Universidade Federal de Minas Gerais
Instituto de Ciéncias Exatas
Graduação em Ciéncia da Computação
Disciplina de Introdução a Banco de Dados

Trabalho Prático

Banco de dados para consumo de arte

dos requisitos à implementação final

Aluna: Giulia Monteiro Silva Gomes Vieira
Matrícula: 2016006492

Data: 4 de dezembro de 2023

Conteúdo

1 Trabalho Prático 01

1.1 Requisitos de dados

Para projetar um banco de dados usado em um sisema de gerenciamento de consumo de arte para alocação de recursos para artistas devemos considerar quais informações são pertinentes serem armazenadas sobre cada tipo de arte e artista, e qual o padrão de consumo de arte dos usuários, para que este aponte os artistas que produzem objetos mais acessados.

A divisão tradicional dos tipos de arte é: Música, Escultura, Pintura, Literatura, Arquitetura, Performática (teatro e dança), e Filme.

Sendo assim, podemos identificar como requisitos para este sisema as seguintes diferenciações:

- Objeto artístico (generalização):
  - Música:
    - ID: identificador único no sistema
    - Nome: nome da música
    - Tema: álbum
    - Ano: ano em que a música foi publicada
    - Autor: banda ou artista que escreveu a música
    - Executor: banda ou artista que toca a música
    - Detentor: quem possui os direitos sobre a música
    - Gênero: genero percentente
  - Escultura:
    - ID: identificador único no sistema
    - Nome: nome da escultura

    * Tema: temática da escultura
    * Material: material de que é feita
    * Ano: ano em que a escultura foi terminada
    * Autor: quem produziu a escultura
    * Detentor: quem é o dono da escultura
    * Gênero: genero percentente
    * Local: onde está a escultura
- Pintura:
    * ID: identificador único no sistema
    * Nome: nome da pintura
    * Tema: temática da pintura
    * Material: material de que é feita
    * Ano: ano em que a pintura foi terminada
    * Autor: quem produziu a pintura
    * Detentor: quem é o dono da pintura
    * Gênero: genero percentente
    * Local: onde está a pintura
- Literatura:
    * ID: identificador único no sistema
    * Nome: nome do livro
    * Tema: temática do livro
    * Ano: ano em que o livro foi terminada

    * Autor: quem escreveu o livro
    * Detentor: quem detém os direitos sobre o livro
    * Gênero: movimento artístico ao qual pertence
- Arquitetura:
    * ID: identificador único no sistema
    * Nome: nome da contrução
    * Material: material da construção
    * Ano: ano em que foi contruído
    * Autor: quem desenhou o projeto
    * Executor: quem construiu
    * Detentor: quem é o dono da construção
    * Local: onde está
    * Gênero: genero percentente
- Performance:
    * ID: identificador único no sistema
    * Nome: nome da performance
    * Tema: temática da performance
    * Tipo: teatro ou dança
    * Ano: ano em que foi publicada
    * Autor: quem escreveu ou coreografou a performance
    * Executor: quem está performando
    * Detentor: quem tem os direitos sobre a performance

        * Gênero: genero percentente
        * Local: onde está acontecendo
    - Filme:
        * ID: identificador único no sistema
        * Nome: nome do filme
        * Tema: temática do filme
        * Gênero: genero cinematográfico percentente
        * Ano: ano em que foi publicado
        * Autor: quem escreveu o filme
        * Diretor: quem dirigiu o filme
        * Executor: quem está performando o filme
        * Detentor: quem tem os direitos sobre o filme
- Artista:
    - ID: identificador único no sistema
    - Nome: nome da pessoa ou grupo
    - AnoInício: ano nascimento ou inicio do grupo
    - AnoFim: ano de morte ou fim do grupo
- Usuário:
    - ID: identificador único no sistema
    - Nome: nome do usuário
    - email: email do usuário
- Logs:

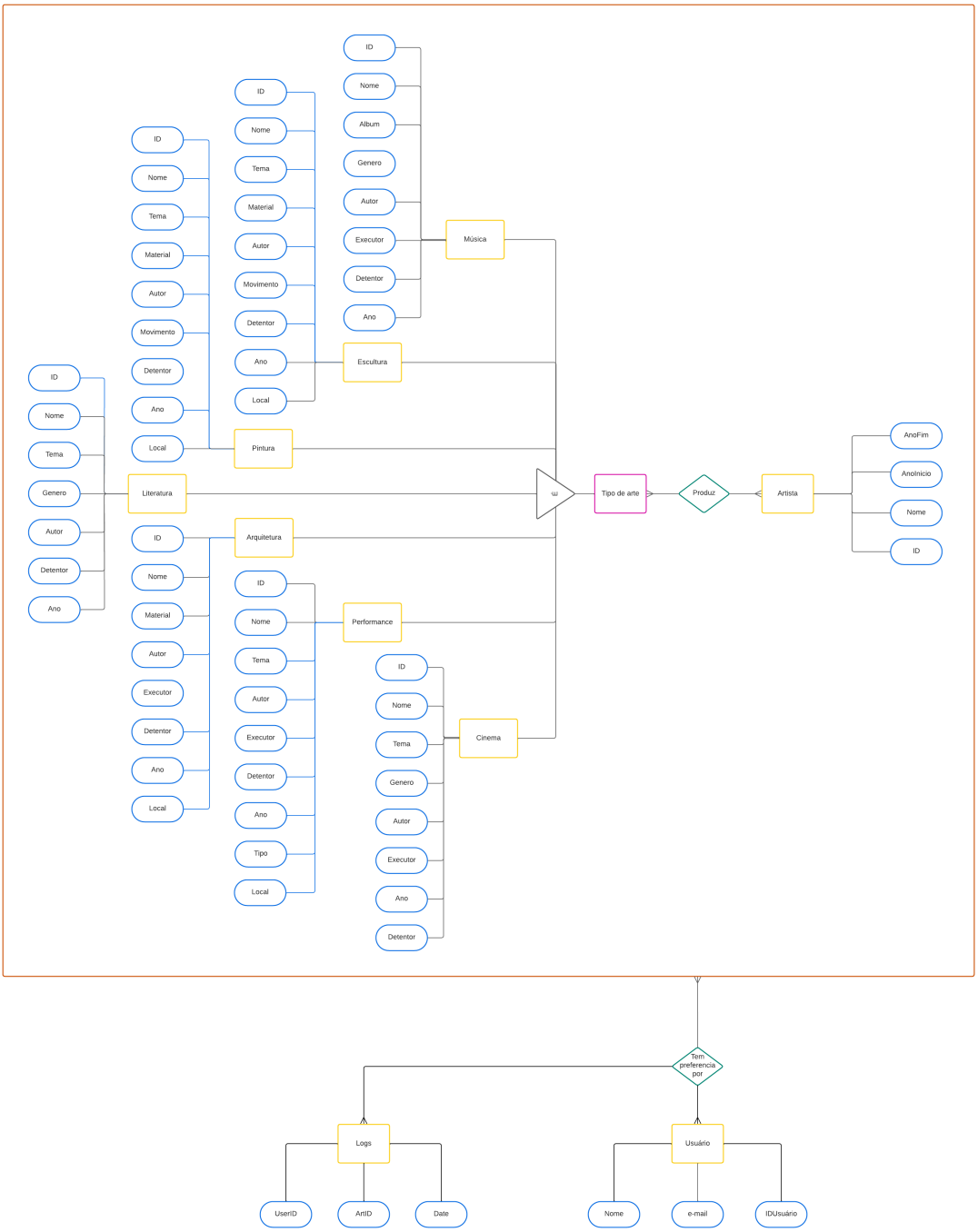
- IDUsuário: identificador do usuário
- IDObjeto: objeto artístico consumido
- Data: data em que foi consumido

Todas as entidades citadas têm um identificador único porque estamos supondo que todas as características podem se repetir e não queremos utilizar chave composta.

Estas entidades se relacionam da seguinte maneira:
Um objeto artístico pode ser produzido por vários artistas e cada artista de produzir vários objetos artísticos de tipos diferentes. Sendo assim, temos uma relação M:N entre estas categorias.
Um usuário pode consumir diversos objetos artísticos, e cada objeto pode ser consumido por vários usuários. Sendo assim, esta relanção também é M:N.
Um usuário pode ter preferência por diversos artistas, e cada artista pode ser preferido o por vários usuários. Sendo assim, esta relanção também é M:N.
Um usuário pode ser registrado nos logs diversas vezes, e os logs dizes respeito a todos os usuários. Portanto a relação também é M:N.
Um objeto pode ser acessado nos logs diversas vezes, e os logs dizes respeito a todos os usuários. Portanto a relação também é M:N.

1.2 Esquema de entidades e relacionamentos estendido

Dadas as restrições apresentadas anteriormente, portanto, é possível desenhar um esquema de entidades e relacionamentos estendido da seguinte maneira:

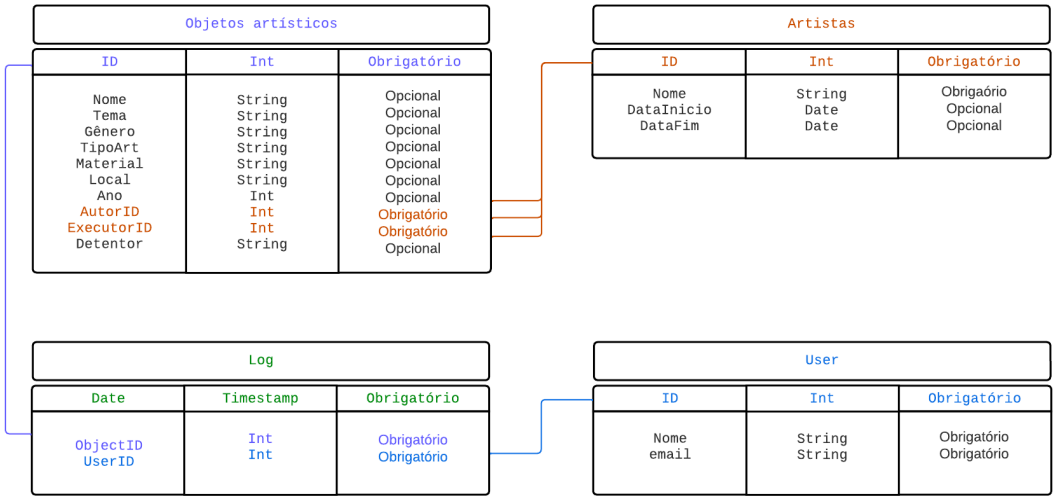


2 Trabalho Prático 02

2.1 Esquema relacional

Prosseguindo com o desenvolvimento iniciado no trabalho anterior, aqui estabeleceremos o esquema relacional derivado do esquema de entidade e relacionamentos estendido.
Como podemos perceber no esquema da sessão anterior, as formas de arte são especificações do tipo geral objeto artístico e seus atributos são muito parecidos. Por este motivo, decidi que a super-entidade Arte será uma tabela só, onde algumas colunas serão opcionais.

Na tabela a seguir, os atributos chave de cada tabela estão coloridos, e suas relações como chave estrangeira demarcadas com conectores da mesma cor. O domínio de cada um está na segunda coluna, e a possibilidade de aceitar-se valor NULL (ou seja, obrigatoriedade ou não do campo ser preenchido) na terceira coluna.



2.2 Script para geração do banco de dados

```sql
-- Table 01: Artistic Object
CREATE TABLE ArtisticObject (
    ID INT PRIMARY KEY,
    Name VARCHAR(255),
    Theme VARCHAR(255),
    Genre VARCHAR(255),
    Year VARCHAR(4),
    TypeArt VARCHAR(255),
    Material VARCHAR(255),
    AuthorID INT NOT NULL,
    ExecutorID INT NOT NULL,
    Owner VARCHAR(255),
    FOREIGN KEY (AuthorID) REFERENCES Artists(ID),
    FOREIGN KEY (ExecutorID) REFERENCES Artists(ID)
);

-- Table 02: Artists
CREATE TABLE Artists (
    ID INT PRIMARY KEY,
    Name VARCHAR(255),
    DateInit DATE,
    DateEnd DATE
);

-- Table 03: Users
CREATE TABLE Users (
    ID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255)
);

-- Table 04: Logs
CREATE TABLE Logs (
    Date TIMESTAMP PRIMARY KEY,
```

```
    ObjectID INT NOT NULL,
    ArtistID INT NOT NULL,
    FOREIGN KEY (ObjectID) REFERENCES ArtisticObject(ID),
    FOREIGN KEY (ArtistID) REFERENCES Artists(ID)
);
```

Este script foi desenvolvido pensando no modelo SQLite. Em outros modelos os CONSTRAINTS de FOREIGN KEY devem ser estabelecidos posteriormente, visto que a tabela referenciada deve existir para ser referenciada. Neste caso, as linhas iniciadas com FOREIGN KEY seriam subsitituídas pela adição das seguintes linhas ao final do script:

```
-- Add Foreign Key Constraints

-- Add foreign key constraints to ArtisticObject table
ALTER TABLE ArtisticObject
ADD FOREIGN KEY (AuthorID) REFERENCES Artists(ID),
ADD FOREIGN KEY (ExecutorID) REFERENCES Artists(ID);

-- Add foreign key constraints to Logs table
ALTER TABLE Logs
ADD FOREIGN KEY (ObjectID) REFERENCES ArtisticObject(ID),
ADD FOREIGN KEY (ArtistID) REFERENCES Artists(ID);
```

2.3 Protótipo do sistema

Para criar o banco:

```
sqlite3 db.db < create_db.sql
```

Para popular o banco:

```
python populate_db.py
```

Para operar o banco: a saïida desta parte deve ser (1) uma lista de todos os artistas vivos até 2019, (2) uma lista vazia de todos os artistas vivos até 2019, gerada após editarmos a data de morte de todos estes artistas para 2018, e (3) o nome do artista mais visitado, com o número de visitas.

```
python operate_db.py
```

3 Trabalho Prático 03

Dado o banco de dados e operações construídas nas sessões anteriores, algumas informações técnicas sobre o projeto podem ser extrapoladas das definições e implementações anteriores. Neste trabalho, então, desenvolveremos sete delas: (1) Transações, (2) Controle de concorrência, (3) Recuperação de falhas, (4) Normalização, (5) NoSQL (Não somente SQL), (6) Manipulação por linguagem hospedeira, (7) Aplicação em pesquisa.

3.1 Transações

3.1.1 Fundamentação teórica

Uma transação é uma sequência de um ou mais comandos SQL executados em conjunto, como uma unidade. Para tanto, opera se em contexto "tudo ou nada", ou seja, se uma parte da transação não funcionar nenhuma é operada no banco, isso ocorre para preservar a integridade do banco e a independência entre transações separadas. Contudo, se a transação for possível por completo seu resultado é persistido no banco, ou seja, é durável.

3.1.2 Aplicação prática

Suponha que um artista morreu hoje e sua arte agora pertence ao governo.

```
-- Begin Transaction
BEGIN TRANSACTION;

-- Step 1: Update the artist's DateEnd to today
DECLARE @ArtistID INT = 1; -- Replace with the actual ArtistID
DECLARE @Today DATE = GETDATE(); -- Get today's date
```

```
UPDATE Artists
SET DateEnd = @Today
WHERE ID = @ArtistID;

-- Step 2: Update the owner of artworks authored by the artist to
    "Government"
UPDATE ArtisticObject
SET Owner = 'Government'
WHERE AuthorID = @ArtistID;

-- Commit the transaction
COMMIT;
```

3.2 Controle de concorrência

3.2.1 Fundamentação teórica

O controle de concorrência é a forma como a execução de transações simultâneas em um sistema com banco de dados multi-usuário será gerenciada para que o banco mantenha-se consistente. Para isso deve-se isolar as transações e as agendar a ordem em que vão ser executadas, Assim, uma transação não interfere com a outra ou duas transações não acessam o mesmo registro ao mesmo tempo.

3.2.2 Aplicação prática

Um exemplo simplista disso seria quando alguém quer performar a atualização descrita na sessão anterior, ao mesmo tempo que outro está tentando acessar quem é o dono do objeto.

```
-- User 1: Wants to Update Owner

-- Begin Transaction with Exclusive Lock
BEGIN TRANSACTION;

DECLARE @ArtistID INT = 1; -- Replace with the actual ArtistID
```

```
-- Obtain Exclusive Lock on the ArtisticObject table
SELECT * FROM ArtisticObject WHERE AuthorID = @ArtistID FOR UPDATE;

-- Update the owner
UPDATE ArtisticObject
SET Owner = 'Government'
WHERE AuthorID = @ArtistID;

-- Commit the transaction
COMMIT;
```

```
-- User 2: Wants to Query

-- Begin Transaction with Shared Lock
BEGIN TRANSACTION;

DECLARE @ArtistID INT = 1; -- Replace with the actual ArtistID

-- Obtain Shared Lock on the ArtisticObject table
SELECT * FROM ArtisticObject WHERE AuthorID = @ArtistID;

-- Query the owner
SELECT Owner FROM ArtisticObject WHERE AuthorID = @ArtistID;

-- Commit the transaction
COMMIT;
```

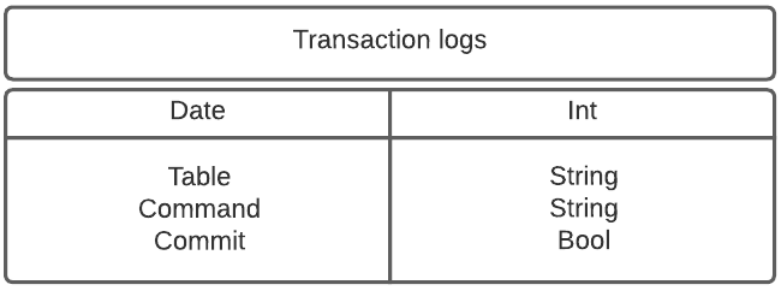
3.3 Recuperação de falhas

3.3.1 Fundamentação teórica

Habilidade do sistema de banco de dados em lidar e se recuperar em situação de falha, por exemplo falha de hardware, falha de software, e crashes mantendo a integridade do banco.

3.3.2 Aplicação prática

Suponha falta de energia enquanto alguém está atualizando a localidade de um grupo de objetos artísticos. No banco de dados implementado não há nada explicito que determine esta capacidade, contudo, o problema seria resolvido adicionando-se logs de transação no banco (diferente da tabela de logs de visitas já implementada no sistema), em que seriam registradas todas as operações a serem executadas no banco. Este registro de logs deveria ser armazenado em uma localidade diferente do banco de dados em si, para evitar o problema de falta de energia. Para evitar uso excessivo de memória, transações podem ser apagadas de tempos em tempos. Neste caso, então, quando o sistema voltasse a funcionar, os logs de transação seriam observados e todas as transações já commitadas seriam persistidas, enquanto as não commitadas seriam revertidas ou não executadas. Esta tabela de logs poderia ter o seguinte formato:



Além disso, para o código em si, pode-se determinar explicitamente que antes de toda transação AUTOCOMMIT está desabilitado, ou seja, as transações devem ser persistidas "manualmente". Supondo o exemplo a transação utilizada até agora de modificar o detentor de um grupo de objetos artísticos:

```
-- Disable AUTOCOMMIT mode to start a transaction explicitly
SET AUTOCOMMIT = 0; -- 0 means OFF

-- Begin Transaction
BEGIN;

DECLARE @ArtistID INT = 1; -- Replace with the actual ArtistID
```

```
UPDATE ArtisticObject
SET Owner = 'Government'
WHERE AuthorID = @ArtistID;

-- Commit the transaction
COMMIT;

-- Check if everything is fine or if there's an error
-- If no error, commit the transaction
COMMIT;

-- If there's an error or some condition is not met, roll back the
   transaction
-- ROLLBACK;
```

3.4 Normalização

3.4.1 Fundamentação teórica

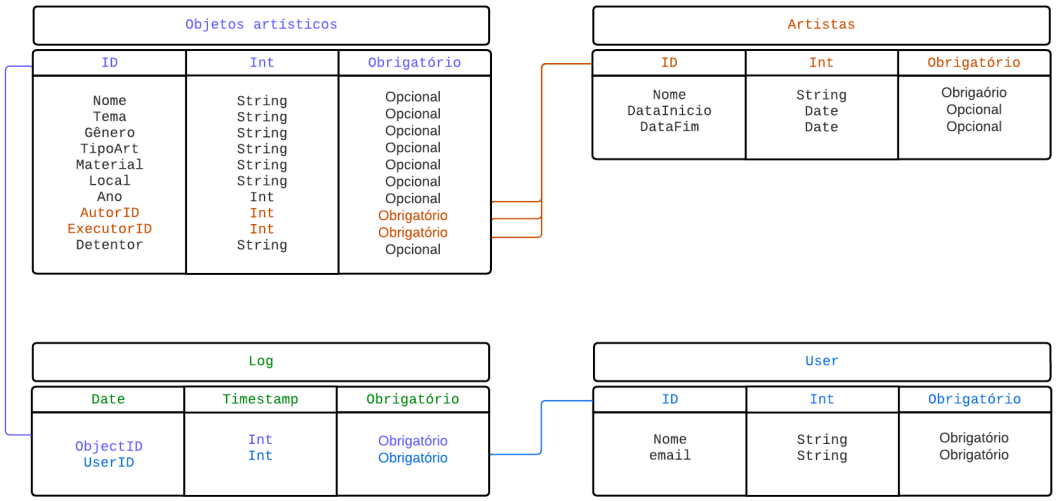
Processo de organização dos dados no banco que diminui redundância e melhora integridade dos dados e evitar anomalias. Há 3 formas principais de alcançar este objetivo:

- 1NF
  - Cada célula deve conter um valor individual (atômico);
  - Valores na mesma coluna devem ser do mesmo tipo de dados (domínio);
  - A ordem de armazenamento dos dados não importa
- 2NF
  - A tabela deve estar em 1NF.
  - Não podem haver dependências parciais. Um artibuto não pode depender de apenas parte da chave.

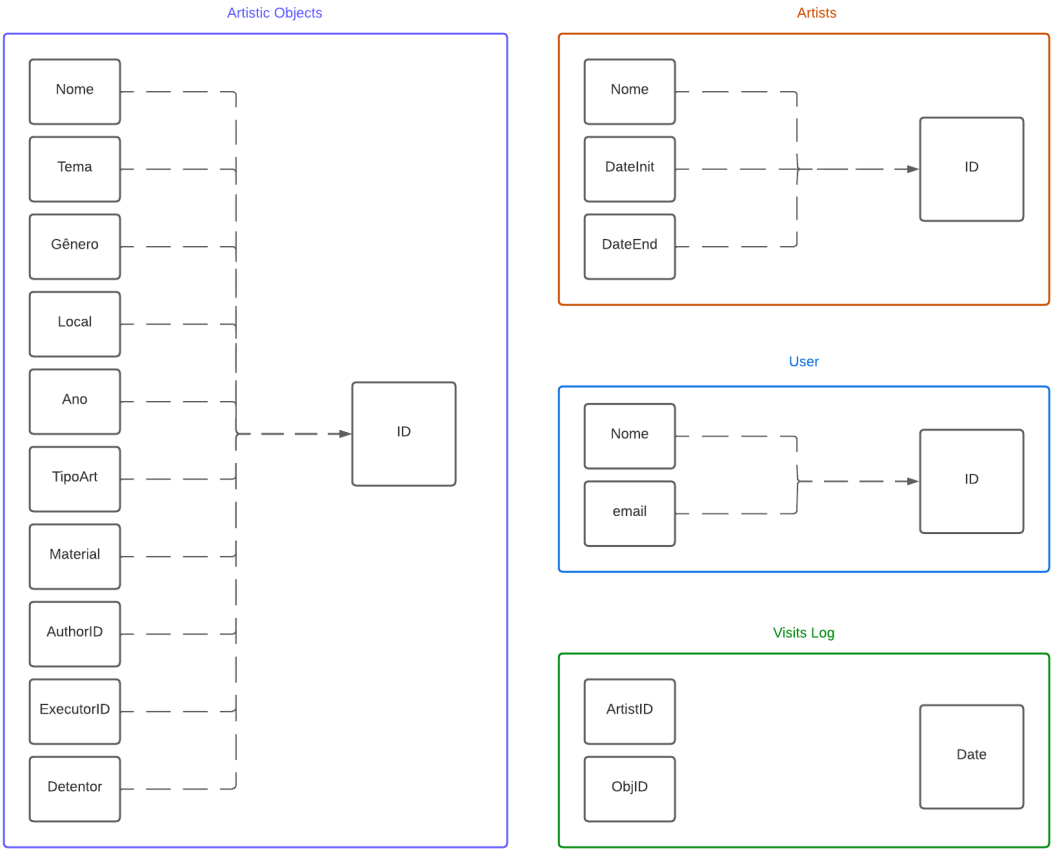
- 3NF
  - A tabela deve estar em 2NF.
  - Não podem haver dependências transitivas. Um atributo não pode depender de outro atributo (não chave).

3.4.2 Aplicação prática

No banco deste trabalho podemos perceber todas as formas de normalização citadas. Como podemos ver no esquema relacional do banco, cada atributo contém apenas um elemento em cada célula, e cada um destes elementos pertence ao mesmo domínio (1NF).



Além disso, o diagrama de dependências entre atributos abaixo, em que as linhas tracejadas explicitam as dependência entre os atributos, cada atributo depende somente de uma chave, não de outro atributo (3NF) ou parte de uma chave composta (2NF).



3.5 NoSQL (Não somente SQL)

3.5.1 Fundamentação teórica

Bancos NoSQL são aqueles não baseados no modelo relacional tradicional. Eles são desenvolvidos para gerenciar grande volume de dados não estruturados (ou semi-estruturados). Dependendo da aplicação, essa característica confere maior flexibilidade e escalabilidade para o sistema.

3.5.2 Aplicação prática

O banco de dados projetado para este trabalho é um exemplo muito claro de banco de dados relacional tradicional, e este é o melhor formato para a aplicação escolhida.

Contudo, poderíamos transformá-lo em NoSQL utilizando algo como MongoDB community version (que deve ser ativado antes de executar o arquivo a seguir). Neste caso, os dados mais se parecem arquivos JSON, linhas de formato livre.

Pode-se perceber que a criação do banco e inserção de dados acontece ao mesmo tempo, essa escolha foi feita porque a falta de estrutura nos dados faz com que a criação de uma tabela vazia seja inútil.

```
python create_and_populate_db_nosql.py
```

Nesta operação exportamos os dados em arquivos JSON, passo desnecessário e não recomendado, visto que é uma cópia do banco de dados e usa muita memória. Contudo, escolhi finalizar o arquivo com este export para que tenhamos uma visualização material de como estes dados estão armazenados, e como é diferente do formato no arquivo *db.db* gerado por SQL.

```
open artists.json
open artistic_objects.json
open users.json
open logs.json
```

3.6 Manipulação por linguagem hospedeira

3.6.1 Fundamentação teórica

Habilidade de interagir e manipular o banco de dados utilizando-se de uma linguagem de programação (host language), além do SQL puro.

3.6.2 Aplicação prática

Neste projeto a linguagem hospedeira é o Python, por meio dos arquivos python anexados neste diretório criamos e populamos bancos SQL e NoSQL. Este uso facilita a leitura e compreensão de ambos os modelos de banco de dados, assim como facilita uma interface com o usuário para que todas as operações requeridas até aqui sejam executadas com facilidade.

3.7 Aplicação em pesquisa

3.7.1 Fundamentação teórica

Aplicação de banco de dados em pesquisa científica. Pode ocorrer de maneira crua, apenas registrando dados relevantes, ou pode ser incrementado pelo uso de ferramentas de análise de dados e experimentos que facilitem a geração de conclusões baseadas nos dados do banco.

3.7.2 Aplicação prática

Neste projeto, se o banco fosse utilizado para pesquisa, algumas análises poderiam ser feitas. Aqui nosso intuito inicial era o de definir o(s) artista(s) mais relevantes(s), para que estes recebessem retornos financeiros por suas obras. Contudo, esta não é a única possibilidade deste banco, poderíamos analisar quais tipos de arte e temáticas são mais populares agora e ao longo do tempo; ou até mesmo quem são os detentores dos objetos mais populares e onde eles estão localizados, o que poderia facilitar por exemplo a execussão de um evento ou exposição. Além disso, poderíamos compreender em quais períodos tem-se maior volume de visitas, em horário do dia, dias da semana, meses ou estações do ano, ou anos comparados, entre outros.

No arquivo a seguir podemos ver queries relacionadas às perguntas científicas mencionadas:

`research_questions_examples.py`

Bibliografia

Elmasri, R., & Navathe, S. (2011). Fundamentals of Database Systems. Retrieved from https://books.google.com.br/books?id=ZdhAQgAACAAJ