Gestão de Dados Sobre a Vacinação da Covid-19

Ayla Santana Florêncio

Ariel Lima Abade Bandeira

Gabriel de Carvalho Silva

José Fagner Silva Junqueira

Tarcísio Almeida de Oliveira Araújo

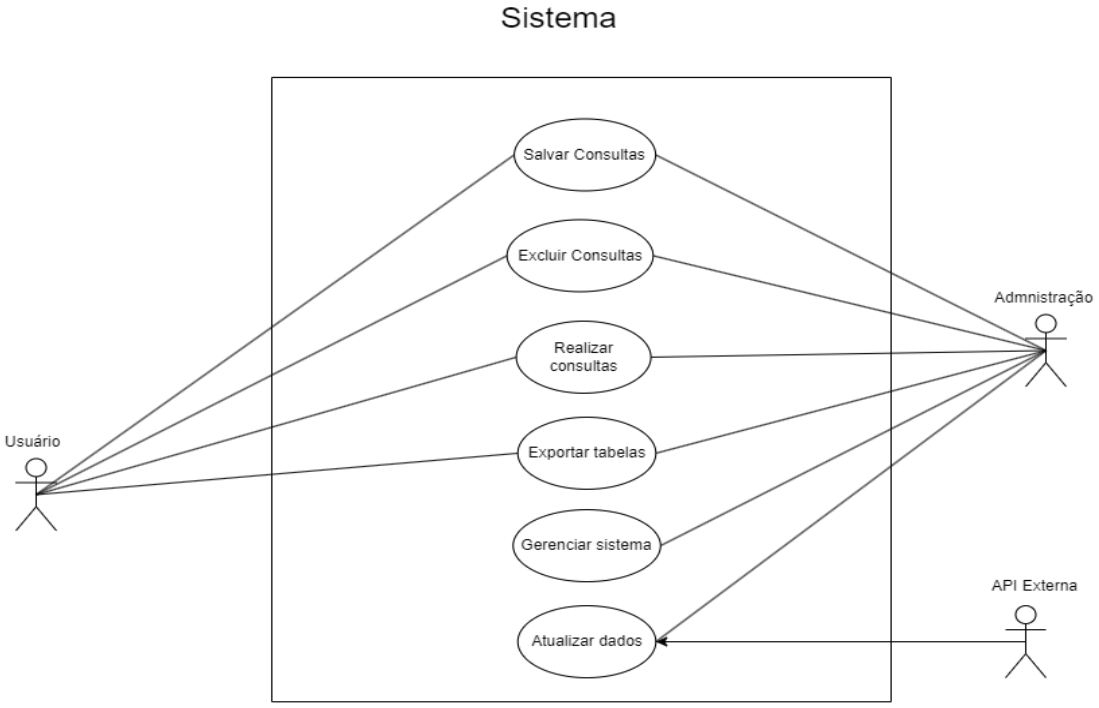
Introdução

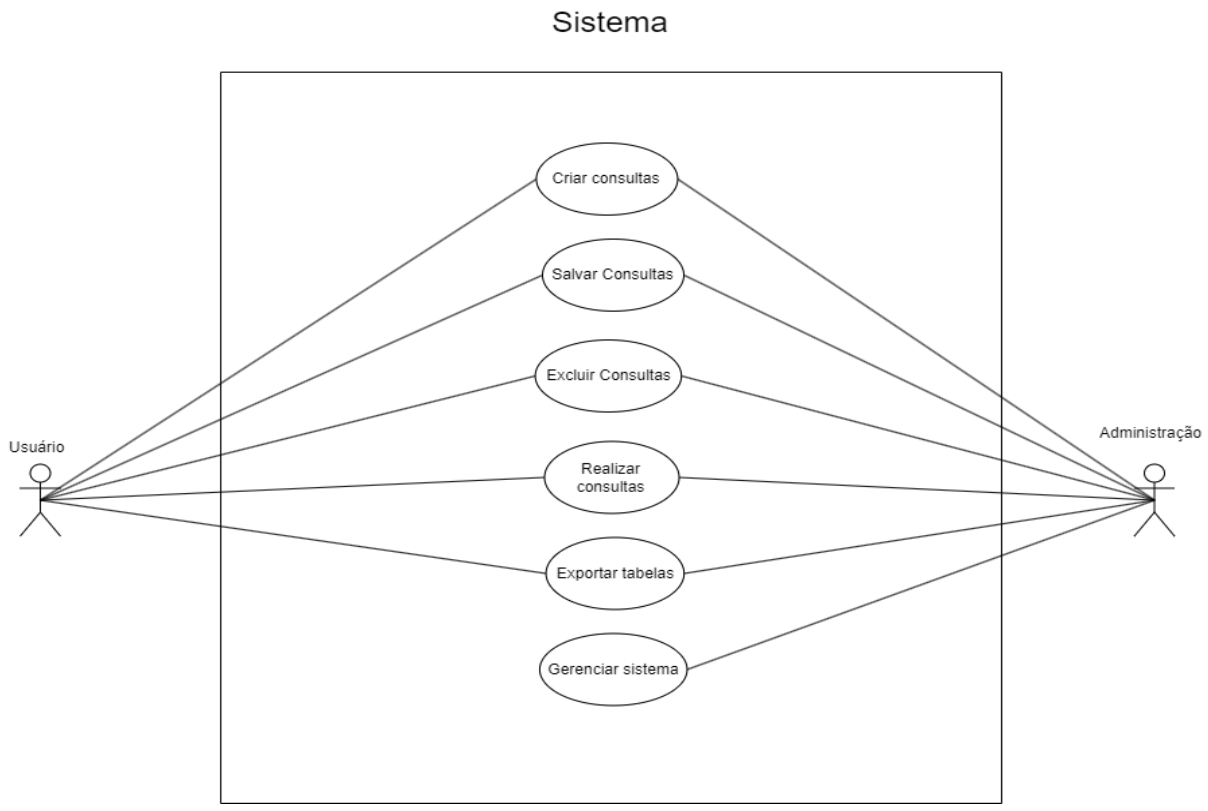
Após a elicitação de requisitos, iniciamos a etapa de modelagem que é essencial para refinamento e validação dos requisitos. Também foram elaborados modelos do banco de dados para garantir uma melhor visualização do funcionamento do sistema. Para tanto, produzimos os seguintes diagramas e modelos:

- Diagrama Use Case
- Diagrama de Classes
- Modelo Entidade Relacionamento
- Modelo Lógico de Banco de dados
- Script mysql

Diagrama Use Case

Como os atores do sistema foram previamente estabelecidos e as histórias dos usuários foram produzidas no documento anterior, a produção dos casos de uso trata-se do refinamento e da ligação desses elementos de maneira mais visual. Para tanto, foi utilizada a ferramenta de desenvolvimento draw.io.


Sistema
Salvar Consultas
Excluir Consultas
Realizar consultas
Exportar tabelas
Gerenciar sistema
Atualizar dados
Usuário
Admnistração
API Externa


Sistema
Criar consultas
Salvar Consultas
Excluir Consultas
Realizar consultas
Exportar tabelas
Gerenciar sistema
Usuário
Administração

As consultas são a união dos filtros de pesquisa com a finalidade de gerar um dashboard específico. No sistema, o usuário será capaz de realizar consultas, sem necessidade de login. Já os administradores poderão realizar outras ações com as consultas, como salvar, excluir e exportar consultas, porém terão login por questões de segurança. Além disso, também terão as funções administrativas de gerência do sistema.

Um dos refinamentos foi a remoção da necessidade de login por parte dos usuários, assim como as funções de salvar e excluir. Visto que o foco da aplicação é praticidade e usabilidade, manter um login retardaria o acesso às informações importantes e não impactaria na segurança do sistema.

Diagrama de Classes

A API selecionada possuía atributos em sua organização dos dados, por exemplo campos como “estabelecimento_uf’ e “estabelecimento_municipio_nome” trazem informações sobre um mesmo estabelecimento. Então, após a filtragem de quais campos interessam ao escopo do trabalho, reunimos eles em classes, definimos seus métodos, estabelecemos as relações e a cardinalidade.

As classes são:

- Cidadão;
- Dose;
- Vacina;
- Endereço;
- Estabelecimento;
- Admin.

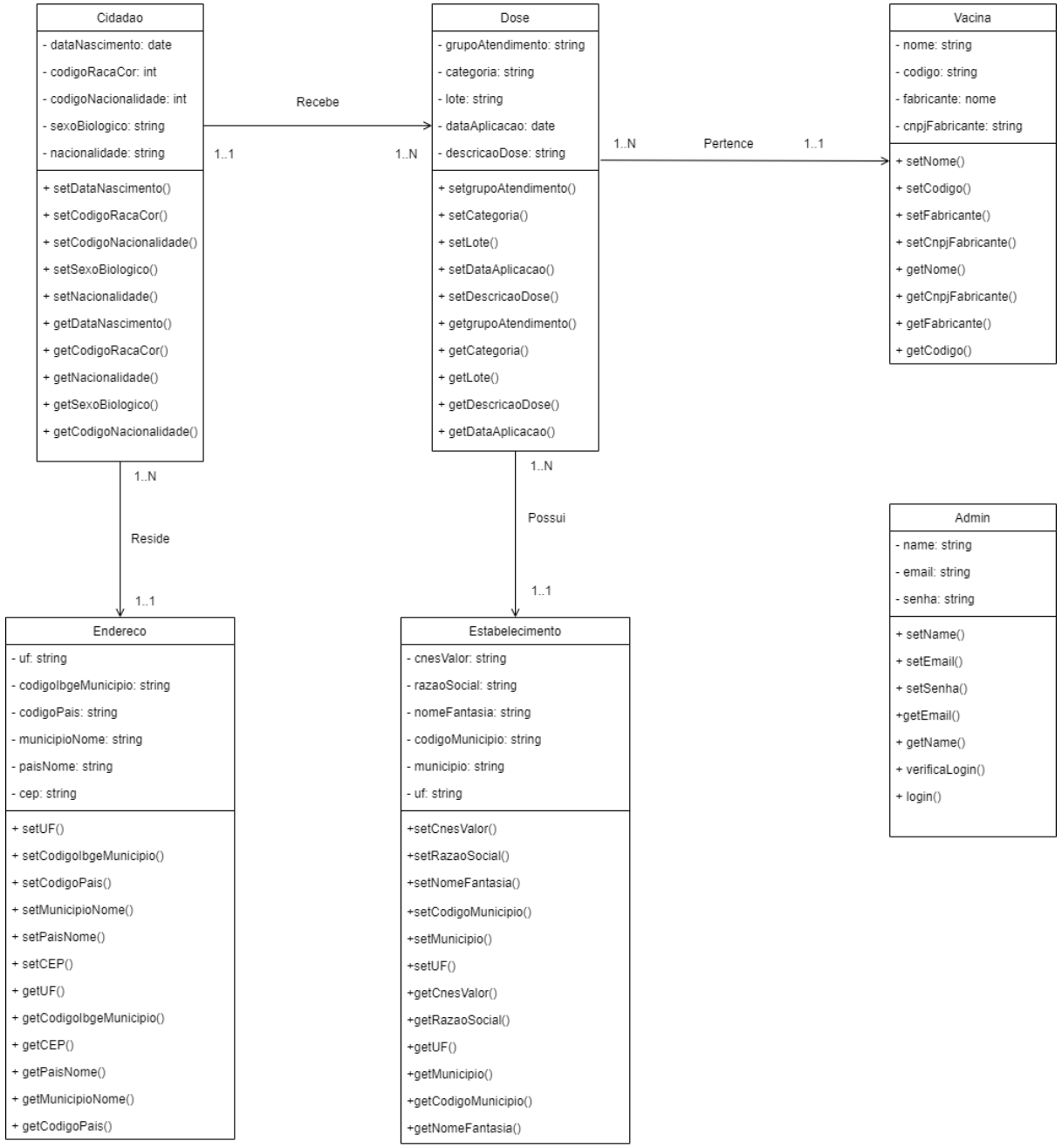
Os métodos são:

- getDoses()
- getEndereçoCompleto()
- getCidadao()
- getFabricante()
- getDosesAplicadas()
- getDosesDistribuidas()
- login()
- verificaLogin()
- updatePerfil()

As relações são:

- Um cidadão pode ter **uma ou mais** doses associadas.
- Um cidadão pode ter **somente um** endereço associado.
- Um endereço pode ter **um ou mais** cidadãos associados.
- Uma dose pode ter **somente um** cidadão associado.
- Uma dose pode ter **somente um** estabelecimento associado.
- Uma dose pode ter **somente uma** vacina associada.
- Um estabelecimento pode ter **uma ou mais** doses associadas.
- Uma vacina pode ter **uma ou mais doses** associadas.
- Admin **não possui** relações com as demais classes.

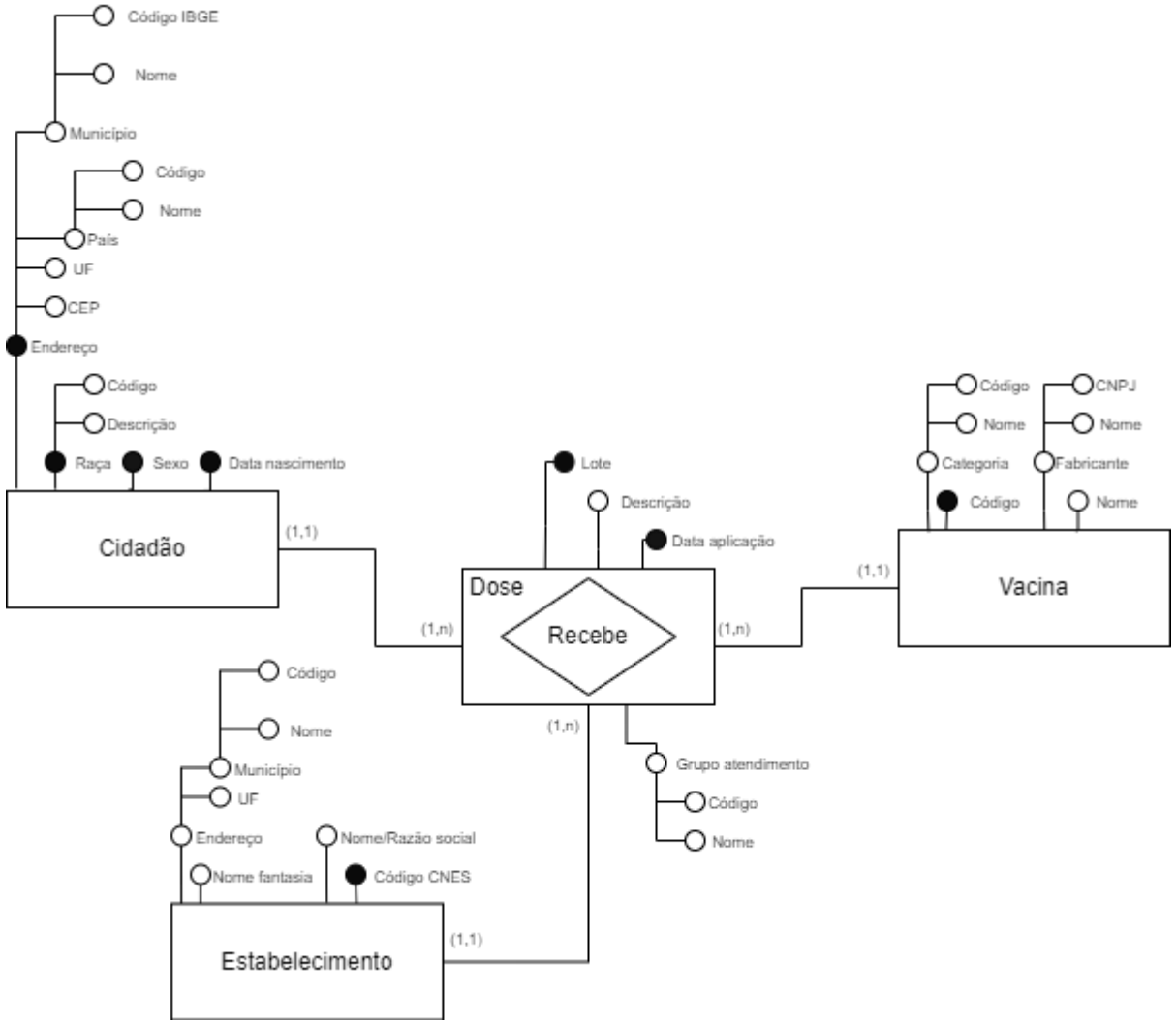
Utilizamos a ferramenta de desenvolvimento draw.io para unir esses elementos e compor o diagrama de classes a seguir:



Modelo DER

Para descrever os elementos fundamentais do sistema, elaboramos um modelo de entidade relacionamento no draw.io, com o objetivo de representar conceitualmente o funcionamento da aplicação.

Como o foco é representar conceitualmente, ele apresenta as informações de maneira mais simples. Sua organização de entidades e atributos menos complexa garante que a compreensão do sistema seja clara e direta.

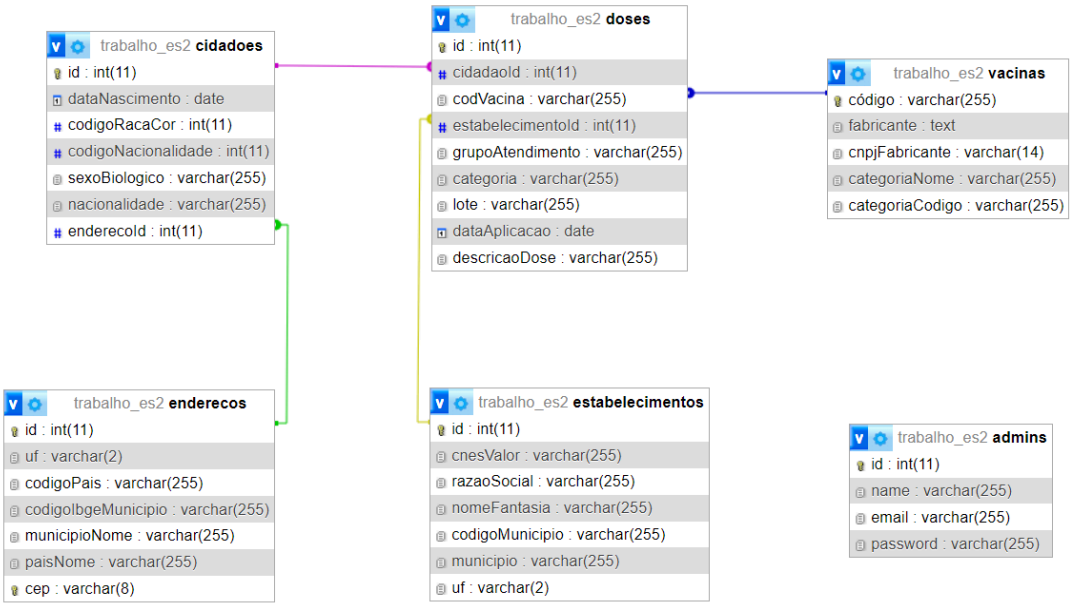


Modelo Lógico

Com o modelo lógico podemos detalhar os requisitos em termos de estrutura de dados (tabelas) e formar uma “tradução” desses requisitos para uma forma que SGBD compreenda, no nosso caso o mySQL.

É possível ter uma visão clara de como o banco de dados vai se comportar e garantimos que manutenções possam ser realizadas mais facilmente. Com o uso das chaves primárias e estrangeiras garantimos a associação das tabelas e com outras restrições de integridade provemos consistência e validação dos dados.

Como falado anteriormente, os dados da API possuía elementos que representavam uma coleção de dados, então algumas chaves primárias eram mais claras, como “paciente_id” que identificaria com unicidade um paciente específico e que utilizamos na tabela “cidadoes”. Utilizamos a ferramenta Phpmyadmin para confecção do modelo que pode ser visualizado abaixo:



Assim como temos id dos “cidadoes”, há também id para doses, endereços, estabelecimento e admin (vacinas possui código)

Script

A ferramenta de desenvolvimento possui uma forma de gerar um script para ser interpretado pelo banco de dados. Essa função utiliza as configurações de chave primária, chave estrangeira e restrições de integridade e traduz para um script mySQL.

```
-- Criação da tabela admins
CREATE TABLE admins (
  id int(11) NOT NULL,
  name varchar(255) NOT NULL,
  email varchar(255) NOT NULL,
  password varchar(255) NOT NULL,
  PRIMARY KEY (id)
```

```
);

-- Criação da tabela cidadoes
CREATE TABLE cidadoes (
  id int(11) NOT NULL,
  dataNascimento date NOT NULL,
  codigoRacaCor int(11) NOT NULL,
  codigoNacionalidade int(11) NOT NULL,
  sexoBiologico varchar(255) NOT NULL,
  nacionalidade varchar(255) NOT NULL,
  enderecoId int(11) NOT NULL,
  PRIMARY KEY (id),
  KEY enderecoId (enderecoId),
     CONSTRAINT  cidadoes_ibfk_1  FOREIGN  KEY  (enderecoId)
REFERENCES enderecos (id)
);

-- Criação da tabela doses
CREATE TABLE doses (
  id int(11) NOT NULL,
  cidadaoId int(11) NOT NULL,
  codVacina varchar(255) NOT NULL,
  estabelecimentoId int(11) NOT NULL,
  grupoAtendimento varchar(255) NOT NULL,
  categoria varchar(255) NOT NULL,
  lote varchar(255) NOT NULL,
  dataAplicacao date NOT NULL,
  descricaoDose varchar(255) NOT NULL,
  PRIMARY KEY (id),
  KEY codVacina (codVacina),
  KEY estabelecimentoId (estabelecimentoId),
  KEY cidadaoId (cidadaoId),
   CONSTRAINT doses_ibfk_1 FOREIGN KEY (codVacina) REFERENCES
vacinas (código),
    CONSTRAINT  doses_ibfk_2  FOREIGN  KEY  (estabelecimentoId)
REFERENCES estabelecimentos (id),
   CONSTRAINT doses_ibfk_3 FOREIGN KEY (cidadaoId) REFERENCES
cidadoes (id)
);

-- Criação da tabela enderecos
CREATE TABLE enderecos (
  id int(11) NOT NULL,
  uf varchar(2) NOT NULL,
  codigoPais varchar(255) NOT NULL,
  codigoIbgeMunicipio varchar(255) NOT NULL,
  municipioNome varchar(255) NOT NULL,
  paisNome varchar(255) NOT NULL,
```

```
  cep varchar(8) NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY cep (cep)
);

-- Criação da tabela estabelecimentos
CREATE TABLE estabelecimentos (
  id int(11) NOT NULL,
  cnesValor varchar(255) NOT NULL,
  razaoSocial varchar(255) NOT NULL,
  nomeFantasia varchar(255) NOT NULL,
  codigoMunicipio varchar(255) NOT NULL,
  municipio varchar(255) NOT NULL,
  uf varchar(2) NOT NULL,
  PRIMARY KEY (id)
);

-- Criação da tabela vacinas
CREATE TABLE vacinas (
  código varchar(255) NOT NULL,
  fabricante text NOT NULL,
  cnpjFabricante varchar(14) NOT NULL,
  categoriaNome varchar(255) NOT NULL,
  categoriaCodigo varchar(255) NOT NULL,
  PRIMARY KEY (código),
  UNIQUE KEY código (código)
);

COMMIT;
```

Elementos como “NOT NULL”, “PRIMARY KEY” ou “UNIQUE KEY” são as restrições de unicidade configuradas previamente no modelo lógico, que ao passar pela tradução são expressas em formato de código.