

**UNIVERSIDADE DE SÃO PAULO**  
**Escola de Engenharia de São Carlos**  
**Departamento de Engenharia Elétrica e de Computação**

SCC0640 - BASES DE DADOS

---

**Enciclopédia para Exploração Espacial**

---

ENGENHARIA DE COMPUTAÇÃO

Docente: Prof<sup>a</sup> Dr<sup>a</sup> Elaine Parros Machado de Sousa

Estagiário PAE: André Moreira Souza

<b>Aluno</b>	<b>NºUSP</b>
Alexandre Lopes Ferreira Dias dos Santos	11801199
Fernando Rosalini Calaza	11231965
Luigi Quaglio	11800563
Gabriel de Avelar Las Casas Rebelo	11800462

São Carlos, SP

10 de dezembro de 2023

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Especificação do problema</b>	<b>1</b>
<b>3</b>	<b>Diagrama</b>	<b>3</b>
<b>4</b>	<b>Entidades Fortes</b>	<b>4</b>
4.1	Estrelas . . . . .	4
4.2	Planetas . . . . .	4
4.3	Astronauta . . . . .	5
4.4	Bioma . . . . .	5
4.5	Minério . . . . .	6
4.6	Elemento Químico . . . . .	6
4.7	Fabricáveis . . . . .	6
4.7.1	Equipamentos . . . . .	6
4.7.2	Construções . . . . .	6
4.7.3	Nave . . . . .	6
4.8	Vendedor . . . . .	7
<b>5</b>	<b>Entidades Fracas</b>	<b>7</b>
5.1	Espécie . . . . .	7
<b>6</b>	<b>Relacionamentos</b>	<b>7</b>
6.1	Contém . . . . .	7
6.2	Ambiente . . . . .	7
6.2.1	Tem . . . . .	8
6.3	Possui . . . . .	8
6.4	Possui (2) . . . . .	8
6.5	Geram . . . . .	8
6.6	Geram(2) . . . . .	9
6.7	Aprimora . . . . .	9

6.8	Cria . . . . .	9
6.9	Vende . . . . .	9
6.10	Vende(2) . . . . .	9
6.11	Viagem . . . . .	10
6.11.1	Percorre . . . . .	10
6.12	Rota . . . . .	10
6.12.1	Viaja . . . . .	10
6.13	Descobre . . . . .	10
6.14	Descobre (2) . . . . .	11
6.15	Mercado . . . . .	11
6.15.1	Comercializa . . . . .	11
6.16	Local . . . . .	11
6.16.1	Constrói . . . . .	11
<b>7</b>	<b>Ciclos de redundância</b>	<b>12</b>
7.1	Planeta - Planeta . . . . .	12
<b>8</b>	<b>Ciclos de dependência</b>	<b>12</b>
8.1	Planeta - Astronauta . . . . .	12
8.2	Planeta - Astronauta (2) . . . . .	12
8.3	Espécies - Astronauta - Planeta . . . . .	12
<b>9</b>	<b>Alguns ciclos que não geram problemas</b>	<b>13</b>
<b>10</b>	<b>Mudanças realizadas entre a parte 1 e parte 2 no MER</b>	<b>13</b>
<b>11</b>	<b>Mapeamento</b>	<b>13</b>
11.1	Considerações iniciais . . . . .	15
11.2	Relacionamentos 1:N . . . . .	15
11.2.1	Espécies e Elemento Químico . . . . .	15
11.2.2	Minério e Elemento Químico . . . . .	15
11.3	Auto-relacionamentos . . . . .	16
11.3.1	Planeta e Planeta . . . . .	16
11.4	Generalização . . . . .	16
11.4.1	Fabricáveis -> Construções/Equipamentos/Nave . . . . .	16

11.5	Atributos Multivalorados . . . . .	16
11.6	Atributos Derivados . . . . .	17
11.6.1	Calor . . . . .	17
<b>12</b>	<b>Mudanças realizadas entre a parte 2 e parte 3 no MER</b>	<b>17</b>
<b>13</b>	<b>Implementação do Banco de Dados</b>	<b>17</b>
13.1	Visão geral . . . . .	17
13.2	<i>Scripts</i> de consulta . . . . .	18
13.3	Aplicação . . . . .	21
13.3.1	Descrição . . . . .	21
13.4	Bibliotecas utilizadas . . . . .	22
13.4.1	Inserção realizada . . . . .	22
13.4.2	Consulta implementada . . . . .	30
<b>14</b>	<b>Conclusão</b>	<b>32</b>
<b>15</b>	<b>Referências</b>	<b>33</b>

# 1 Introdução

A aplicação escolhida pelo grupo foi inspirada no jogo *No Man's Sky*, que é focado na exploração de planetas gerados aleatoriamente dentro de alguns padrões e, portanto, se encaixa perfeitamente no tema “Exploração de Planetas”.

Partindo disso, nosso objetivo é criar um sistema que serviria como uma enciclopédia, ou seja, um catálogo dos sistemas de dados e suas relações.

O sistema deve auxiliar o jogador em seu progresso dentro do jogo com informações a respeito da exploração de recursos, suas viagens interplanetárias, localização de construções, entre outras. Como o jogo é um tipo de MMO(*Massive Multiplayer Online*), é importantíssimo lembrar que a aplicação deve ser escalável de modo que o banco de dados seja compartilhado por todos os jogadores. Para que, assim, as mudanças feitas no mundo por alguns reflitam na experiência de todos.

## 2 Especificação do problema

Neste jogo cada **planeta**, indicado pelo nome, pode ser descoberto pelos jogadores. Foi considerado que eles possuem um tamanho próprio, uma velocidade, composta pelos valores de translação e rotação, além de um nível de segurança e número de luas. Apesar de poderem ser acessados, os satélites naturais, configuram uma coordenada especial, portanto, não são uma entidade. Os **planetas** têm, também, uma distância média da **estrela** a qual ele necessariamente deve orbitar e, por conseguinte, define sua órbita. Somado a isso, de acordo com a implementação do jogo, o calor de um **planeta** é calculado a partir desta distância e do tamanho da **estrela** cujo **planeta** está relacionado. Assim, é necessário que um **planeta** pertença a um sistema com um astro central.

Cada estrela tem um nome único no sistema, além de atributos como tamanho, que pode variar de subanãs até supergigantes, além de um *booleano* chamado binário, o qual definirá se existe um par de estrela em órbita entre si, o que forma o chamado sistema binário.

Pensando na exploração interplanetária, o jogo é feito de forma que exista apenas um **bioma** por **planeta**. Estes têm um tipo pré-definido que indica o nível de radiação e toxicidade. A partir disso, as informações contidas nos **planetas** (calor, tamanho, etc) relacionadas com o **bioma**, encontrado no mesmo, configuram o **ambiente** do **planeta**.

Já olhando para este **ambiente**, ele possuem um atributo descrevendo o tipo fauna/flora o

qual serve como lista de restrições para que um ambiente possua determinada entidade **espécies**. O mesmo ocorre para o tipo de **minérios**. A partir desta lista é necessário garantir que todas as **espécies** e **minérios** possuídos pelo **ambiente** tenham tipos que estejam contidos nessa lista. Apesar de parecer redundante, a lista representa limitações e, portanto, devem ser armazenadas no ambiente. Outrossim, cada animal tem seu próprio tipo que deve estar contido em cada entidade. Por fim, o ambiente terá, também, uma lista de perigos que ajudará o astronauta a saber das adversidades que ele pode encontrar em solo desconhecido.

Toda espécie deve necessariamente pertencer a um ambiente. Elas são categorizadas tanto pelo ambiente em que estão inseridos quanto por um nome ou apelido único para si. Além disso, devem possuir um tipo e natureza geral. O minério por sua vez possui um nome científico que serve como forma de catalogá-lo, além de apresentar também um tipo e dureza.

Um **astronauta**, é catalogado a partir de seu nome ou apelido, e deve possuir também uma data de nascimento. Ele é capaz de descobrir **planetas** que não tenha visitado ainda, sendo que estas informações poderão ser acessadas numa espécie de diário de bordo baseado na aplicação. Ao realizar um deslocamento de um planeta para outro pela primeira vez, cria-se uma rota, a qual é definida pela sua origem e destino. Essa rota terá guardada em si a distância total. Uma rota interplanetária permite que as descobertas sejam feitas pelo jogador, porém, atrelado à data e hora em que o caminho foi percorrido, analogamente caracteriza uma viagem. Deste modo, uma pessoa inicia sua jornada em um **planeta** (primeira descoberta) e pode viajar para outro, descobrindo-o assim que pousar sua nave, este segundo deverá ser a origem de sua próxima viagem. Em outras palavras, o ponto de partida deve ter sido o destino da última travessia interplanetária e já ter sido descoberto. É importante lembrar que se o destino for igual à origem, não há necessidade de registrar no diário de bordo. Por fim, a data e hora da viagem deve ser anotada para que um jogador possa observar quantas vezes fez uma mesma rota espacial.

Semelhantemente, as **espécies** de animais e plantas também podem ser descobertas desde que o **planeta** em que habitam já tenha sido visitado. Os jogadores, assim que as descobrirem, têm a oportunidade de nomeá-los da maneira que preferirem. Com isso, deve-se limitar espécies diferentes com o mesmo nome. Esta restrição acontece pois cada uma delas só pode ser encontrada em um único **planeta**. Este relacionamento é importante para o progresso do jogo já que tanto estes seres vivos quanto os **minérios** geram **elementos químicos** que são essenciais para a evolução do **astronauta**.

Estes elementos, definidos pela sua sigla encontrada na tabela periódica, são responsáveis

tanto pela criação quanto pelo aprimoramento de **fabricáveis**, os quais são divididos entre **naves**, **equipamentos** e **construções**. Assim, estas melhorias e fabricações devem ser armazenadas em forma de receitas para que possam ser consultadas. Além disso, alguns destes **fabricáveis**, apesar de poderem ser aprimorados, só podem ser adquiridos através de vendas. O comércio de **fabricáveis** e **elementos químicos** é feito por um **vendedor**. Um mercado será estabelecido a partir da união entre um vendedor o qual terá um nome único, um comprador(astronauta) em uma dada hora e dia.

Já os fabricáveis são definidos pelo seu nome e devem possuir durabilidade, nível, o qual especifica sua qualidade e/ou raridade em geral, além de um tipo que atrela uma utilidade a ele.

Dentre os três possíveis tipos cada um possui os seguintes atributos:


As construções terão uma funcionalidade, ou seja, habitação, armazenamento, dentre outros; Equipamento terá uma potência, presumindo que eles serão usados primariamente para transporte e mineração; Nave terá uma capacidade de carga máxima e velocidade limite.

Pensando na esfera de jogabilidade em contraste às receitas e pesquisas, foi evidenciado que faria sentido um **astronauta** conseguir consultar as posições de suas **construções**. Deste modo, um astronauta constrói algo em uma posição(x,y,z) que será traduzida em um mapa para o jogador. Vale lembrar também que estas instâncias de localização não representam a funcionalidade de construir em si, ou seja, não há a necessidade de checar o inventário do astronauta já que se trata apenas da localização em que aparecera no espaço do jogo.

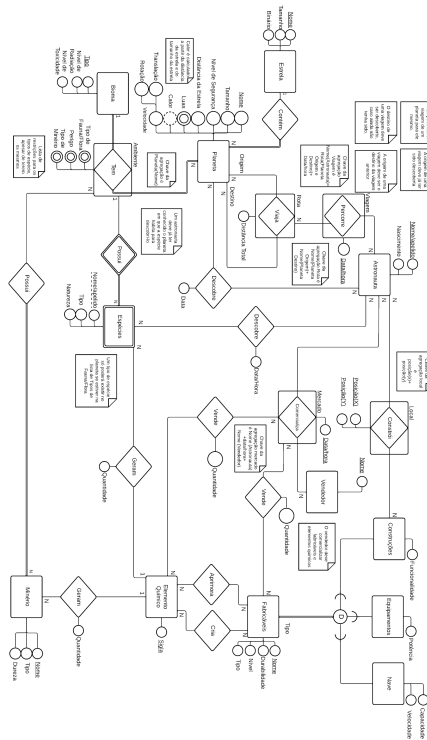
### 3 Diagrama

Para representação gráfica das entidades e relacionamentos entre si que temos discutido até então, foi utilizada a notação MER (Modelo entidade relacionamento).

A figura 1 representa o produto final obtido pelo grupo, acompanhado de notações auxiliares que esclarecem possíveis ambiguidades, além de destacarem pontos relevantes de se manter em mente para etapas futuras. Nessa tarefa, tomou-se como ferramenta, o *software* de diagramação em web *lucidchart*.

Link para visão do diagrama *lucidchart* 

**Figura 1:** Diagrama MER criado para modelagem do banco de dados



## 4 Entidades Fortes

## 4.1 Estrelas

- Nome [**Chave principal**]: característica individual do corpo celeste e sua forma de busca;
- Tamanho: estabelece a categoria em uma escala predeterminada [pequena, média, grande];
- Binário: revela se o sistema possui duas estrelas em seu centro ao invés de apenas uma [sim ou não].

## 4.2 Planetas

- Nome [**chave principal**]: característica individual do corpo celeste e sua forma de busca;
- Tamanho: estabelece a categoria em uma escala predeterminada [pequeno, médio, grande];
- Nível de Segurança: escala referente à presença de robôs guardiões ao longo do local [sem segurança (sem sentinelas), com segurança (sentinelas atacam o astronauta se ele



destruir o ambiente), com muita segurança (sentinelas atacam o astronauta assim que o encontram)];

- Distância da Estrela: será calculada como a média da órbita feita em torno da estrela [valor numérico];
- Calor [**Atributo calculado**]: é definido com base no tamanho e distância média do planeta para a estrela [valor numérico];
- Velocidade: será utilizado como um composto de dois valores para o acompanhamento da distância de um jogador até a posição das construções em um local do planeta [**Atributo composto**];
  - Rotação: armazena a velocidade do corpo para uma revolução completa em torno de si [valor numérico];
  - Translação: armazena a velocidade do corpo para uma revolução completa em torno da estrela [valor numérico];
- Luas [**multivalorado**]: um planeta pode apresentar um ou mais satélites naturais.

### 4.3 Astronauta

Esse será o avatar controlado pelo jogador ao interagir efetivamente com o produto final.

- Nome/apelido [**chave principal**]: característica individual do jogador e sua forma de ;busca no sistema;
- Nascimento: seu local de origem, ou seja, o primeiro local de aparecimento no jogo.

### 4.4 Bioma

- Tipo [**chave principal**]: categoriza o bioma dentre certas características comuns entre si [Exemplo: Floresta Tropical, Savana etc];
- Nível de Radiação: escala definindo a presença de isótopos radioativos no ar [valor numérico];
- Nível de Toxicidade: escala definindo a presença de elementos nocivos no ar [valor numérico].

## 4.5 Minério

- Nome [**chave principal**]: nome científico para pesquisa dos recursos geológicos;
- Tipo: agrupamento a qual pertence o minério [Orgânico, metálico e etc];
- Dureza: escala de resistência á tensão [valor numérico].

## 4.6 Elemento Químico

- Sigla [**chave principal**]: correspondência na tabela periódica, utilizada para busca.

## 4.7 Fabricáveis

- Nome [**chave principal**]: Identificador do item [valor numérico];
- Durabilidade: quão resistente é ao estresse físico [forte, médio e fraco];
- Nível: raridade do item construído [S, A, B, C e D].
- Tipo: define a utilidade do fabricável, a qual pode pertencer a 3 categorias distintas, listadas nas subseções seguintes.

Esta entidade é obrigatoriamente uma das seguintes opções, além disso, se trata de uma disjunção, ou seja, apenas uma.

### 4.7.1 Equipamentos

- Potência: requerimento de energia para funcionamento [valor numérico].

### 4.7.2 Construções

- Funcionalidade: utilidades da construção atrelada em sua criação [hospital, casa, armazém etc].

### 4.7.3 Nave

- Capacidade: volume disponível para transporte de carga [valor numérico];
- Velocidade: valor máximo atingido pela espaço-nave durante voo [valor numérico].

## 4.8 Vendedor

- Nome [**Chave principal**]: característica individual do mercador e sua forma de busca no sistema;

# 5 Entidades Fracas

## 5.1 Espécie

É atrelada ao ambiente (bioma + planeta) no qual a entidade está inserida. Dessa forma, sua chave principal de identificação será a chave da agregação, nome do planeta, somado ao nome da espécie.

- Nome/Apelido [**chave parcial**]: Nome dado a espécie que o astronauta tem a oportunidade de escolher, se ele decidir não escolher o sistema gera um nome aleatório para a espécie;
- Tipo: Família da espécie [Felino];
- Natureza: determina o quão agressiva é a espécie [pacífica, neutra, agressiva].

# 6 Relacionamentos

## 6.1 Contém

Estabelece uma dependência entre certas entidades. São elas: Estrela e Planeta

Como é um relacionamento com participação obrigatória, todo planeta deve, necessariamente, orbitar em torno de uma estrela. Entretanto, não é necessário que uma estrela tenha um planeta orbitando em torno de si. Como a relação é 1:N, isso garante efetivamente que podem haver um número indeterminado de planetas em um sistema.

## 6.2 Ambiente

Agregação composta pela relação entre um planeta e seu bioma. Sua chave de agregação será o atributo nome do Planeta.

### 6.2.1 Tem

Como a relação é N:1, a relação estabelece um planeta contém, única e exclusivamente, um bioma, o qual pode se repetir em outros locais. A relação de pertencimento define os seguintes atributos:

- Tipo de Fauna/Flora [**multivalorado**]:
- Perigo [**multivalorado**]:
- Tipo de Minério [**multivalorado**]:

Pelo relacionamento com participação obrigatória, todo planeta terá um bioma, necessariamente.

## 6.3 Possui

Estabelece a relação entre Ambiente e Espécies. Como a relação é 1:N, isso garante que um ambiente possui um número indefinido de espécies. Note que, a participação obrigatória define que, toda espécie deve estar contida em um ambiente. Como a espécie é uma entidade fraca, diferentemente do que ocorre com minério, permite-se que haja espécies distintas com nomes iguais em planetas diferentes, visto que elas terão ainda o identificador de sua origem [Exemplo: macaco de marte, macaco da terra].

## 6.4 Possui (2)

Estabelece a relação entre Ambiente e Minério. Como a relação é N:N, isso garante que um ambiente possui um número indefinido de minérios e que os minérios podem ser encontrados em diversos planetas. Entretanto, um minério deve estar obrigatoriamente incluso em um planeta.

## 6.5 Geram

Estabelece a relação entre Espécie e Elemento Químico. Como a relação é N:1, isso garante que uma dada espécie após ser abatida (sendo ela um animal, planta, fungo etc) pelo jogador irá gerar apenas um elemento, apesar de um elemento poder ser gerado por espécies distintas.

- Quantidade: Cada espécie ao ser morta ou coletada gera uma quantidade de elemento químico.

## **6.6 Geram(2)**

Estabelece a relação entre Minério e Elemento Químico. Como a relação é N:1, isso garante que uma dada rocha ao ser minerada deve gerar apenas um tipo de químico, entretanto, um elemento pode ser obtido por diferentes minérios.

- Quantidade: Cada minério ao ser coletado gera uma quantidade de elemento químico.

## **6.7 Aprimora**

Estabelece a relação entre Elemento Químico e Fabricáveis. Sendo ela N:N, assim o melhoramento de um item pode ser feito a partir de um número indefinido de elementos, assim como um elemento pode contribuir na melhoria de itens distintos.

## **6.8 Cria**

Estabelece a relação entre Elemento Químico e Fabricáveis. Sendo ela N:N, assim a receita de um item pode ser feita a partir de um número indefinido de elementos, da mesma forma que um elemento pode contribuir para a receita de itens distintos.

## **6.9 Vende**

Estabelece a relação entre Mercado e Elemento Químico. Como a relação é N:N, isso garante que vários comércios podem vender o mesmo elemento e também que um material possa ser vendido em diversos mercados.

- Quantidade: numero de itens vendidos na transação.

## **6.10 Vende(2)**

Estabelece a relação entre Mercado e Fabricáveis. Como a relação é N:N, isso garante que vários comércios podem vender os mesmos fabricáveis, e também que um fabricável possa ser vendido em diversos mercados.

- Quantidade: numero de itens vendidos na transação.

## **6.11 Viagem**

Agregação composta pela rota tomada em um determinado momento por um astronauta.

### **6.11.1 Percorre**

Determina a movimentação do jogador por uma rota. Sendo ela N:N, é garantida a possibilidade de uma rota ser percorrida por inúmeros jogadores, assim como um deles pode percorrer mais de uma rota. A chave da agregação será composta pelo nome do jogador os nomes dos planetas de origem e destino, tal como a data e hora do deslocamento de forma que uma rota possa ser percorrida em diferentes momentos.

- Data/Hora: registra o tempo no relógio global de chegada ao destino [valor numérico].

## **6.12 Rota**

Agregação composta pelos possíveis caminhos estabelecidos no jogo de movimentação entre planetas.

### **6.12.1 Viaja**

Estabelece a relação entre origem e destino. Sendo ela N:N, é garantido a possibilidade de sair de um dado planeta fixado para inúmeros possíveis destinos. A chave da agregação será composta pelos nomes dos planetas de origem e destino.

- Distância Total: medida de espaço que deve ser percorrido para alcançar um destino a partir de uma dada origem [valor numérico].

## **6.13 Descobre**

Estabelece a relação entre Astronauta e Planeta. Sendo ela N:N, tanto um dado jogador pode descobrir múltiplos planetas, como um mesmo planeta pode ser descoberto por vários jogadores que não trocam informações entre si.

- Data/Hora: registra o tempo no relógio global em que o viajante chegou no seu destino não explorado pela primeira vez [valor numérico].

## 6.14 Descobre (2)

Estabelece a relação entre Astronauta e Espécie. Sendo ela N:N, tanto um dado jogador pode descobrir múltiplas espécies, como uma mesma espécie pode ser descobertas por vários jogadores que não trocam informações entre si.

- Data/Hora: registra o tempo no relógio global em que o viajante avista o ser vivo não conhecido pela primeira vez [valor numérico].

Vale notar que o planeta habitado pela espécie já deve ter sido descoberto previamente pelo astronauta para que ele tenha contato com uma nova espécie.

## 6.15 Mercado

Agregação composta pela transação de itens entre Astronauta e Vendedor. A chave da agregação será uma composta pelos argumentos nome tanto de Astronauta, quanto de Vendedor, além da data em que ocorre a interação, para que um jogador possa interagir com o vendedor múltiplas vezes em horários diferentes.

### 6.15.1 Comercializa

Denota a relação comercial estabelecida. Sendo a relação N:N garante que uma transação comercial pode ser feita por vários astronautas com um comerciante, da mesma forma que um jogador pode interagir com múltiplos comerciantes.

- Data/Hora: registra a hora exata em que ocorreu a venda.

## 6.16 Local

Agregação composta pela união de Astronauta com as Construções que ele escolhe instanciar em um dado ponto do mapa espacial. Para a chave da agregação, usa-se apenas as coordenadas tridimensionais no mapa do jogo, já que temos que garantir que uma construção não pode ser feita no mesmo lugar por múltiplos jogadores.

### 6.16.1 Constrói

Relação de criação entre os jogadores e as construções. A relação N:N garante que uma Construção pode ser feita por vários astronautas, da mesma forma que um jogador pode gerar

múltiplos construtos.

- Posição (X,Y,Z): posição x,y,z do jogo em si.

## **7 Ciclos de redundância**

### **7.1 Planeta - Planeta**

Para configurar uma viagem deve-se sair de um planeta e chegar em um outro planeta diferente do primeiro. O modelo não garante que o planeta de origem seja diferente do de destino. Esta restrição deve ser feita pela própria aplicação com o intuito de evitar o armazenamento desnecessário de dados.

## **8 Ciclos de dependência**

### **8.1 Planeta - Astronauta**

O astronauta tem dois relacionamentos com planetas que são dependentes entre si. Ao registrar uma viagem de um planeta a outro, a origem deve já ter sido descoberta pelo astronauta anteriormente. Ademais, o astronauta deve descobrir o planeta de destino caso ainda não tenha ido até lá. Deste modo, a aplicação deverá garantir que a origem já tenha sido descoberta e checar o possível descobrimento de novos planetas.

### **8.2 Planeta - Astronauta (2)**

A origem de uma viagem deve ser o destino da viagem anterior, com exceção da primeira viagem cuja origem será o planeta de nascimento do jogador. A aplicação deve garantir esta restrição no armazenamento de dados.

### **8.3 Espécies - Astronauta - Planeta**

Para que um astronauta descubra uma espécie ele necessariamente deve ter descoberto o planeta em que esta espécie habita. Tendo em vista que as espécies só existem em um planeta específico. Isto pode gerar inconsistências que devem ser tratadas na aplicação.



## **9 Alguns ciclos que não geram problemas**

Os ciclos listados abaixo não ocasionam inconveniências para a consistência de dados na aplicação.

- Astronautas - Espécies - Elemento químico.
- Elemento químico - Fabricáveis.
- Elemento químico - Fabricáveis - Mercado.

## **10 Mudanças realizadas entre a parte 1 e parte 2 no MER**

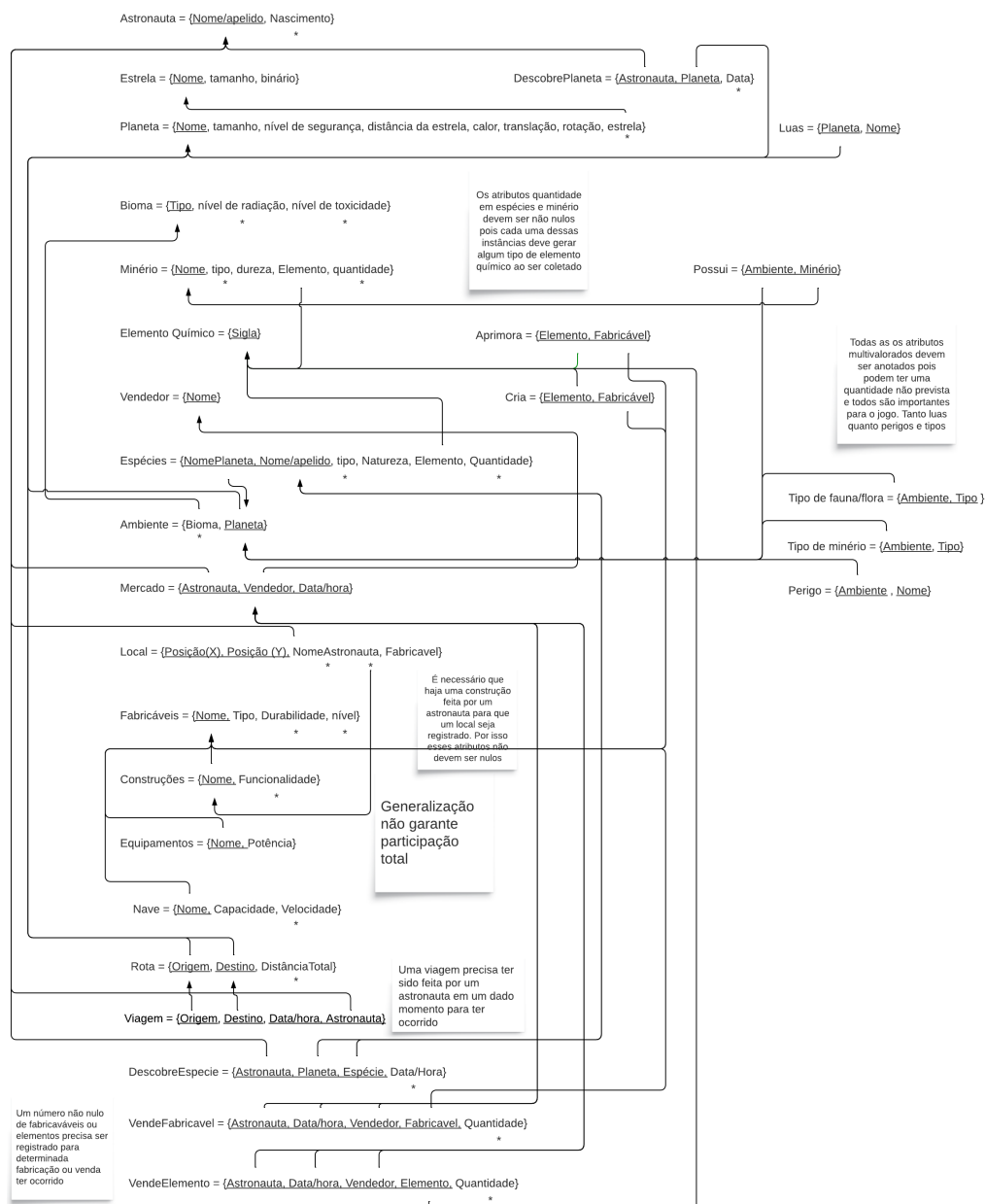
Adaptou-se o modelo de acordo com os erros observados. Foram elas:

- A criação de uma nova agregação Viagem com chave composta pelo Astronauta viajante, a Rota percorrida por ele e a data/hora. Esta entidade guardará a movimentação feita pelo jogador de um planeta origem ao destino em um dado instante de tempo. Isso permite que ele percorra a mesma Rota mais de uma vez;
- Modificou-se a agregação Rota, trocando seu atributo por uma distância total entre os planetas;
- Foi retirado o atributo catálogo da entidade Vendedor, pois o seu funcionamento não estava sendo garantido graças a possíveis redundâncias. Em lugar disso será colocada uma nota para esclarecer a forma desejada de implementação em etapas seguintes do projeto.
- Foi adicionado o atributo Data na relação Descobre entre Planeta e Astronauta
- Foi acrescentado o atributo Tipo na entidade Fabricáveis o qual estabelece a funcionalidade do item, podendo ser uma Construção, Equipamento ou Nave.

## **11 Mapeamento**

O diagrama gerado está elencado na figura 2.

**Figura 2:** Diagrama Modelo Relacional para modelagem do banco de dados



Nessa tarefa, tomou-se como ferramenta, o *software* de diagramação em web *lucidchart*. Abaixo tem-se um link para o projeto contendo o diagrama e notações auxiliares que esclarecem as decisões de projeto tomadas

Link para uma melhor visualização do diagrama MR *lucidchart*: [🔗](#)

Na etapa seguinte do projeto foi elaborado o Modelo Relacional(MR). O mapeamento segue regras gerais, porém, em certas instâncias, a estratégia de conversão do MER para o novo modelo varia de acordo com o contexto e abordagem do projetista. Dessa forma, discorre-se

abaixo a abordagem tomada em certas etapas abertas para interpretação, assim como um breve sumário de suas vantagens e desvantagens.

## 11.1 Considerações iniciais

As agregações e relacionamentos N:N foram mapeados criando uma nova tabela para facilitar as buscas já que a maioria têm atributos próprios ou apresentam repetição de pares do relacionamento. Além disso, as entidades fracas também foram mapeadas criando-se uma tabelas adicionais como de costume.

## 11.2 Relacionamentos 1:N

### 11.2.1 Espécies e Elemento Químico

**Solução adotada:** Para esta solução o relacionamento foi mapeado para dentro de espécies. Assim, um dos atributos de espécie faz o relacionamento ou recebe *NULL* se não participar do relacionamento.

**Vantagens:** Não criamos uma tabela a mais para o relacionamento evitando, assim, o armazenamento desnecessário de dados em disco.

**Desvantagens:** Como a participação não é total, alguns valores nulos serão gerados. Porém, a maioria das espécies participará do relacionamento, assim, a desvantagem desta escolha é bem pequena.

**Solução alternativa:** Criar uma tabela a mais para o relacionamento.

### 11.2.2 Minério e Elemento Químico

**Solução adotada:** Para esta solução o relacionamento foi mapeado para dentro de minério. Assim, um dos atributos de minério faz o relacionamento ou recebe *NULL* se não participar do relacionamento.

**Vantagens:** Não criamos uma tabela a mais para o relacionamento evitando, assim, o armazenamento desnecessário de dados em disco.

**Desvantagens:** Como a participação não é total, alguns valores nulos serão gerados. Porém, a maioria dos minérios participará do relacionamento tornando a desvantagem desta escolha pequena.

**Solução alternativa:** Criar uma tabela a mais para o relacionamento.

## 11.3 Auto-relacionamentos

### 11.3.1 Planeta e Planeta

**Solução adotada:** Para este auto relacionamento, como é N:N, a única solução possível foi a de criar uma nova tabela. **Vantagens:** Para este caso esta é a única solução possível.

**Desvantagens:** Não garantimos a quebra dos ciclos explicados na primeira etapa do projeto.

## 11.4 Generalização

### 11.4.1 Fabricáveis -> Construções/Equipamentos/Nave

**Solução adotada:** O método adotado foi criar tanto uma tabela para a entidade pai quanto outra para os seus filhos, já que certas entidades filhas fazem relacionamentos além de possuírem certos atributos importantes a serem mapeados separadamente, facilitando a consulta, tendo em vista o grande número de itens que serão gerados ao longo do jogo.

**Vantagens:** Esta solução facilita a criação de relacionamentos próprios de cada uma das entidades específicas da generalização assim como evita o excesso de *NULL* já que cada uma delas possui vários atributos próprios.

**Desvantagens:** Varias tabelas devem ser criadas. a implementação deve garantir que só exista um filho, do tipo especificado pelo atributo de **Fabricáveis** já que se trata de um disjunção. A participação total na generalização não é garantida (deve ser tratada na aplicação).

**Solução alternativa:** Uma alternativa seria mapear todos os atributos filhos dentro da tabela principal ou possivelmente criar tabelas apenas para os tipos.

## 11.5 Atributos Multivalorados

**Solução adotada:** Todos os atributos multivalorados foram anotados, ou seja, criou-se uma nova tabela para cada um deles. São eles: Luas, Perigos, Tipos de Fauna/Flora e Tipos de Minério. Isso porque podem ter uma quantidade não prevista o que seria um empecilho para o funcionamento adequado do jogo, já que todos esses atributos são essenciais para a experiência do jogador.

**Vantagens:** Dessa forma garante-se que o número de atributos não precisa estar limitado a um número arbitrário definido para os campos dentro da tabela da entidade. Como o propósito do jogo é incentivar a exploração não seria condizente limitar as possibilidades de geração de

ambientes

**Desvantagens:** Uma tabela a mais dificulta a busca dos multivalorados dentro de uma entidade além de armazenar mais informações em disco.

**Solução alternativa:** Criar o numero máximo  $N_{max}$  do valor dentro da entidade. Esta solução geraria muitos *NULL*, ja que os numeros variam muito.

## 11.6 Atributos Derivados

### 11.6.1 Calor

Calor é calculado a partir da distância da estrela e do tamanho da estrela.

## 12 Mudanças realizadas entre a parte 2 e parte 3 no MER

Adaptou-se o modelo de acordo com os erros observados. Foram elas:

- Foi removida a anotação de apoio “Entidade Forte” do MR;
- Foi modificada a tabela “Local”, agora ela referencia a chave da tabela “Construções”;
- Foi corrigido os notes que definiam as chaves das agregações Viagem e Rota, e modificado o erro na chave de Viagem no MR;
- Foram acrescentados comentários acerca das alternativas para o mapeamento de agregação, assim como, justificativas para a escolha adotada para o mapeamento dos atributos derivados.

## 13 Implementação do Banco de Dados

### 13.1 Visão geral

Para a parte final do projeto, o diagrama MR foi transcrito para linguagem SQL utilizando o Oracle SGBD. Ademais, foram feitas 5 consultas na base de dados as quais serão descritas mais adiante. Além disso, foi desenvolvida uma aplicação em Python com interface gráfica integrada ao banco de dados, permitindo o cadastro de novos Planetas e Astronautas, os quais são adicionados automaticamente à base da dados. A aplicação é composta também por uma

funcionalidade de consulta. Foi feito por fim um tratamento de erros básico, permitindo, por exemplo, o *rollback* o que garante a atomicidade da aplicação, verificação dos dados de entrada entre outros.

A seguir, encontra-se um *link* para o repositório no *Github* com nosso código assim como o resto do projeto completo [Link para o repositório Github](#)

## 13.2 *Scripts de consulta*

Após a criação das tabelas do banco e inserção dos dados foram elaboradas as seguintes consultas:

- Dado um elemento químico em específico, definido por uma sigla, buscar por planetas que possuam uma forma de obter tal elemento e o meio pelo qual ele será adquirido, ou seja, matando espécies de seres vivos ou minerando minérios

```
1 SELECT
2     EQ.SIGLA AS ELEMENTO_QUIMICO ,
3     P.NOME AS PLANETA ,
4     'Especie: ' || E.NOME AS ENCONTRADO_EM
5 FROM
6     ELEMENTO_QUIMICO EQ
7 INNER JOIN ESPECIES E ON EQ.SIGLA = E.ELEMENTO
8 LEFT JOIN AMBIENTE A ON E.NOMEPLANETA = A.PLANETA
9 LEFT JOIN PLANETA P ON A.PLANETA = P.NOME
10 WHERE
11     EQ.SIGLA = 'Ni '
12
13 UNION
14
15 SELECT
16     EQ.SIGLA AS ELEMENTO_QUIMICO ,
17     P.NOME AS PLANETA ,
18     'Minerio: ' || M.NOME AS ENCONTRADO_EM
19 FROM
20     ELEMENTO_QUIMICO EQ
21 INNER JOIN MINERIO M ON EQ.SIGLA = M.ELEMENTO
22 INNER JOIN POSSUI POS ON M.NOME = POS.MINERIO
23 LEFT JOIN AMBIENTE A ON POS.AMBIENTE = A.PLANETA
```

```

24 LEFT JOIN PLANETA P ON A.PLANETA = P.NOME
25 WHERE
26     EQ.SIGLA = 'Ni';

```

- São feitos dois comandos SELECT os quais irão buscar por instâncias do elemento químico requisitado em espécies, as quais serão associadas ao seus planetas natais. Separadamente, ele faz a mesma busca porém para minérios. Para exibição, juntam-se as duas tabelas de forma concisa com o comando UNION. Optou-se por essa abordagem para tratar os casos em que os elementos pudessem ser obtidos tanto por minérios quanto por espécies, do contrário (ou seja, se um elemento não pudesse ser retirado de duas origens mineral e animal distintas) seria possível utilizar uma clausula CASE a qual os separaria entre os dois casos possíveis.

- Nessa consulta busca-se o vendedor que lucrou mais com suas vendas em um dia específico

```

1 SELECT V.NOME, SUM(VF.QUANTIDADE * F.DURABILIDADE) AS LUCRO_TOTAL
2
3 FROM VENDE_FABRICAVEL VF JOIN FABRICAVEIS F ON VF.FABRICAVEL = F.NOME
4 JOIN VENDEDOR V ON VF.VENDEDOR = V.NOME
5 GROUP BY V.NOME, VF."DATA"
6 ORDER BY LUCRO_TOTAL DESC
7 FETCH FIRST ROW ONLY;

```

- É feito um SELECT tanto no nome do vendedor quanto em uma soma de variáveis que determina o dito lucro. A tabela com registro das vendas é associada com a de fabricáveis pelo nome. A mesma é associada também com a de vendedores, permitindo assim retornar o NPC responsável pela transação, assim como o item negociado. Os resultados são agrupados (GROUP BY) por nome e data e ordenados (ORDER BY) de forma decrescente.

- Essa consulta irá buscar pelas construções instanciadas no mapa mais próximas de outra passada como parâmetro. A distância é calculada pela fórmula euclidiana com base nas chaves primárias, isto é, as coordenadas X,Y atribuídas a cada item

```

1 SELECT F.NOME AS FABRICAVEL, L.POSICA0X, L.POSICA0Y, TRUNC(SQRT(POWER
    (5 - L.POSICA0X, 2) + POWER(8 - L.POSICA0Y, 2))) AS DISTANCIA

```

```

2 FROM FABRICAVEIS F JOIN "LOCAL" L ON F.NOME = L.FABRICAVEL
3 WHERE F.NOME <> 'Space Station'
4 ORDER BY DISTANCIA
5 FETCH FIRST 5 ROWS ONLY;

```

- Selecionam-se os nomes dos fabricáveis associados a aqueles instanciados em um local, o qual fornece a posição geográfica X e Y. É aplicada uma condição de filtro para excluir as linhas em que o valor da coluna NOME na tabela FABRICAVEIS seja igual a 'Space Station'. E, por fim, são ordenados os construtos tendo como base as distâncias calculadas. O comando FETCH limita o resultado retornando apenas as primeiras 5 linhas do conjunto resposta.

- Calcula a distância total percorrida por cada Astronauta e ordenada por quem viajou mais

```

1 SELECT V.ASTRONAUTA, SUM(R.DISTANCIA_TOTAL) AS DISTANCIA_TOTAL
2 FROM VIAGEM V
3 JOIN ROTA R ON V.ORIGEM = R.ORIGEM AND V.DESTINO = R.DESTINO
4 GROUP BY V.ASTRONAUTA
5 ORDER BY DISTANCIA_TOTAL DESC;

```

- São selecionados astronautas que saíram de dadas origens e destinos em viagens com suas respectivas rotas. As distâncias de cada rota trilhada por um dado astronauta são somadas e agrupa-se o resultado pelo nome de cada viajante em ordem decrescente. O primeiro astronauta na tabela, portanto, será aquele que mais viajou.

- Nessa consulta são buscadas as espécies e minérios mais predominantes de cada bioma

```

1 WITH MostCommonMineralInBiome AS (
2     SELECT A.BIOMA,
3           M.TIPO AS TIPO_DE_MINERIO,
4           M.NOME AS MINERIO MAIS_COMUM,
5           COUNT(*) AS QTD,
6           ROW_NUMBER() OVER (PARTITION BY A.BIOMA ORDER BY COUNT(*)
7                               DESC) AS RN
8     FROM AMBIENTE A
9     JOIN POSSUI P ON A.PLANETA = P.AMBIENTE
10    JOIN MINERIO M ON P.MINERIO = M.NOME
11    GROUP BY A.BIOMA, M.TIPO, M.NOME
12 ),

```



```

12 MostCommonSpeciesInBiome AS (
13     SELECT A.BIOMA,
14            E.TIPO AS TIPO_DE_ESPECIE,
15            E.NOME AS ESPECIE MAIS_COMUM,
16            COUNT(*) AS QTD,
17            ROW_NUMBER() OVER (PARTITION BY A.BIOMA ORDER BY COUNT(*)
18                                DESC) AS RN
19 FROM AMBIENTE A
20 JOIN ESPECIES E ON A.PLANETA = E.NOMEPLANETA
21 GROUP BY A.BIOMA, E.TIPO, E.NOME
22 )
23 SELECT A.BIOMA,
24        M.TIPO_DE_MINERIO AS MINERIO MAIS_COMUM,
25        M.MINERIO MAIS_COMUM AS NOME_DO_MINERIO MAIS_COMUM,
26        S.TIPO_DE_ESPECIE AS ESPECIE MAIS_COMUM,
27        S.ESPECIE MAIS_COMUM AS NOME_DA_ESPECIE MAIS_COMUM
28 FROM (SELECT DISTINCT BIOMA FROM AMBIENTE) A
29 LEFT JOIN MostCommonMineralInBiome M ON A.BIOMA = M.BIOMA AND M.RN = 1
30 LEFT JOIN MostCommonSpeciesInBiome S ON A.BIOMA = S.BIOMA AND S.RN =
31     1;

```

- A implementação escolhida utiliza CTE's (*Common Table Expression*), os quais são usados para melhor agrupar e tratar os conjuntos de espécies e minérios, contando as instâncias de cada um, com o comando COUNT. A função ROW\_NUMBER() atribui um número de linha sequencial para cada linha dentro de cada bioma, com base na contagem decrescente de ocorrências. Isso é feito afim de ordenar, em cada CTE, as instâncias obtidas em biomas por ordem decrescente. Por fim, são escolhidos os primeiros das ordenações, através do através do parâmetro RN = 1 e associados com a tabela bioma.

## 13.3 Aplicação

### 13.3.1 Descrição

Nossa aplicação utiliza de uma interface gráfica que permite o usuário inserir um astronauta novo, representando um novo player, um planeta ou buscar as construções próximas da construção em que se encontra.

## 13.4 Bibliotecas utilizadas

- tkinter: Esta biblioteca é a interface gráfica padrão do Python. Ela fornece ferramentas para criar janelas, botões, entradas de texto e outros elementos de GUI.
- Módulos adicionais tkinter:
  - simpledialog: Fornece uma caixa de diálogo simples para interação com o usuário.
  - StringVar: Variável especial usada para rastrear e atualizar o conteúdo de widgets dinamicamente.
  - Toplevel: Cria uma nova janela superior.
- Módulos tkinter.ttk:
  - Combobox: Um *widget* de caixa de combinação, que permite a seleção de itens de uma lista suspensa.
  - Treeview: Um *widget* de tabela interativa que exibe dados em colunas.
- Módulos PIL (Pillow): *Python Imaging Library*, oferece suporte a uma variedade de formatos de imagem
  - Image: Da biblioteca Pillow, usada para representar e manipular imagens.
  - ImageTk: Permite converter imagens Pillow para um formato suportado por Tkinter para exibição em widgets.
- cx\_Oracle: Uma biblioteca para acesso a bancos de dados Oracle. Permite a conexão e interação com bancos de dados Oracle usando a linguagem Python.
- datetime: Fornece classes para manipulação de datas e horas em Python.
- json: Biblioteca para manipulação de dados JSON em Python. Pode ser usada para codificar e decodificar dados JSON.

### 13.4.1 Inserção realizada

Para as inserções, ao abrir a interface de aplicação, o usuário poderá selecionar entre astronauta ou planeta para que a coleta de dados comece. Uma vez escolhida uma das opções uma nova janela será aberta para coletar todos os dados necessários.

- **Coleta de dados e tratamento de erro:** Cada janela é encerrada chamando a próxima até terminar as informações necessárias. Para o astronauta basta o nome já que seu nascimento é definido pela data da inserção (momento em que o player se cadastra). Já para o planeta é preciso coletar alguns dados numéricos que podem ser NULL e dados que só podem ser uma dentre 3 opções. Para estes últimos, foi utilizada uma caixa de seleção para garantir que os dados coletados estejam no padrão. Valores não numéricos, varchar estourados ou unicidades violadas são impedidas com um *rollback* e uma janela de erro explicando o motivo da falha.

```
1 def coletar_dados_astronauta(conexao, etiqueta_nome, janela):
2     try:
3         nome_astronauta = simpledialog.askstring("Astronauta", "Digite
4         o nome/apelido do astronauta:")
5
6         if inserir_dados_astronauta(conexao, nome_astronauta) == 0:
7             # Se ocorrer um erro, abrir uma nova janela com a mensagem
8             de erro
9             mensagem_erro = "Erro ao inserir, nome inv lido"
10            exibir_janela_erro(janela, mensagem_erro)
11        else:
12            mensagem = f"Inserindo Astronauta: Nome - {nome_astronauta
13            }"
14            etiqueta_nome.config(text=mensagem)
15
16    except ValueError as erro:
17        # Se ocorrer um erro, abrir uma nova janela com a mensagem de
18        erro
19        exibir_janela_erro(janela, str(erro))
20
21 def coletar_dados_planeta(conexao, etiqueta_nome, janela):
22     try:
23         # Coletar o nome do planeta
24         nome_planeta = simpledialog.askstring("Planeta", "Digite o
25         nome do planeta:")
26
27         # Criar variáveis StringVar para armazenar as escolhas do
28         usu rio
29         escolha_tamanho_planeta = StringVar()
```

```

24     escolha_nivel_seguranca = StringVar()
25
26     # Criar a janela temporária para a seleção de opções
27     janela_opcoes = Toplevel(janela)
28     janela_opcoes.title("Selecione o Tamanho e Nível de
Segurança do Planeta")
29
30     # Exibir uma mensagem para instruir o usuário
31     Label(janela_opcoes, text="Selecione o tamanho e o nível de
segurança do planeta:").pack(pady=10)
32
33     # Criar opções para tamanho do planeta e nível de
segurança
34     opcoes_tamanho_planeta = ["Pequeno", "Médio", "Grande"]
35     opcoes_nivel_seguranca = ["Baixo", "Médio", "Alto"]
36
37     # Adicionar opções de tamanho do planeta ao menu
38     Label(janela_opcoes, text="Selecione o tamanho do planeta:").
pack(pady=5)
39     tamanho_planeta_menu = OptionMenu(janela_opcoes,
escolha_tamanho_planeta, *opcoes_tamanho_planeta)
40     tamanho_planeta_menu.pack(pady=5)
41
42     # Adicionar opções de nível de segurança ao menu
43     Label(janela_opcoes, text="Selecione o nível de segurança do
planeta:").pack(pady=5)
44     nivel_seguranca_menu = OptionMenu(janela_opcoes,
escolha_nivel_seguranca, *opcoes_nivel_seguranca)
45     nivel_seguranca_menu.pack(pady=5)
46     # Criar uma função para verificar se ambas as opções foram
selecionadas
47     def verificar_selecao():
48         tamanho_planeta = escolha_tamanho_planeta.get()
49         nivel_seguranca = escolha_nivel_seguranca.get()
50         if tamanho_planeta and nivel_seguranca:
51             # Se ambas as opções foram selecionadas, fechar a
janela
52             janela_opcoes.destroy()
53         else:

```

```

54         # Caso contrário, exibir uma mensagem de erro
55         mensagem_erro = "Selecione uma opção para o tamanho
e o nível de segurança do planeta."
56         exibir_janela_erro(janela, mensagem_erro)
57
58         # Adicionar um botão para verificar a seleção
59         Button(janela_opcoes, text="OK", command=verificar_selecao).
pack(pady=10)
60
61         # Esperar até que o usuário faça uma escolha
62         janela_opcoes.wait_window(janela_opcoes)
63
64         distancia_estrela = simpledialog.askstring("Distância da
Estrela", "Digite a distância da estrela (ou deixe em branco para
NULL):")
65         if distancia_estrela == "":
66             distancia_estrela = None
67         else:
68             try:
69                 distancia_estrela = int(distancia_estrela)
70             except ValueError:
71                 mensagem_erro = "A distância deve ser um número
inteiro."
72                 exibir_janela_erro(janela, mensagem_erro)
73                 return
74
75         # Coletar a temperatura do planeta
76         calor = simpledialog.askstring("Calor", "Digite a temperatura
do planeta (ou deixe em branco para NULL):")
77         if calor == "":
78             calor = None
79         else:
80             try:
81                 calor = int(calor)
82             except ValueError:
83                 mensagem_erro = "A temperatura deve ser um inteiro."
84                 exibir_janela_erro(janela, mensagem_erro)
85                 return
86

```

```

87         # Coletar a velocidade de transla o do planeta
88         translacao = simpledialog.askstring("Transla o", "Digite a
velocidade de transla o do planeta (ou deixe em branco para NULL
):")
89         if translacao == "":
90             translacao = None
91         else:
92             try:
93                 translacao = int(translacao)
94             except ValueError:
95                 mensagem_erro = "A transla o deve ser um inteiro."
96                 exibir_janela_erro(janela, mensagem_erro)
97                 return
98
99         # Coletar a velocidade de rota o do planeta
100        rotacao = simpledialog.askstring("Rota o", "Digite a
velocidade de rota o do planeta (ou deixe em branco para NULL)
:")
101        if rotacao == "":
102            rotacao = None
103        else:
104            try:
105                rotacao = int(rotacao)
106            except ValueError:
107                mensagem_erro = "A rota o deve ser um inteiro."
108                exibir_janela_erro(janela, mensagem_erro)
109                return
110
111        # Coletar o nome da estrela
112        nome_estrela = simpledialog.askstring("Estrela", "Digite o
nome da estrela que o planeta orbita:")
113
114        # Criar um dicion rio com os dados coletados
115        dicionario_planeta = {
116            "nome": nome_planeta,
117            "tamanho": escolha_tamanho_planeta.get(),
118            "nivel_seguranca": escolha_nivel_seguranca.get(),
119            "distancia_estrela": distancia_estrela,
120            "calor": calor,

```

```

121         "translacao": translacao,
122         "rotacao": rotacao,
123         "estrela": nome_estrela
124     }
125
126     if inserir_dados_planeta(conexao, dicionario_planeta) == 0:
127         # Se ocorrer um erro, abrir uma nova janela com a mensagem
de erro
128         mensagem_erro = "Erro ao inserir, dados inv lidos"
129         exibir_janela_erro(janela, mensagem_erro)
130     else:
131         mensagem = f"Inserindo Planeta: Nome - {nome_planeta}"
132         etiqueta_nome.config(text=mensagem)
133
134     except ValueError as erro:
135         # Se ocorrer um erro, abrir uma nova janela com a mensagem de
erro
136         exibir_janela_erro(janela, str(erro))

```

- **Inserção no SQL:** Após isso, as funções de inserção são chamadas. Deste modo, os valores, já tratados, são passados para os comandos SQL especificados anteriormente

```
1     def inserir_dados_astronauta(conexao, nome_dado):
2         if conexao is None:
3             print("N o      poss vel inserir dados sem uma conex o
v lida.")
4             return
5
6         cursor = conexao.cursor()
7
8         try:
9             Nascimento = datetime.now()
10            conexao.begin()
11
12            cursor.execute("INSERT INTO ASTRONAUTA (NOME, NASCIMENTO)
VALUES (:1, :2)", (nome_dado, Nascimento))
13            conexao.commit()
14
15            print("Cadastro realizado com sucesso!")
16
17        except cx_Oracle.IntegrityError as erro:
18            if erro.args[0].code == 12899: # C digo de erro para tamanho
excedido
19                raise ValueError("Erro: Nome do astronauta excede o
tamanho m ximo permitido.")
20                conexao.rollback()
21            elif erro.args[0].code == 1: # C digo de erro para
viola o de unicidade
22                raise ValueError("Nome duplicado, por favor, insira um
nome nico .")
23                conexao.rollback()
24            else:
25                print(f"Erro no banco de dados ao inserir dados: {erro}")
26                conexao.rollback()
27                return 0
28
29        finally:
30            cursor.close()
```



```

31
32     return 1 # Retorno indicando sucesso
33
34     def inserir_dados_planeta(conexao, dados_planeta):
35         if conexao is None:
36             print("N o      poss vel inserir dados sem uma conex o
37             v lida.")
38             return
39
40         cursor = conexao.cursor()
41
42         try:
43             conexao.begin()
44
45             consulta = """
46                 INSERT INTO PLANETA (NOME, TAMANHO, NIVEL_DE_SEGURANCA,
47                 DISTANCIA_DA_ESTRELA, CALOR, TRANSLACAO, ROTACAO, ESTRELA)
48                 VALUES (:nome, :tamanho, :nivel_seguranca, :
49                 distancia_estrela, :calor, :translacao, :rotacao, :estrela)
50             """
51
52             # Executar a consulta, tratando os valores nulos
53             cursor.execute(consulta, {
54                 'nome': dados_planeta['nome'],
55                 'tamanho': dados_planeta.get('tamanho', None),
56                 'nivel_seguranca': dados_planeta.get('nivel_seguranca',
57                 None),
58                 'distancia_estrela': dados_planeta.get('distancia_estrela
59                 ', None),
60                 'calor': dados_planeta.get('calor', None),
61                 'translacao': dados_planeta.get('translacao', None),
62                 'rotacao': dados_planeta.get('rotacao', None),
63                 'estrela': dados_planeta['estrela']
64             })
65             conexao.commit()
66
67             print("Cadastro realizado com sucesso!")
68
69         except cx_Oracle.IntegrityError as erro:

```

```

65         if erro.args[0].code == 12899: # Código de erro para tamanho
            excedido
66             raise ValueError("Erro: Nome do astronauta excede o
tamanho máximo permitido.")
67             conexao.rollback()
68         elif erro.args[0].code == 1: # Código de erro para
violação de unicidade
69             raise ValueError("Nome duplicado, por favor, insira um
nome único.")
70             conexao.rollback()
71         else:
72             print(f"Erro no banco de dados ao inserir dados: {erro}")
73             conexao.rollback()
74             return 0
75
76     finally:
77         cursor.close()
78
79     return 1 # Retorno indicando sucesso
80

```

### 13.4.2 Consulta implementada

Analogamente ao item anterior, para a consulta os dados são coletados pela janela e já tratados antes da chamada da função SQL. Ao chamar o SQL passamos a posição da variável dentro do dicionário para que seja possível alterar o valor a ser buscado. Através da função Fetchall pegamos todos os dados retornados pela busca e imprimimos em forma de tabela na aplicação

```

1 def busca_construcoes_proximas(conexao, nome_construcao, etiqueta_nome,
janela):
2     if conexao is None:
3         print("Não é possível buscar dados sem uma conexão válida.")
4         return None # Retorne None para indicar erro
5
6     cursor = conexao.cursor()
7
8     try:
9         cursor.execute("""

```

```

10         SELECT
11             F.NOME AS FABRICAVEL ,
12             L.POSICA0X ,
13             L.POSICA0Y ,
14             TRUNC(SQRT(POWER(5 - L.POSICA0X, 2) + POWER(8 - L.POSICA0Y,
25         2))) AS DISTANCIA
15         FROM
16             FABRICAVEIS F
17         JOIN "LOCAL" L ON F.NOME = L.FABRICAVEL
18         WHERE
19             F.NOME <> :nome
20         ORDER BY
21             DISTANCIA
22         FETCH FIRST 5 ROWS ONLY
23         """ , {'nome': nome_construcao})
24
25         dados = cursor.fetchall() # Retorna uma lista de tuplas
26         conexao.commit()
27
28         print("Consulta realizada com sucesso!")
29         cabecalho = ["Constru o", "Posi o X", "Posi o Y", "
Dist ncia"]
30         dados_com_cabecalho = [cabecalho] + dados
31
32         mostrar_tabela_resultados(janela, dados_com_cabecalho)
33         return dados
34
35     except cx_Oracle.DatabaseError as erro:
36         print(f"Erro no banco de dados ao executar consulta: {erro}")
37         conexao.rollback()
38
39     finally:
40         cursor.close()
41
42     return None # Retorne None em caso de erro
43
44 def coletar_dados_consulta(conexao, etiqueta_nome, janela):
45     try:

```

```

46     nome_construcao = simpledialog.askstring("Construção", "Digite o
nome da construção em que você se encontra:")
47     if busca_construcoes_proximas(conexao, nome_construcao,
etiqueta_nome, janela) == 0:
48         # Se ocorrer um erro, abrir uma nova janela com a mensagem de
erro
49         mensagem_erro = "Erro ao buscar construções próximas, nome
inválido"
50         exibir_janela_erro(janela, mensagem_erro)
51     else:
52         mensagem = f"Consulta realizada para a construção: {
nome_construcao}"
53         etiqueta_nome.config(text=mensagem)
54
55     except ValueError as erro:
56         # Se ocorrer um erro, abrir uma nova janela com a mensagem de erro
57         exibir_janela_erro(janela, str(erro))

```

## 14 Conclusão

Ao final do trabalho obteve-se um saldo positivo do trabalho. Foi possível treinar todos os conceitos que a matéria cobriu desde do diagrama MER, até a fase final com uso do SGBD, elaboração do código em SQL e integração com uma aplicação própria.

## 15 Referências

1. Fandom.com - “Wiki *No Man’s Sky*” - Comunidade No Man’s Sky.