EP de MAC0350

O objetivo dessa segunda etapa do projeto é adicionar uma nova funcionalidade para o armazenamento histórico dos serviços utilizados pelos usuários num laboratório. Esse histórico armazenado será utilizado para auditoria, análise e predição da utilização de serviços utilizados pelos usuários.

Autores:

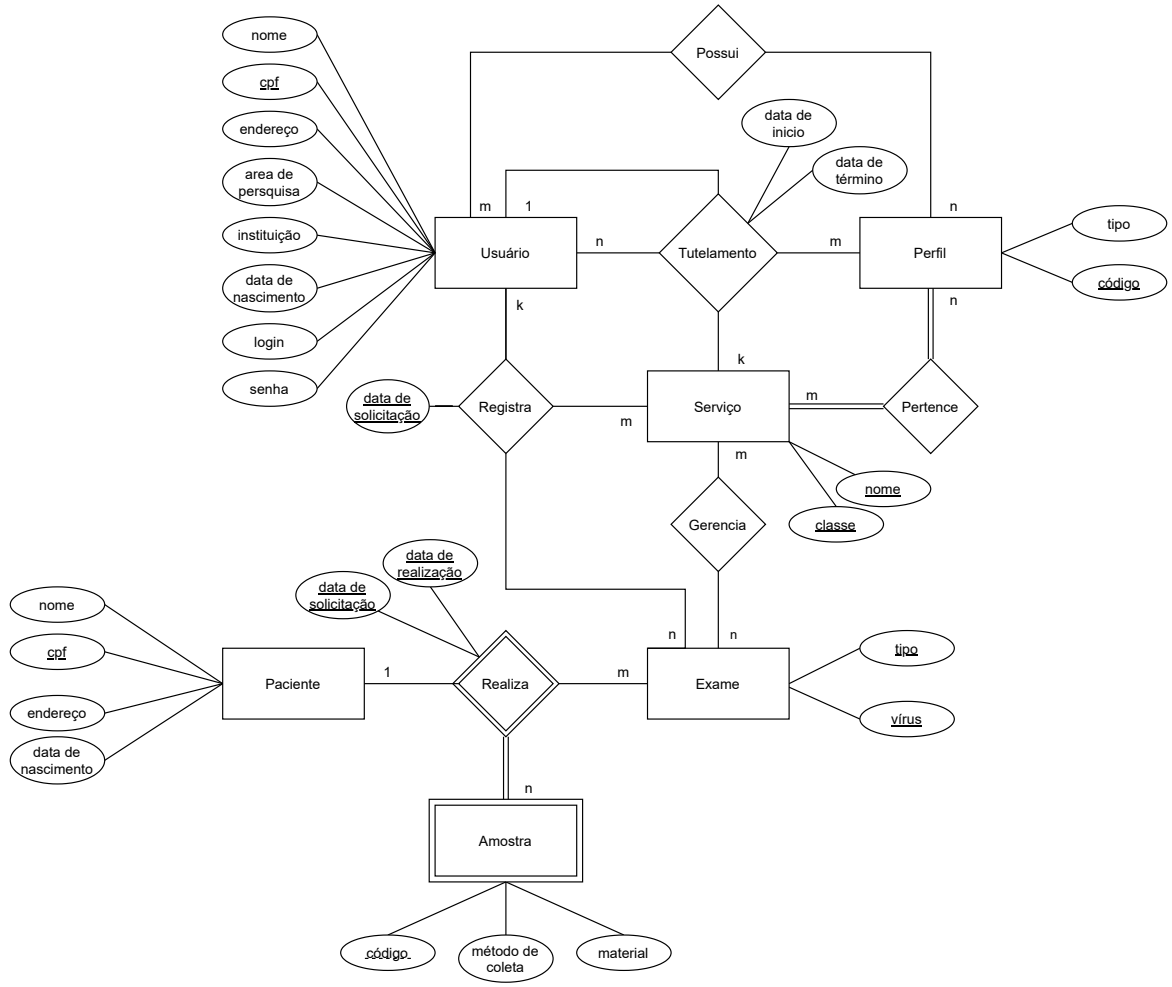
- Rodrigo Orem da Silva (8921590)
- Wander de Souza (10737011)

Modelos

Modelo Conceitual

Neste modelo, adicionamos o relacionamento `Registra`, que guarda as informações de histórico. Este é um relacionamento entre as entidades `Usuário`, `Serviço` e `Exame`, com um atributo-chave `data de solicitação`.

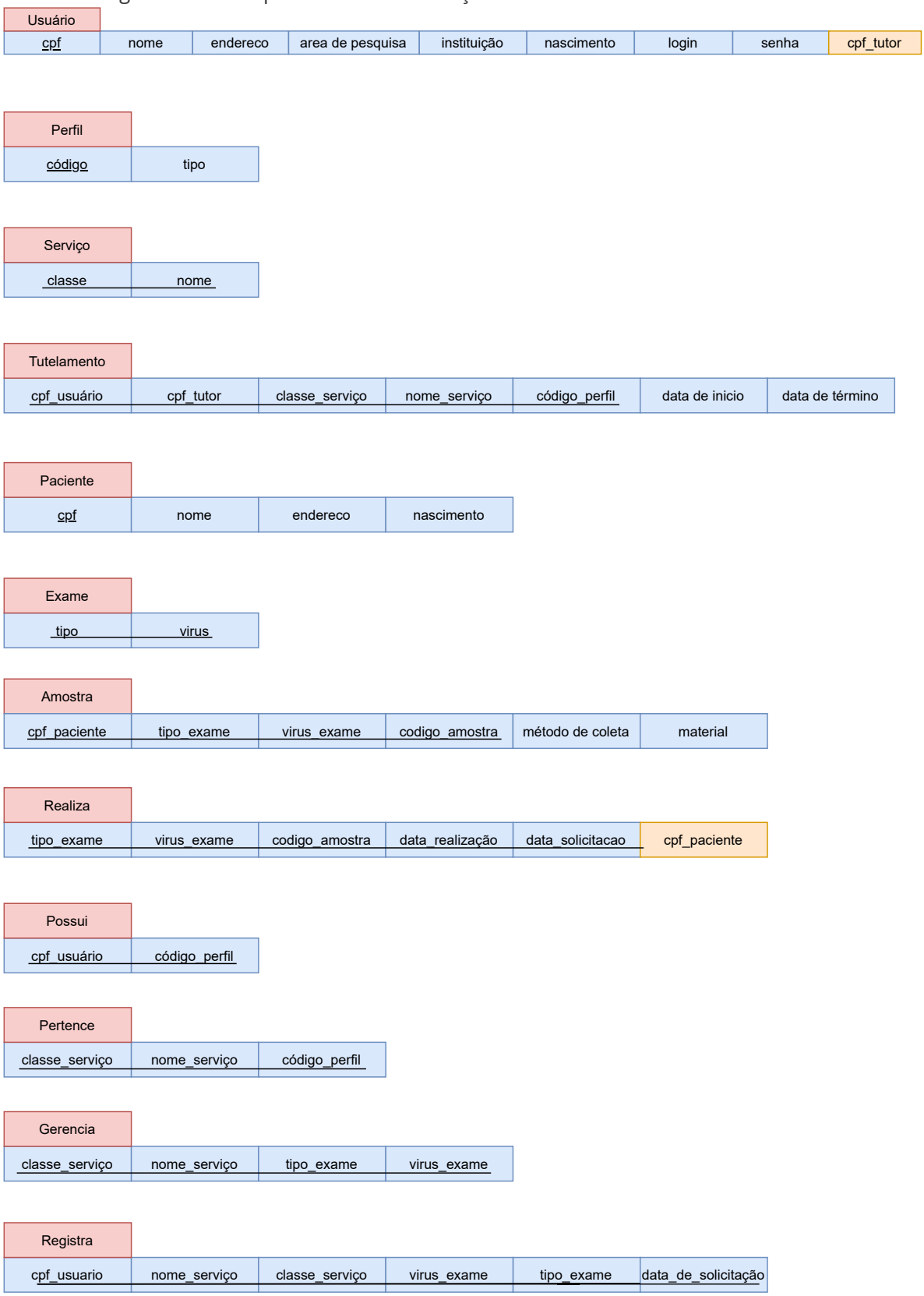
Além disso, adicionamos o atributo `data de solicitação` ao relacionamento triplo `realiza`, conforme orientado no fórum da disciplina. Essa mudança foi necessária porque não é possível relacionar uma tupla em `Registra` com um tupla em `Realiza`. Acreditamos que essa não é a melhor forma de resolver esse problema, porque a `data de solicitação` passa a ter múltiplas fontes, ao invés de uma Fonte Única da Verdade®, o que pode gerar inconsistência nos dados. Além disso, não é possível descobrir qual usuário solicitou determinado exame (não há relacionamento envolvendo usuário e realiza). Mas acatamos a sugestão do monitor no fórum por uma questão de comodidade.



MAC0350 2020 primeiro semestre
DER

Modelo Lógico

O modelo lógico é alterado para refletir as alterações descritas no modelo conceitual.



Scripts SQL

Os scripts SQL estão em `dist/`:

- `create.sql`: cria as tabelas, chaves e índices;
- `delete.sql`: remove todos os registros de todas as tabelas;
- `drop.sql`: remove todas as tabelas;
- `insert.sql`: popula todas as tabelas com dados simulados.

Consultas

4.1 Liste todos os exames realizados, com seus respectivos tipos, bem como os seus usuários com suas respectivas datas de solicitação e execução.

Nota: ao invés de usuário, essa query obtém o paciente, conforme retificação no fórum da disciplina.

> [...] o termo "usuário" se refere ao paciente, pois não temos uma ligação entre a tabela realiza e a tabela usuário.

```
SELECT
    paciente.nome as "Paciente",
    exame.tipo as "Tipo",
    exame.virus as "Vírus",
    realiza.data_de_solicitacao as "Data de solicitação",
    realiza.data_de_realizacao as "Data de execução"
FROM
    realiza
INNER JOIN exame
    ON realiza.id_exame = exame.id_exame
INNER JOIN paciente
    ON realiza.id_paciente = paciente.id_paciente
;
```

4.2 Liste os 5 exames realizados com maior eficiência (diferença entre data de execução e data de solicitação).

```
SELECT tipo,virus, realiza.data_de_realizacao - realiza.data_de_solicitacao as
"Tempo de espera"
FROM exame
INNER JOIN realiza
      ON realiza.id_exame = exame.id_exame
ORDER BY "Tempo de espera"
LIMIT 5;
```

4.3 Liste os serviços que podem ser utilizados pelos os usuários.

```
SELECT DISTINCT servico.nome, servico.classe
FROM servico
INNER JOIN pertence
      ON servico.id_servico=pertence.id_servico
INNER JOIN perfil
      ON pertence.id_perfil=perfil.id_perfil
INNER JOIN possui
      ON perfil.id_perfil = possui.id_perfil
INNER JOIN usuario
      ON possui.id_usuario = usuario.id_usuario;
```

4.4 Liste os serviços que podem ser utilizados por usuários tutelados (dependentes).

```sql
SELECT DISTINCT nome, classe
FROM servico
INNER JOIN tutelamento
      ON servico.id_servico=tutelamento.id_servico
;
```

4.5 Liste em ordem crescente o total de serviços utilizados agrupados pelos tipos de serviços disponíveis e pelo perfil dos usuários.

Nota: nessa consulta, a semântica adotada para o perfil segue a descrita pelo monitor no fórum:

> A semântica de perfil nesta query não é "o perfil que estava ativo na realização", mas, se me lembro bem (por favor, me corrijam caso eu esteja relembrando errado a aula), dos perfis que cada usuário possui, ou seja, uma linha no histórico com um serviço do tipo S1 e um usuário U que possui 2 perfis (P1 e P2) deve ser contabilizada tanto no agrupamento P1S1 quanto no agrupamento P2S1.

```sql
SELECT
    servico.nome,
    servico.classe,
    perfil.tipo as "Perfil",
    count(registra.id_servico) as "Quantidade"
FROM registra

INNER JOIN servico
    ON registra.id_servico = servico.id_servico

INNER JOIN usuario
    ON usuario.id_usuario = registra.id_usuario

INNER JOIN possui
    ON possui.id_usuario = usuario.id_usuario

INNER JOIN perfil
    ON perfil.id_perfil = possui.id_perfil

GROUP BY servico.id_servico, perfil.id_perfil

ORDER BY "Quantidade";
```

Build

Não é necessário fazer nenhuma operação descrita na seção para obter os scripts solicitados pelo EP, pois eles já estão nesse relatório ou na pasta `dist`. Esta seção apenas descreve o funcionamento do script para fins de documentação interna.

O projeto contém alguns scripts quem auxiliam na geração dos scripts em SQL. Para gerar tudo, apenas rode `make`, e os scripts em SQL serão gerados na pasta `dist/`.

Os scripts de inserção utilizaram um pequeno programa que fizemos em JavaScript que, com o auxílio da biblioteca `faker.js`, geram `INSERTs` a partir de um gerador de CPF e uma pequena base de dados de nomes e endereços fictícios, além de funções úteis para gerar datas em intervalos específicos. Esse programa não funciona para todas as tabelas, apenas para `usuario`, `possui` e `paciente`. Achamos melhor escrever os `INSERTs` das outras manualmente.

Gerando novos dados fictícios

Para gerar novos dados, basta instalar as dependências rodando `yarn` em `script/`, e então `make generate_faker`. Ao fazer isso, será necessário rescrever os scripts de inserção das outras tabelas manualmente, pois há muita lógica envolvendo chaves estrangeiras que não foram programadas (por exemplo, não faz muito sentido um pesquisador ser tutelado por visitante). O objetivo desse programa não é usar metaprogramação para fazer o EP inteiro, e sim automatizar a tarefa mais tediosa envolvendo algumas tabelas com muitos campos, ganhando produtividade.