Projeto Final de Banco de Dados
Porto de Suape

Igor David – 18/0102141
Luca Delpino Barbabella - 18/0125559

1. Introdução

A necessidade de monitorar e organizar tarefas e pessoas é algo cada vez mais relevante e necessário nos ambientes de trabalho modernos. Uma das ferramentas atuais mais poderosas para lidar com essa necessidade são os bancos de dados. Os bancos de dados nos permitem ter um alto nível de abstração sobre os sistemas que queremos gerenciar, e os sistemas de gerenciamento de bancos de dados tornam essas interações ainda mais acessíveis.

Ainda pensando em ambientes de trabalho, devemos conseguir relacionar os funcionários com suas respectivas tarefas, ferramentas de trabalho, etc. Para isso, os Bancos de Dados Relacionais foram desenvolvidos. Eles são baseados no modelo relacional, que possibilita conexões que representam relações entre suas diversas entidades (tabelas que guardam informações, ou atributos).

Um exemplo de um ambiente de trabalho complexo que se beneficiaria utilizando um banco de dados relacional é um porto de navios cargueiros, onde é preciso relacionar funcionários a equipes e ferramentas de trabalho (caminhões, gruas, empilhadeiras), relacionar diferentes atividades a equipes, acompanhar deslocamentos de containers, etc. Com essa ideia em mente, implementamos um banco de dados relacional que faz a gestão de um porto cargueiro com suporte para CRUD em todas as tabelas. Nosso sistema é construído em SQL no SGBD PostgreSQL, e nossa interface e camada de persistência foram construídos em Python.

Organizamos esse relatório de forma que seja possível apresentar conceitos teóricos e práticos sobre a aplicação do nosso sistema de forma segmentada.

2. Diagrama de Entidade Relacionamento

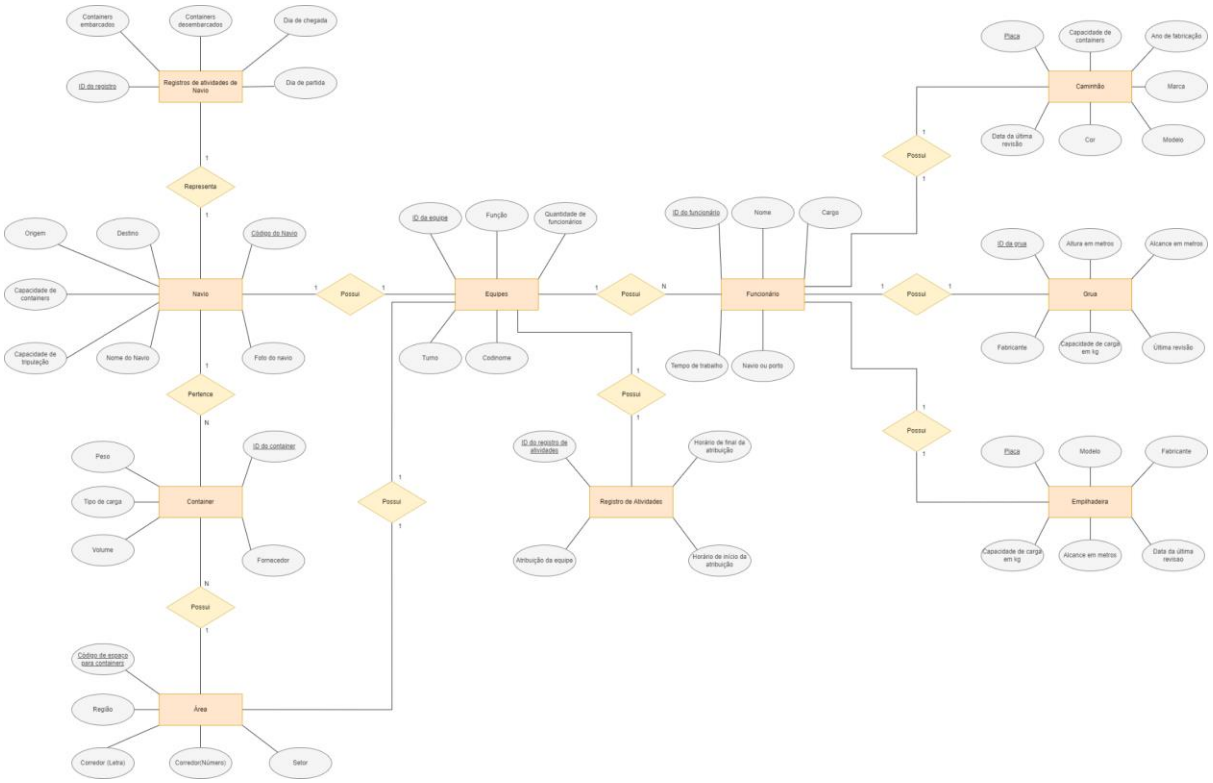


Figura 1. Diagrama de Entidade Relacionamento do porto

3. Modelo Relacional

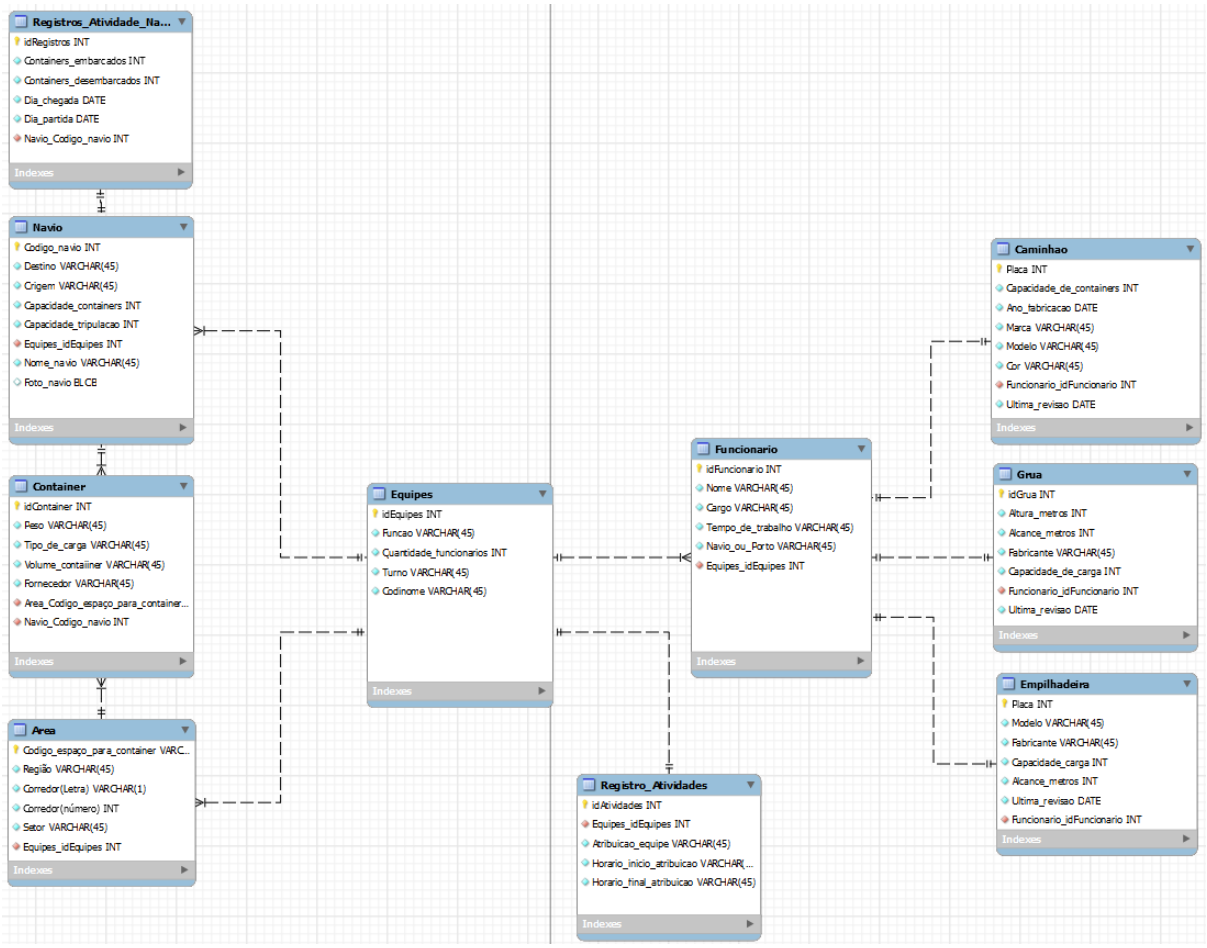


Figura 2. Modelo relacional montado no mySQL Workbench

4. Álgebra relacional

$$\sigma_{(fk.navio_{codigo_{navio}} = codigo_{navio}\ AND\ fk.area\_codigo\_espaco\_para\_container = codigo\_espaco\_para\_container)}(NAVIO\ X\ AREA\ X\ CONTAINER)$$

Na equação acima é feito um produto cartesiano entre as tabelas navio, área e container considerando as chaves primárias e estrangeiras derivadas das mesmas tabelas.

$$\pi^{idcontainer,codigo_{navio},corredor_{letras},coderredor_{numero}} \sigma_{(fk.navio_{codigo_{navio}} = codigo_{navio}\ AND\ fk.area\_codigo\_espaco\_para\_container = codigo\_espaco\_para\_container)}(NAVIO\ X\ AREA\ X\ CONTAINER)$$

Repete-se a primeira equação, porém selecionando apenas as colunas especificadas pela operação de projeção.

$$\sigma(navio_{(codigo\_navio)} |X| container_{(navio\_codigo\_navio)}) |X|\ registros\_atividade\_navios_{(navio\_codigo\_navio)}$$

Operação de junção utilizando chaves primárias e estrangeiras em comuns e a seleção desses atributos.

$$\pi^{nome\_navio,containers\_embarcados,tipo\_de\_carga} \sigma(navio_{(codigo\_navio)} |X| container_{(navio\_codigo\_navio)}) |X|\ registros\_atividade\_navios_{(navio\_codigo\_navio)}$$

Mesma operação realizada na equação anterior, porém separando atributos específicos de cada tabela para projeção.

$$\pi^{n.nome\_navio,c.tipo\_de\_carga,r.containers\_embarcados} \sigma_{(navio_{codigo_{navio}} = codigo_{navio} and\ navio_{codigo_{navio}} = codigo_{navio} and\ containers_{embarcados} > 35)}(NAVIO\ X\ registros\_atividade\_navios\ X\ CONTAINER)$$

Na operação acima são projetados apenas um atributo de cada tabela, além disso na seleção é utilizado operações de and e maior que.

5. Avaliação das formas normais

Visando a diminuição da redundância, aumentar a integridade dos dados e aumentar o desempenho, a normalização de bancos de dados é um conjunto de regras. Essa avaliação é feita analisando os atributos das entidades do banco de dados. Faremos a análise de cinco tabelas do nosso banco de dados de acordo com as seguintes formas normais:

- **1ª Forma normal:** Para essa forma ser cumprida, deve-se ter apenas atributos com um único valor e não devem haver atributos repetidos. Ou seja, devem apenas existir atributos atômicos.

- **2ª Forma normal:** Primeiramente, é preciso estar na 1ª forma normal. Além disso, os atributos não chave devem depender unicamente do atributo chave.

- **3ª Forma normal:** Para estar na 3ª forma normal, é preciso estar na 2ª forma normal e todos os atributos devem ser independentes uns dos outros, e dependentes somente da chave primária.

Com isso em mente, vamos analisar as seguintes tabelas do nosso banco de dados sobre o porto:

5.1 Tabela de Registro de atividades de Equipes

```
CREATE TABLE IF NOT EXISTS Registro_de_trabalhos (
  idTrabalho INT NOT NULL,
  Equipes_idEquipes INT NOT NULL,
  Atribuicao_equipe VARCHAR(60) NOT NULL,
  Horario_inicio_atribuicao VARCHAR(45) NOT NULL,
  Horario_final_atribuicao VARCHAR(45) NOT NULL,
  PRIMARY KEY (idTrabalho),
  --INDEX fk_Inspetores_Equipes1_idx (Equipes_idEquipes ) ,
  CONSTRAINT fk_Trabalho_Equipes1
    FOREIGN KEY (Equipes_idEquipes)
    REFERENCES Equipes (idEquipes)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

- **1FN:** A tabela está na primeira forma normal pois todos os seus atributos são atômicos.
- **2FN:** A tabela está na primeira forma normal, então um dos fatores necessários já é cumprido. Além disso, todos os atributos dependem unicamente do atributo chave *idTrabalho* pois só a partir dele é possível distinguir o registro.
- **3FN:** A tabela está na segunda forma normal e todos os seus atributos são independentes entre si. Pode-se concluir que a tabela está na terceira forma normal.

5.2 Tabela de Caminhões

```
CREATE TABLE IF NOT EXISTS Caminhao (
  Placa INT NOT NULL,
  Capacidade_de_containers INT NOT NULL,
  Ano_fabricacao DATE NOT NULL,
  Marca VARCHAR(45) NOT NULL,
  Modelo VARCHAR(45) NOT NULL,
  Cor VARCHAR(45) NOT NULL,
  Funcionario_CPF INT NOT NULL,
  Ultima_revisao DATE NOT NULL,
  PRIMARY KEY (Placa),
  --INDEX fk_Caminhao_Funcionario1_idx (Funcionario_idFuncionario ) ,
  CONSTRAINT fk_Caminhao_Funcionario1
    FOREIGN KEY (Funcionario_CPF)
    REFERENCES Funcionario (CPF)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

- **1FN:** A tabela está na primeira forma normal pois todos os seus atributos são atômicos.
- **2FN:** A tabela está na primeira forma normal, então um dos fatores necessários já é cumprido. Além disso, todos os atributos dependem unicamente do atributo chave *Placa*, pois só a partir dele é possível distinguir o caminhão do qual estamos falando.
- **3FN:** A tabela está na segunda forma normal, porém os atributo *Marca* e *Modelo* dependem um do outro. Logo, a tabela não está na terceira forma normal.
-

5.3 Tabela de Empilhadeiras

```sql
CREATE TABLE IF NOT EXISTS Empilhadeira (
  Placa INT NOT NULL,
  Modelo VARCHAR(45) NOT NULL,
  Fabricante VARCHAR(45) NOT NULL,
  Capacidade_carga_kg INT NOT NULL,
  Alcance_metros INT NOT NULL,
  Ultima_revisao DATE NOT NULL,
  Funcionario_CPF INT NOT NULL,
  PRIMARY KEY (Placa),
  --INDEX fk_Empilhadeira_Funcionario1_idx (Funcionario_idFuncionario ) ,
  CONSTRAINT fk_Empilhadeira_Funcionario1
    FOREIGN KEY (Funcionario_CPF)
    REFERENCES Funcionario (CPF)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

- **1FN:** A tabela está na primeira forma normal pois todos os seus atributos são atômicos.
- **2FN:** A tabela está na primeira forma normal, então um dos fatores necessários já é cumprido. Além disso, todos os atributos dependem unicamente do atributo chave *Placa*, pois só a partir dele é possível distinguir a empilhadeira da qual estamos falando.
- **3FN:** A tabela está na segunda forma normal e todos os atributos são independentes uns dos outros. Por isso, a tabela está na terceira forma normal.

5.4 Tabela de Gruas

```sql
CREATE TABLE IF NOT EXISTS Grua (
  idGrua INT NOT NULL,
  Altura_metros INT NOT NULL,
  Alcance_metros INT NOT NULL,
  Fabricante VARCHAR(45) NOT NULL,
  Capacidade_de_carga_kg INT NOT NULL,
  Funcionario_CPF INT NOT NULL,
  Ultima_revisao DATE NOT NULL,
  PRIMARY KEY (idGrua),
  --INDEX fk_Grua_Funcionario1_idx (Funcionario_idFuncionario ) ,
  CONSTRAINT fk_Grua_Funcionario1
    FOREIGN KEY (Funcionario_CPF)
    REFERENCES Funcionario (CPF)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

- **1FN:** A tabela está na primeira forma normal pois todos os seus atributos são atômicos.
- **2FN:** A tabela está na primeira forma normal, então um dos fatores necessários já é cumprido. Além disso, todos os atributos dependem unicamente do atributo chave *idGrua*, pois só a partir dele é possível distinguir a grua da qual estamos falando.

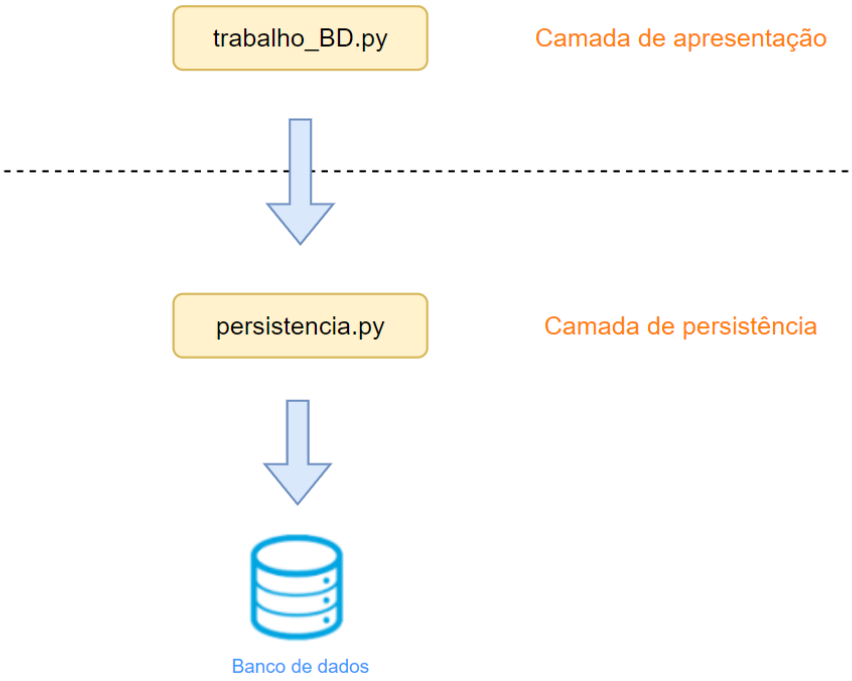
- **3FN:** A tabela está na segunda forma normal e todos os atributos são independentes uns dos outros. Por isso, a tabela está na terceira forma normal.

5.5 Tabela de Containers

```
CREATE TABLE IF NOT EXISTS Container (
  idContainer INT NOT NULL,
  Peso_KG INT NOT NULL,
  Tipo_de_carga VARCHAR(45) NOT NULL,
  Volume_container_litros VARCHAR(45) NOT NULL,
  Fornecedor VARCHAR(45) NOT NULL,
  Area_Codigo_espaco_para_container INT NOT NULL,
  Navio_Codigo_navio INT NOT NULL,
  PRIMARY KEY (idContainer),
  --INDEX fk_Container_Area1_idx (Area_Codigo_espaco_para_container ) ,
  --INDEX fk_Container_Navio1_idx (Navio_Codigo_navio ) ,
  CONSTRAINT fk_Container_Area1
    FOREIGN KEY (Area_Codigo_espaco_para_container)
    REFERENCES Area (Codigo_espaco_para_container)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_Container_Navio1
    FOREIGN KEY (Navio_Codigo_navio)
    REFERENCES Navio (Codigo_navio)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

- **1FN:** A tabela está na primeira forma normal pois todos os seus atributos são atômicos.
- **2FN:** A tabela está na primeira forma normal, então um dos fatores necessários já é cumprido. Além disso, todos os atributos dependem unicamente do atributo chave *idContainer*, pois só a partir dele é possível distinguir o container do qual estamos falando.
- **3FN:** A tabela está na segunda forma normal e todos os atributos são independentes uns dos outros. Por isso, a tabela está na terceira forma normal.

6. Camada de mapeamento para a tabela Funcionário



Na imagem acima vemos uma representação do fluxo de dados no sistema que construímos. Nele, o usuário interage com o computador na camada de apresentação, composta pelo arquivo trabalho_BD.py, que envia requisições para a chamada de persistência composta pelo arquivo pesistencia.py, que faz as chamadas ao banco de dados em SQL.

7. Fontes

https://www.oracle.com/br/database/what-is-a-relational-database/

https://docs.microsoft.com/pt-br/office/troubleshoot/access/database-normalization-description

https://pt.wikipedia.org/wiki/Normaliza%C3%A7%C3%A3o_de_dados

https://github.com/igordavid13/PORTO