projeto Lógico

3.1) Diagrama do Modelo Relacional



3.2) Discussão sobre os mapeamentos propostos

A seguir estão listadas as justificativas quanto às tomadas de decisão durante o processo de mapeamento do modelo conceitual (MER) para o projeto lógico (MREL).

1. **Generalização “Funcionário”:**
   - ➢ **Solução adotada:** Foram criadas apenas as tabelas das especializações "Administrador", "Empregado" e "Piloto". Nas duas primeiras, a chave é o atributo "CF" (Cadastro Funcionário), enquanto na especialização "Piloto", a chave primária é o atributo "CP" (Cadastro Piloto) e a chave secundária é "CF". Todos os outros atributos da entidade genérica foram replicados nas especializações com a restrição "NOT NULL". A criação da entidade genérica não foi necessária, já que não possui relacionamentos com outras entidades do sistema. Além disso, devido ao foco das consultas, não houve necessidade de armazenar o tipo da especialização.
   - ➢ **Vantagens:** O mapeamento adotado assegura a especialização total de "Funcionários" e ainda proporciona economia de recursos. Ao evitar a criação de tabelas adicionais para tipos de especialização ou entidades genéricas, o sistema preserva recursos de memória e poder de processamento, tornando-o mais leve e ágil durante as operações diárias.
   - ➢ **Desvantagens:** Não garante a disjunção e a consulta precisa ser focada, já que não é armazenado o tipo da especialização. No nosso sistema, isso acaba não sendo um problema, uma vez que as consultas serão específicas e focalizadas.
   - ➢ **Alternativa:** Uma primeira alternativa seria criar uma tabela para o tipo de especialização, mas dado que nosso sistema consistirá apenas em consultas específicas, isso resultaria apenas em um aumento de custo de memória desnecessário. Por outro lado, uma segunda solução de mapeamento envolveria a criação de uma tabela separada para cada tipo de especialização, juntamente com uma para a entidade genérica. No entanto, isso não garantiria uma especialização total, e dado que a entidade genérica não possui nenhum relacionamento, isso também resultaria apenas em um aumento de custo de memória sem benefícios claros para o sistema.

2. **Agregação “Serviço”:**
   - ➢ **Solução adotada:** O relacionamento "Realiza_Serviço" entre as entidades "Empregado" e "Espaçonave", que gera a agregação "Serviço", possui uma cardinalidade N:N e não tem atributos próprios. Portanto, foi mapeado dentro da agregação, com "Código" como chave primária e "Empregado" e "Espaçonave" como chaves estrangeiras,

com a restrição "NOT NULL", garantindo que cada relação entre as entidades resulte em um novo "Serviço".
- ➢ **Vantagens:** Garante todas as restrições da modelagem.
- ➢ **Desvantagens:** Não há.
- ➢ **Alternativa:** Não há.

3. **Generalização “Espaçonave”:**
   - ➢ **Solução adotada:** Foi criada uma tabela para representar a generalização "Espaçonave" e outras duas tabelas para cada tipo de especialização, "Carga" e "Transporte", todas com "Número_de_Série" como chave primária. Nas tabelas de especialização, a chave primária também é utilizada como chave estrangeira referenciando a tabela genérica.
   - ➢ **Vantagens:** Cada entidade possui relacionamentos e atributos exclusivos, tornando essa a solução ideal para reduzir o custo de memória e manter a consistência dos relacionamentos.
   - ➢ **Desvantagens:** Não garante disjunção e especialização total, e ainda aumenta o custo de busca devido à necessidade de usar operações "JOIN".
   - ➢ **Alternativa:** Uma primeira opção, que não é viável, seria colocar todas as entidades em uma única tabela. Contudo, devido aos relacionamentos e atributos exclusivos de cada entidade, haveria uma grande quantidade de valores "NULL", aumentando consideravelmente o custo de memória. Outra solução seria criar apenas as tabelas das especializações, garantindo assim a especialização total. No entanto, como a entidade genérica possui dois relacionamentos, seria necessário replicar esses relacionamentos para as entidades específicas, o que, novamente, aumentaria o custo de memória.

4. **Entidade fraca “Assento”:**
   - ➢ **Solução adotada:** Criação de uma tabela para entidade fraca “Assento”, com chave primária composta “Transporte” e “Nro_Assento”.
   - ➢ **Vantagens:** Garante que todo “Assento” será de alguma “Espaçonave” do tipo “Transporte” de acordo com o número do assento.
   - ➢ **Desvantagens:** Não garante que toda “Espaçonave” do tipo “Transporte” tenha “Assento”.
   - ➢ **Alternativa:** Não há.

5. **Agregação “Viagem”:**
   - ➢ **Solução adotada:** O relacionamento "Reserva" entre as entidades "Assento" e "C.Físico", que gera a agregação "Viagem", possui uma cardinalidade N:N e não tem atributos próprios. Portanto, foi mapeado dentro da agregação, com chave primária composta: "Data", “Hora”, “Transporte” e “Nro_Assento” e "Transporte", “Nro_Assento”,

"C.Físico", “Origem” e “Destino” como chaves estrangeiras, com a restrição "NOT NULL", garantindo que cada relação entre as entidades resulte em uma nova "Viagem".

- ➢ **Vantagens:** Garante que diferentes “C.Fisicos” possam realizar a mesma viagem, com “Nro_Assento” diferentes.
- ➢ **Desvantagens:** É necessário garantir que, para uma mesma combinação de {Data, Hora, Transporte}, as informações {Origem, Hora_Saída, Destino, Hora_Chegada} sejam cadastradas de maneira consistente, evitando inconsistências. Desta forma, será necessário implementar mecanismos de validação e controle para garantir a consistência dos dados, especialmente ao lidar com inserções, atualizações e exclusões. Além da chave ser muito grande para busca.
- ➢ **Alternativa:** Criar uma tabela separada para “Reserva”. Entretanto, iria existir o mesmo problema de inconsistência de dados entre “Transporte” e “Nro_Assento”, sendo necessário tratar nas próximas etapas também.

6. **Agregação “Transporte”:**
   - ➢ **Solução adotada:** O relacionamento "Contrata" entre as entidades "Carga" e "C.Jurídico", que gera a agregação "Transporte", possui uma cardinalidade N:N e não tem atributos próprios. Portanto, foi mapeado dentro da agregação, com "Cod_Transporte” como chave primária e "Carga” e "C.Jurídico" como chaves estrangeiras, com a restrição "NOT NULL", garantindo que cada relação entre as entidades resulte em um novo "Transporte".
   - ➢ **Vantagens:** Garante todas as restrições da modelagem.
   - ➢ **Desvantagens:** Não há.
   - ➢ **Alternativa:** Não há.

7. **Generalização “Cosmonauta”:**
   - ➢ **Solução adotada:** Foram criadas apenas as tabelas das especializações "C. Jurídico" e "C.Físico". Cada uma possui uma chave primária específica, sendo “CCJ” (Cadastro Cosmonauta Jurídico) a chave primária de “C. Jurídico” e “CCF” (Cadastro Cosmonauta Físico) de “C. Físico. Ambos possuem “CC” (Cadastro Cosmonauta) como chave secundária. Todos os outros atributos da entidade genérica foram replicados nas especializações com a restrição "NOT NULL". A criação da entidade genérica não foi necessária, já que não possui relacionamentos com outras entidades do sistema. Além disso, devido ao foco das consultas, não houve necessidade de armazenar o tipo da especialização.
   - ➢ **Vantagens:** O mapeamento adotado assegura a especialização total de "Cosmonauta" e ainda proporciona economia de recursos. Ao evitar a criação de tabelas adicionais para tipos de especialização ou

entidades genéricas, o sistema preserva recursos de memória e poder de processamento, tornando-o mais leve e ágil durante as operações diárias.

- ➢ **Desvantagens:** Não garante a disjunção e a consulta precisa ser focada, já que não é armazenado o tipo da especialização. No nosso sistema, isso acaba não sendo um problema, uma vez que as consultas serão específicas e focalizadas.
- ➢ **Alternativa:** Uma primeira alternativa seria criar uma tabela para o tipo de especialização, mas dado que nosso sistema consistirá apenas em consultas específicas, isso resultaria apenas em um aumento de custo de memória desnecessário. Por outro lado, uma segunda solução de mapeamento envolveria a criação de uma tabela separada para cada tipo de especialização, juntamente com uma para a entidade genérica. No entanto, isso não garantiria uma especialização total, e dado que a entidade genérica não possui nenhum relacionamento, isso também resultaria apenas em um aumento de custo de memória sem benefícios claros para o sistema.

8. **Atributo multivalorado “Problemas”:**
   - ➢ **Solução adotada:** Foi criada uma tabela para o atributo multivalorado "Problemas", tendo como chave primária composta apenas os campos "Serviço" e "Descrição". Isso ocorre porque o atributo "Solução" pode ser nulo nos casos em que não houver soluções.
   - ➢ **Vantagens:** Garante que diversos “Problemas” sejam armazenados.
   - ➢ **Desvantagens:** Maior custo de memória, por ser necessário criar uma tabela nova, e custo de junções.
   - ➢ **Alternativa:** Nesse caso não há alternativas, já que não sabemos a quantidade de “Problemas”.

9. **Atributo derivado “Nr_Assentos_Disponíveis”:**
   - ➢ **Solução adotada:** O atributo foi mapeado dentro da tabela “Viagem” com restrição de “NOT NULL”.
   - ➢ **Vantagens:** Possui o número atualizado de assentos disponíveis, facilitando na hora da reserva. O cálculo é realizado de acordo com o número de assentos da “Espaçonave” subtraindo o número de “Reserva” para viagem.
   - ➢ **Desvantagens:** Maior custo de memória, já que terá que armazenar sempre esse dado atualizado na base de dados, possuindo constantes atualizações. Entretanto, o custo para calcular ainda sim é menor do que se tivesse que fazer por demanda.
   - ➢ **Alternativa:** Calcular sobre demanda direto na aplicação, mas o custo seria maior, já que seria necessário calcular toda vez que houvesse uma nova reserva.

10. **Atributo derivado “Valor”:**
    - ➢ **Solução adotada:** O atributo derivado “Valor” não foi mapeado nas tabelas de “Transporte” e “Viagem” por conta do custo para calcular e manter esse dado atualizado, além de possíveis atualizações futuras na forma de calcular esse “Valor”. Dessa forma, esse atributo terá que ser calculado sob demanda na própria aplicação.
    - ➢ **Vantagens:** Menor custo para calcular e facilidade em atualizações futuras.
    - ➢ **Desvantagens:** Não possui o dado atualizado na base de dados.
    - ➢ **Alternativa:** Calcular e armazenar o “Valor” direto na base de dados.

11. **Relacionamento 1:N “Transporte_Origem”:**
    - ➢ **Solução adotada:** O relacionamento “Transporte_Origem” e seu atributo foram mapeados dentro da entidade “Viagem” utilizando NOT NULL.
    - ➢ **Vantagens:** Garante a cardinalidade 1:N e participação total de “Viagem” com o relacionamento “Transporte_Origem”.
    - ➢ **Desvantagens:** Não garante que “Transporte_Origem” seja diferente de “Transporte_Destino”, sendo necessário tratar no SQL ou aplicação.
    - ➢ **Alternativa:** Não há.

12. **Relacionamento 1:N “Transporte_Destino”:**
    - ➢ **Solução adotada:** O relacionamento “Transporte_Destino” e seu atributo foram mapeados dentro da entidade “Viagem”.
    - ➢ **Vantagens:** Garante a cardinalidade 1:N e participação total de “Viagem” com o relacionamento “Transporte_Destino”.
    - ➢ **Desvantagens:** Não garante que “Transporte_Destino” seja diferente de “Transporte_Origem”, sendo necessário tratar no SQL ou aplicação.
    - ➢ **Alternativa:** Não há.

13. **Relacionamento 1:N “Carga_Origem”:**
    - ➢ **Solução adotada:** O relacionamento “Carga_Origem” e seu atributo foram mapeados dentro da entidade “Transporte”.
    - ➢ **Vantagens:** Garante a cardinalidade 1:N e participação total de “Transporte” com o relacionamento “Carga_Origem”.
    - ➢ **Desvantagens:** Não garante que “Carga_Origem” seja diferente de “Carga_Destino”, sendo necessário tratar no SQL ou aplicação.
    - ➢ **Alternativa:** Não há.

3.3) Mudanças relacionadas à Segunda Entrega

- **Alteração da chave de “Problemas”:** a chave da entidade era composta, sendo o código e descrição do serviço. Entretanto, a descrição poderá ser grande, na maioria dos casos, tornando, consequentemente, a chave grande, dificultando nas buscas. Dessa forma, foi criado o atributo “Tipo” do problema, encurtando o tamanho da chave. A chave composta de “Problemas” agora é código e tipo.

- **Inserção da chave estrangeira de “Carga_Origem”:** o atributo “Carga_Origem” da entidade “Transporte” não estava referenciando para a chave da entidade “Planeta”. Dessa forma, foi inserido a chave estrangeira para corrigir esse problema.

- **Alteração da localização das chaves estrangeiras:** no desenho do MRel, as chaves estrangeiras dos atributos “Origem” e “Destino”, da entidade “Viagem”, estavam, incorretamente, nos atributos “Hora_Saida” e “Hora_Chegada”. Isso ocorreu por alguma mudança no desenho e passou despercebido, mas já foi arrumado.

- **Alteração da chave composta de “Viagem”:** A chave composta inicial da entidade "Viagem" consistia apenas nos atributos "Data" e "Hora", permitindo apenas uma viagem por data e hora com um único cosmonauta físico. Para corrigir esse problema, foram adicionados os atributos "Transporte" e "Nro_Assento" à chave composta. No entanto, uma nova preocupação surgiu, pois agora é necessário garantir que, para uma mesma combinação de {Data, Hora, Transporte}, as informações {Origem, Hora_Saída, Destino, Hora_Chegada} sejam cadastradas de maneira consistente, evitando inconsistências. Nas etapas seguintes, será necessário implementar mecanismos de validação e controle para garantir a consistência dos dados, especialmente ao lidar com inserções, atualizações e exclusões. Além disso, deve-se assegurar que a relação entre {Data, Hora, Transporte} e {Origem, Hora_Saída, Destino, Hora_Chegada} seja mantida de maneira adequada para evitar inconsistências no sistema.

- **Inserção de justificativas do MRel:** todos os comentários de complemento às justificativas foram inseridos.

4) Implantação da base de dados e implementação do Sistema

4.1) Apresentação das consultas

1. **Consulta 1: Listar todas as cargas transportadas com informações completas:**

```
-- Consulta 1: Listar todas as cargas transportadas com informações completas:
SELECT Transporte.Cod_Transporte, C_Juridico.Nome AS Cliente, Carga_Destino.Planeta AS Destino, Transporte.Data, Transporte.Hora_Saida
FROM Transporte
JOIN C_Juridico ON Transporte.C_Juridico = C_Juridico.CCJ
JOIN Carga_Destino ON Transporte.Cod_Transporte = Carga_Destino.Transporte;
```

A consulta acima retorna informações relacionadas a transportes de cargas, incluindo o nome do cliente, o planeta de destino da carga, a data do transporte e a hora de saída. Para isso, foram necessárias duas junções (JOIN): o primeiro entre as tabelas “Transporte” e “C_Juridico”, usando a chave estrangeira *C_Juridico* presente na tabela “Transporte”, permitindo vincular informações sobre o cliente jurídico associado ao transporte, e o outro entre as tabelas “Transporte” e “Carga_Destino”, usando a chave estrangeira *Cod_Transporte* presente na tabela “Carga_Destino”, vinculando informações sobre o destino da carga associada ao transporte.

2. **Consulta 2: Calcular a média de idade dos passageiros de cada viagem:**

```
-- Consulta 2: Calcular a média de idade dos passageiros de cada viagem:
SELECT
    V.Data,
    V.Hora,
    V.Transporte,
    AVG(EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM CF.Nascimento)) AS Media_Idade_Passageiros
FROM
    Viagem V
JOIN
    C_Fisico CF ON V.C_Fisico = CF.CCF
GROUP BY
    V.Data, V.Hora, V.Transporte;
```

A consulta retorna informações sobre as viagens, incluindo a data, hora e transporte, juntamente com a média de idade dos passageiros para cada grupo de viagem, por meio da função AVG() e da cláusula “GROUP BY”..

3. **Consulta 3: Selecionar o planeta com o maior número de destinos (considerando transporte de viagem e de carga):**

```
SELECT Planeta, Qtd_Destinos
FROM (
    SELECT Planeta, COUNT(*) AS Qtd_Destinos
    FROM (
        SELECT Destino AS Planeta
        FROM Viagem
        UNION ALL
        SELECT Planeta
        FROM Carga_Destino
    )
    GROUP BY Planeta
)
ORDER BY Qtd_Destinos DESC
FETCH FIRST 1 ROW ONLY;
```

A subconsulta interna une os destinos das viagens da tabela “Viagem” e os destinos das cargas da tabela “Carga_Destino” usando a cláusula UNION ALL, permitindo a inclusão de duplicatas. Já a subconsulta agrupada, mais externa, conta a quantidade de ocorrências para cada planeta (considerando destinos de viagens e destinos de carga). A função COUNT(*) é utilizada para contar o número de ocorrências para cada planeta. Por fim, a consulta final, ordena os resultados pela quantidade de destinos em ordem decrescente (ORDER BY Qtd_Destinos DESC) e utilizamos a cláusula FETCH FIRST 1 ROW ONLY para limitar o resultado a apenas uma linha, ou seja, o planeta com o maior número de destinos.

4. **Consulta 4: Selecionar todas as espaçonaves da Astrix e, caso tenham realizado algum serviço, trazer nome do empregado, serviço, tipo, descrição e solução do problema:**

```
SELECT
    E.Nro_Serie,
    EM.Nome AS Nome_Empregado,
    S.Codigo AS Codigo_Servico,
    B.Tipo AS Tipo_Problema,
    B.Descricao,
    B.Solucao
FROM
    Espaconave E
LEFT JOIN
    Pilota P ON E.Nro_Serie = P.Espaconave
LEFT JOIN
    Servico S ON E.Nro_Serie = S.Espaconave
LEFT JOIN
    Problemas B ON S.Codigo = B.Servico
LEFT JOIN
    Empregado EM ON S.Empregado = EM.CF
Order By E.Nro_Serie;
```

A consulta retorna informações das espaçonaves da Astrix e, caso tenham realizado algum serviço, traz detalhes adicionais como nome do empregado, código do serviço, tipo, descrição e solução do problema associado ao serviço. A utilização de junções à esquerda permite que espaçonaves sem serviços associados também sejam incluídas no resultado. Além disso, foi utilizado o “Order by E.Nro_Serie” para ordenar o resultado pelo número de série da espaçonave em ordem crescente.

5. **Consulta 5: Selecionar todos os C.Fisicos que viajaram em todas as viagens:**

```sql
-- Consulta 5: Selecionar todos os C.Fisicos que viajaram em todas as viagens:
SELECT CCF, Nome
FROM C_Fisico CF
WHERE NOT EXISTS (
    SELECT DISTINCT V.Data, V.Hora, V.Transporte
    FROM Viagem V
    WHERE NOT EXISTS (
        SELECT 1
        FROM Viagem V2
        WHERE V2.C_Fisico = CF.CCF AND V2.Data = V.Data AND V2.Hora = V.Hora AND V2.Transporte = V.Transporte
    )
);
```

A consulta utiliza a lógica de divisão relacional para selecionar todos os C.Fisicos que participaram de todas as viagens disponíveis na tabela Viagem. A subconsulta interna mais interna retorna todas as datas, horas e transportes distintas de viagens da tabela Viagem (a utilização de DISTINCT garante que cada data seja considerada apenas uma vez). Já a consulta interna mais externa, para cada data, hora e transporte de viagem recuperada na subconsulta mais interna, verifica-se se existe pelo menos uma viagem (SELECT 1) para o mesmo C.Fisico (V2.C_Fisico = CF.CCF) na mesma data (V2.Data = V.Data), hora (V2.Hora = V.Hora) e transporte (V2.Transporte = V.Transporte). Por fim, na subconsulta externa, para cada C.Fisico (C_Fisico CF), verifica-se se não existe uma data, hora e transporte de viagem para a qual o C.Fisico não viajou. Isso é feito utilizando a cláusula NOT EXISTS e a subconsulta interna. Portanto, a subconsulta externa retorna todos os C.Fisicos que viajaram em todas as viagens, pois apenas aqueles para os quais não existe uma data de viagem não são incluídos no resultado final.

5) Aplicação

5.1) Descrição geral

A aplicação foi desenvolvida em Python 3, utilizando a biblioteca Psycopg2 para realizar a conexão e as operações na base de dados do Postgre. As funções implementadas são referentes à perspectiva de um usuário específico da base, i.e. Administrador, Piloto, Funcionário, Cosmonauta Físico ou Cosmonauta Jurídico, e requerem uma autenticação simples com o identificador de quem manipula a aplicação.

As operações foram todas feitas para serem usadas sob interface de linha de comando, ou seja, a aplicação é inteiramente manipulada pelo terminal, como demonstrado abaixo.

```
connection = conectar()
if connection is not None:
    print("Conectado ao banco de dados!")

while True:
    print("Escolha o tipo de cargo:")
    print("1. Cosmonauta Físico (CF)")
    print("2. Cosmonauta Jurídico (CJ)")
    print("3. Piloto")
    print("4. Administrador")
    print("5. Sair")

    opcao = input("Escolha uma opção: ")
    if connection:
        cursor = connection.cursor()

        if opcao == "1":
            menu_cf(connection, cursor)
        elif opcao == "2":
            menu_cj(connection, cursor)
        elif opcao == "3":
            menu_piloto(connection, cursor)
        elif opcao == "4":
            menu_administrador(connection, cursor)
        elif opcao == "5":
            print("Saindo do programa.")
            cursor.close()
            connection.close()
            break
        else:
            print("Opção inválida. Tente novamente.")
```

Segue abaixo a função em Python que descreve a conexão com a base de dados, junto a seu próprio tratamento de erros.

```python
def conectar():
    try:
        connection = psycopg2.connect(os.environ.get('DB_PATH'))
        print("Conexão bem-sucedida!\n")
        return connection
    except psycopg2.Error as psy:
        print(f"Erro ao conectar ao banco de dados: {psy}")
        return None
    except Exception as e:
        print(f"Erro ao conectar ao banco de dados: {e}")
        return None
```

5.2) Funções de cadastro, edição, remoção e busca de tuplas específicas

Abaixo estão descritas funções pertinentes ao administrador, porém a lógica pode ser estendida ao registro de pilotos, funcionários, espaçonaves, planetas, cosmonautas físicos e jurídicos.

5.2.1 Cadastrar

Assumindo que exista ao menos um usuário administrador criado através do próprio banco, o programa permite que um administrador cadastre outro durante as telas de operações. Caso o código de funcionário para cadastro já exista, o programa retorna uma mensagem na tela do terminal, do contrário o processo é continuado, requisitando o nome da pessoa, nome do planeta de origem, coordenadas e número do administrador.

```python
    cf_adm = input("código do funcionário: ")
    if administrador_existe(cursor, cf_adm):
        print("Código já existe!")
        return
    nome = input("Nome do funcionário: ")
    planeta = input("Planeta do administrador: ")
    cg = input("Coordenada galática: ")
    nro = int(input("número do administrador: "))
    comando = 'INSERT INTO ADMINISTRADOR (cf, nome, planeta, cg, nro) VALUES (%s, %s, %s, %s, %s)'
    dados = (cf_adm, nome, planeta, cg, nro)
    cursor.execute(comando, dados)
    connection.commit()
    print("Administrador criado com sucesso!\n")
```

5.2.2 Buscar

A função recebe um número de funcionário e tenta resgatar a entrada pertinente na tabela de administradores, retornando uma mensagem de erro caso nenhum seja encontrado.

```
busca = input("Insira o código do funcionário a buscar(Enter para busca geral): ")
if busca != "":
 comando = "SELECT * FROM ADMINISTRADOR WHERE cf=%s"
 cursor.execute(comando, busca)
else:
    comando2 = "SELECT * FROM ADMINISTRADOR"
    cursor.execute(comando2)

resultados = cursor.fetchall()

if resultados:
    for resultado in resultados:
        print(resultado)
else:
    print("Nenhum administrador encontrado.\n")
```

5.2.3 Editar

A função permite a um usuário administrador autenticado editar qualquer campo de uma tupla da tabela de administrador a partir do código de funcionário “CF” da tupla alvo.

```
cf_adm = input("Digite seu código de administrador: ")
campos_para_atualizar = []
cf = input("Novo código do administrador (pressione Enter para manter o mesmo): ")
if cf:
    campos_para_atualizar.append(f"cf = '{cf}'")

nome = input("Nome do administrador (pressione Enter para manter o mesmo): ")
if nome:
    campos_para_atualizar.append(f"nome = '{nome}'")

planeta_adm = input("Novo Planeta do administrador (pressione Enter para manter o mesmo): ")
if planeta_adm:
    campos_para_atualizar.append(f"planeta = '{planeta_adm}'")
cg = input("Nova coordenada galáctica (pressione Enter para manter o mesmo): ")
if cg:
    campos_para_atualizar.append(f"cg = '{cg}'")

nro = input("Novo número do administrador (pressione Enter para manter o mesmo): ")
if nro:
    campos_para_atualizar.append(f"nro = {nro}")

# Construir a consulta de atualização
if campos_para_atualizar:
    comando = f'''
        UPDATE ADMINISTRADOR
        SET {', '.join(campos_para_atualizar)}
        WHERE cf = '{cf_adm}'
    '''
    cursor.execute(comando)
    connection.commit()
    print("Administrador editado com sucesso!\n")
else:
    print("Nenhum campo para atualizar.\n")
```

5.2.4 Remover

Utilizando o código de funcionário, é possível deletar uma tupla específica da tabela de administrador.

```
deleta = input("Digite a chave do administrador a ser deletado: ")
if deleta:
    comando = 'DELETE FROM ADMINISTRADOR WHERE cf = %s'
    try:
        cursor.execute(comando, (deleta,))
        connection.commit()
        print("Administrador deletado com sucesso!\n")
    except Exception as e:
        print(f"Erro ao deletar administrador: {e}")
else:
    print("Chave não existe. Operação encerrada!")
    return
```

5.3) Funções referentes a transporte e viagens

5.3.1 Verificar transporte

Assumindo a existência de um piloto operando o aplicativo, a função transporte recebe o código do transporte e retorna todas as informações disponíveis.

```
def buscar_transporte(connection, cursor):
    try:
        if not check_null(cursor, "piloto"):
             cp_piloto = input("Digite o código do Piloto: ")
             if not piloto_existe(cursor, cp_piloto):
                print("Piloto não encontrado. Acesso não autorizado.")
                return

        busca = input("Código do transporte: ")
        comando = "SELECT * FROM transporte WHERE cod_transporte=%s"
        cursor.execute(comando, busca)
        resultados = cursor.fetchall()

        if resultados:
            for resultado in resultados:
                print(resultado)
        else:
            print("Tipo de transporte não identificado!\n")

    except Exception as e:
        print(f"Erro ao executar a busca: {e}")
```

5.3.2 Verificar viagem (Piloto)

Assumindo a autenticação de um piloto, a aplicação exige data, hora e nome de um cosmonauta físico para executar a busca completa das informações pertinentes ao viajante.

```
def buscar_viagem(connection, cursor):
    try:
        if not check_null(cursor, "piloto"):
           cp_piloto = input("Digite o código do Piloto: ")

           if not piloto_existe(cursor, cp_piloto):
                print("Piloto não encontrado. Acesso não autorizado.")
                return

        data = input("Data da viagem(YYYY-MM-dd): ")
        hora = input("Hora da viagem: ")
        c_fisico = input(input("Código do cosmonauta: "))
        comando = "SELECT * FROM viagem WHERE data=%s AND hora=%s AND c_fisico=%s"
        dados = (data, hora, c_fisico)
        cursor.execute(comando, dados)
        resultados = cursor.fetchall()

        if resultados:
            for resultado in resultados:
                print(resultado)
        else:
            print("Nenhuma viagem realizada com essas especificações!.\n")

    except Exception as e:
        print(f"Erro ao executar a busca: {e}")
```

5.3.3 Verificar viagem (Cosmonauta)

Um cosmonauta é permitido verificar as informações de sua viagem sabendo o modelo de sua espaçonave (ele não precisa digitar o modelo, apenas escolher, como mostrado abaixo)

```
print("1. Criar/Buscar/Deletar/Atualizar Cosmonauta Físico")
print("2. Buscar Planeta")
print("3. Editar Viagens da Nave A330")
print("4. Deletar Viagens da Nave A770")
print("5. Consultar dados do piloto")
print("6. Consultar dados das naves")
print("7. Consultar Valor da viagem")
print("8. Voltar")
opcao = input("Escolha a operação:")
```

```python
def buscar_viagens_a330(connection, cursor):
    try:
        if not check_null(cursor, "c_fisico"):
            ccf_codigo = int(input("Digite o código do Cosmonauta Físico: "))
            if not cosmonauta_fisico_existe(cursor, ccf_codigo):
                print("Cosmonauta Físico não encontrado. Acesso não autorizado.")
                return

        comando = "SELECT * FROM viagem WHERE transporte=A330"
        cursor.execute(comando)
        resultados = cursor.fetchall()

        if resultados:
            for resultado in resultados:
                print(resultado)
        else:
            print("Nenhuma viagem encontrada para a espaçonave A330.\n")

    except Exception as e:
        print(f"Erro ao executar a busca: {e}")
```

5.3.4 Consultar preço da viagem (Cosmonauta)

Essa operação requer o nome do cosmonauta, sua origem e seu destino para realizar a busca do valor de sua passagem. O preço de transporte do cosmonauta jurídico segue uma lógica análoga à mostrada a seguir.

```
def consultar_dados_transporte_valor_destino(connection, cursor):
    try:
        if not check_null(cursor, "c_fisico"):
            ccf_codigo = int(input("Digite o código do Cosmonauta Físico: "))
            if not cosmonauta_fisico_existe(cursor, ccf_codigo):
                print("Cosmonauta Físico não encontrado. Acesso não autorizado.")
                return

        origem = input("Digite o nome do planeta de origem: ")
        destino = input("Digite o nome do planeta de destino: ")

        cursor.execute("""
            SELECT t.*, v.Origem, v.Destino,
                   CASE WHEN e.Tipo = 'TRANSPORTE' THEN e.Modelo
                        WHEN e.Tipo = 'CARGA' THEN 'N/A'
                   END AS Tipo_Espaconave,
                   CASE WHEN e.Tipo = 'TRANSPORTE' THEN et.Qnt_Assentos
                        WHEN e.Tipo = 'CARGA' THEN c.Capacidade
                   END AS Capacidade,
                   CASE WHEN e.Tipo = 'TRANSPORTE' THEN calcular_valor_transporte(v.Transporte)
                        WHEN e.Tipo = 'CARGA' THEN calcular_valor_carga(v.Transporte)
                   END AS Valor_Viagem
            FROM Transporte t
            JOIN Viagem v ON t.Cod_Transporte = v.Transporte
            JOIN Espaconave e ON t.Carga = e.Nro_Serie OR t.C_Juridico = e.Nro_Serie
            LEFT JOIN E_TRANSPORTE et ON e.Nro_Serie = et.Espaconave
            LEFT JOIN Carga c ON e.Nro_Serie = c.Espaconave
            JOIN C_Fisico cf ON v.C_Fisico = cf.CCF
            WHERE v.Origem = %s AND v.Destino = %s AND cf.CCF = %s
        """, (origem, destino, ccf_codigo))

        resultados = cursor.fetchall()

        if resultados:
            for resultado in resultados:
                print(resultado)
        else:
            print("Nenhum Transporte encontrado para os planetas informados.\n")

    except Exception as e:
        print(f"Erro ao executar a consulta: {e}")
```

5.4) Tratamento de exceções internas

Funções básicas de controle foram implementadas para discriminar eventuais erros na aplicação relacionados à existência dos operadores e dos registros na base de dados, imprimindo na tela uma notificação de falha.

```
def administrador_existe(cursor, cf_admin):
    try:
        cursor.execute("SELECT COUNT(*) FROM Administrador WHERE CF = %s", (cf_admin,))
        count = cursor.fetchone()[0]
        return count > 0
    except Exception as e:
        print(f"Erro ao verificar a existência do administrador: {e}")
        return False
```

```
def check_null(cursor, table_name):
        cursor.execute(f"SELECT COUNT(*) FROM {table_name}")
        count = cursor.fetchone()[0]
        if count == 0:
            return True
        return False
```

6) Conclusão

Levando-se em conta o decorrer da disciplina, o grupo considera o projeto e a matéria de Bases de Dados desafiadora, mas ao mesmo tempo, muito enriquecedora. Desde as aulas, aos atendimentos de monitoria, práticas de laboratório e ao desenvolvimento do projeto.

Sobre as aulas, a docente possui uma ótima didática, ao qual contribui, e muito, no entendimento dos alunos. Entretanto, muitas aulas apresentam uma alta carga de conteúdo e conceitos, sendo difícil compreender de imediato. Ademais, os laboratórios poderiam ocorrer desde o início das aulas de SQL, já que na prática é mais fácil de aprender.

Sobre o projeto, o grupo entende como necessário para fixação e aprendizado de todos os conceitos apresentados em sala de aula. Além disso, os atendimentos com o monitor são essenciais para o desenvolvimento e sucesso do trabalho.

Sobre as avaliações, o conteúdo cobrado foi totalmente coerente, mas a falta de tempo dificultava a execução dela, assim como realizar a prova de programação no papel.

De modo geral, o grupo se sente muito satisfeito com a realização da matéria e apto a exercer atividades ligadas a Bases de Dados.