

SUMÁRIO

1. INTRODUÇÃO	5
2. DESENVOLVIMENTO	6
2.1. - INSTALAÇÃO E CONFIGURAÇÃO MYSQL COMMUNITY SERVER	7
2.2. - ETAPA 1: CRIAÇÃO DA ESTRUTURA DO BANCO DE DADOS E TABELAS	9
2.3. - ETAPA 2: INSERÇÃO DE DADOS	13
2.4. - ETAPA 3: CRIAÇÃO DA VISÃO (VIEW)	15
3. RESULTADOS	16
4. CONCLUSÃO	20
5. REFERÊNCIAS	21

LISTA DE ILUSTRAÇÕES

Figura 1.	Criando conta de usuário root	7
Figura 2.	Acesso ao MySQL Community Server	8
Figura 3.	Interface MySQL Community Server	8
Figura 4.	Criação da estrutura do banco de dados	9
Figura 5.	Tabela Estado	9
Figura 6.	Tabela Municipio	10
Figura 7.	Tabela Cliente	11
Figura 8.	Tabela ContaReceber	12
Figura 9.	Inserção de dados na tabela Estado	13
Figura 10.	Inserção de dados na tabela Municipio	14
Figura 11.	Inserção de dados na tabela Cliente	14
Figura 12.	Inserção de dados na tabela ContaReceber	14
Figura 13.	Criação da VIEW	15
Figura 14.	Adicionando o Banco de Dados na Plataforma	16
Figura 15.	Consulta de contas a receber	17
Figura 16.	Consulta de clientes cadastrados	17
Figura 17.	Consulta de Estado	18
Figura 18.	Consulta de Município	18
Figura 19.	Diagrama do Banco de Dados	19

1. INTRODUÇÃO

Este trabalho detalha uma atividade prática de Programação e Desenvolvimento de Banco de Dados, centrada na criação de um sistema de gerenciamento de banco de dados utilizando o MySQL Server por meio do software MySQL Workbench. O objetivo principal é criar um banco de dados usando linguagem SQL e realizar operações de manipulação e acesso aos dados.

A atividade foi dividida em três etapas distintas. Na primeira etapa, criamos uma base de dados "Loja" seguindo um diagrama entidade-relacionamento, utilizando comandos DDL e aderindo às regras de implementação das estruturas e relacionamentos. Isso incluiu a definição de chaves primárias como autoincremento e o uso de ENUM para o campo "Situação" na tabela "ContaReceber".

Em seguida, na segunda etapa elaboramos um script "inserir.sql", contendo comandos DML para inserir dados em todas as tabelas existentes na base de dados.

Finalmente, na terceira etapa, desenvolvemos o script "consulta.sql" com comandos DQL, que inclui uma visão (VIEW) para retornar todas as contas pendentes de pagamento, fornecendo informações como ID da conta, nome e CPF do cliente associado, data de vencimento e valor da conta.

Para a criação da infraestrutura do banco, optamos pelo MySQL Community Server e MySQL Workbench devido à ampla eficiência dessas ferramentas. O MySQL Server é uma das ferramentas mais amplamente utilizadas para gerenciamento de bancos de dados relacionais, oferecendo uma variedade de recursos e funcionalidades para desenvolvedores e administradores de banco de dados. Por sua vez, o MySQL Workbench é uma ferramenta de modelagem visual e administração de banco de dados que simplifica o processo de design, desenvolvimento e manutenção de bancos de dados MySQL.

Em resumo, este portfólio não apenas registra as etapas da atividade, mas também serve como uma fonte de aprendizado sobre os fundamentos de ferramentas de criação e modelagem de banco de dados, bem como sobre o uso prático da linguagem SQL e das linguagens do MySQL.

2. DESENVOLVIMENTO

Para iniciar nossa jornada, vamos abordar o desafio de criar um banco de dados utilizando o MySQL Workbench e o MySQL Community Server, com base no problema proposto:

Objetivos - Criar um banco de dados utilizando a linguagem SQL e realizar operações de manipulação e acesso aos dados.

Atividade proposta

- Criação da estrutura de um banco de dados (tabelas) com a linguagem SQL por meio de um diagrama entidade-relacionamento pré-definido.
- Inserir dados no banco de dados criado.
- Consultar os dados armazenados por meio da criação de uma visão (View).
- Criar um relatório no final da atividade.

Procedimentos para a atividade

Etapas 1 - Crie uma base de dados chamada "Loja" com o MySQL Server por meio do software MySQL Workbench. Adicione as estruturas de dados neste banco, utilizando os comandos de definição de dados (DDL) da linguagem SQL, e respeitando o modelo definido no DER da Figura a seguir.

*** Na criação do banco de dados da figura, respeite as seguintes regras:**

- As chaves primárias devem ser colocadas todas como autoincremento.
- Respeite os relacionamentos, tipos, precisões e restrições de não nulo.
- O campo "Situação" da tabela "ContaReceber" deve ser do tipo ENUM e possuir apenas os valores 1, 2 ou 3, sendo 1 – Conta registrada, 2 – Conta cancelada, 3 – Conta paga.

Etapas 2 - Crie um script chamado "inserir.sql" contendo os comandos de manipulação (DML), com o objetivo de popular todas as tabelas existentes na base de dados (ou seja, inserir dados nas tabelas!). Insira ao menos três registros por tabela.

Etapas 3 - Por meio dos comandos de consulta (DQL) da linguagem SQL, elabore um script chamado "consulta.sql" que irá conter uma visão (VIEW) que retornará todas as contas que ainda não foram pagas (Situação = 1), devendo conter as seguintes informações:

- ID da conta a receber

RELATÓRIO DE AULA PRÁTICA

Programação e Desenvolvimento de Banco de Dados

- *Nome e CPF do Cliente associado à conta*
- *Data de vencimento da conta*
- *Valor da conta*

2.1. – INSTALAÇÃO E CONFIGURAÇÃO MYSQL COMMUNITY SERVER

Considerando o exposto, acessamos a plataforma do software MySQL Workbench disponível para download no MySQL Community Downloads através do link <https://dev.mysql.com/downloads/workbench/>. Com o software MySQL já previamente instalado e configurado, apenas baixamos o pacote de instalação do MySQL Community Server.

Após concluir o download do pacote, iniciamos a sua instalação e durante o processo de configuração, estabelecemos uma conta de usuário root, definindo um login e senha na seção de Contas e Funções.

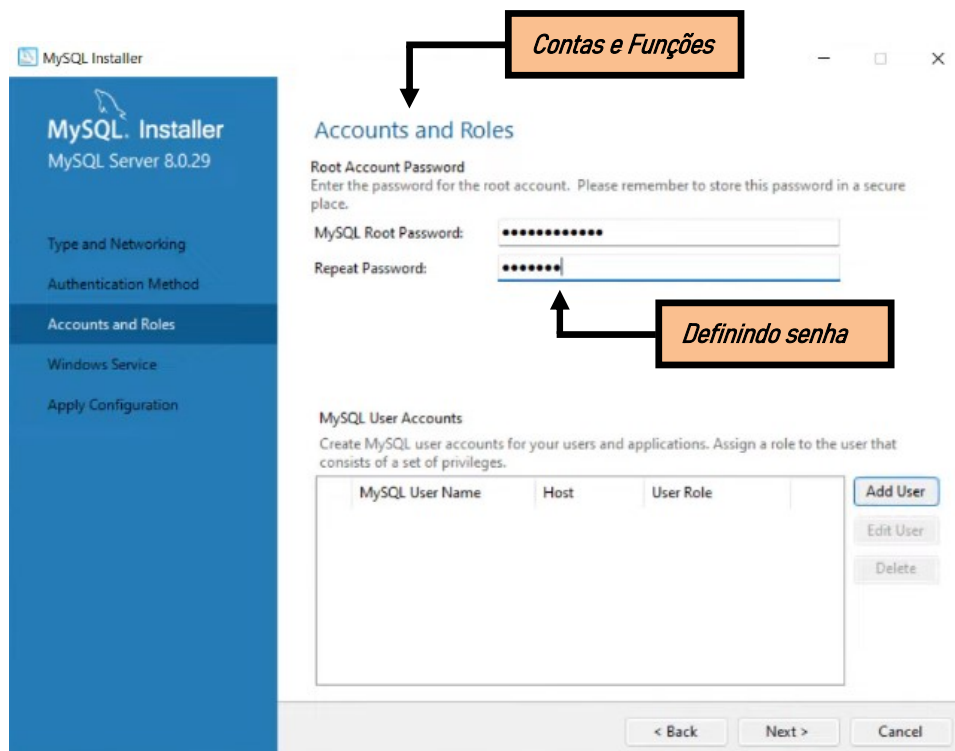


Figura 1. | Criando conta de usuário root

Essa etapa é importante para garantir a segurança e o acesso adequado ao MySQL Server, fornecendo as permissões necessárias para a administração e operação do banco de dados.

Logo em seguida, dentro do ambiente integrado do MySQL Workbench, configuramos e inicializamos o MySQL Community Server.

RELATÓRIO DE AULA PRÁTICA

Programação e Desenvolvimento de Banco de Dados

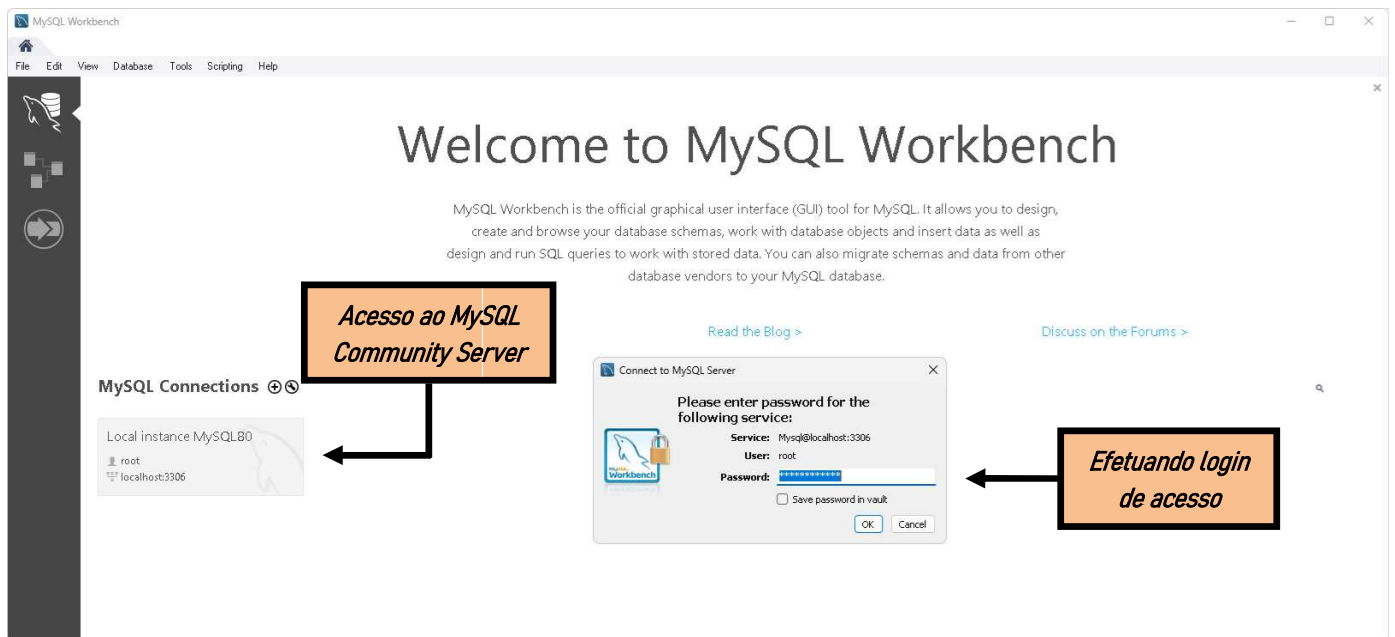


Figura 2. | Acesso ao MySQL Community Server

Na imagem seguinte, destacamos a Interface intuitiva do MySQL Community Server, que oferece um ambiente de desenvolvimento de banco de dados acessível e amigável para nosso projeto.

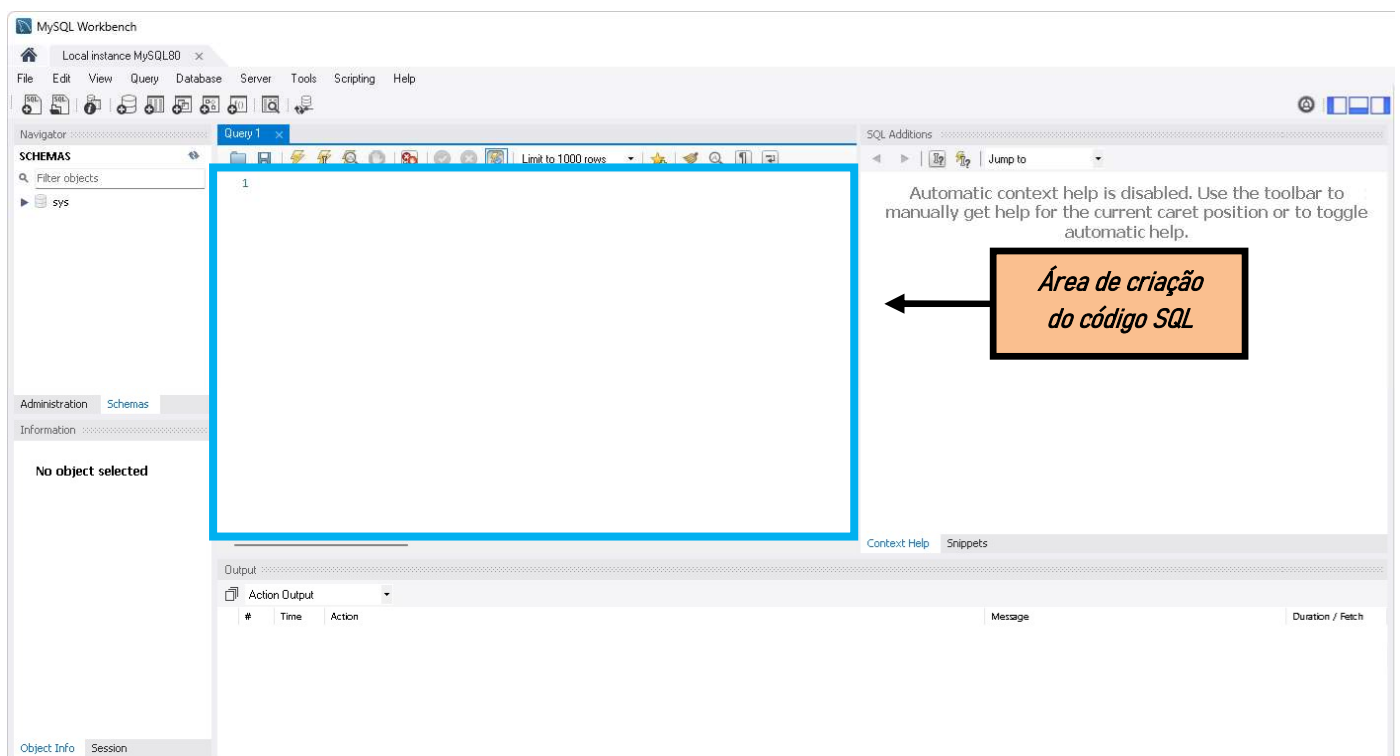


Figura 3. | Interface MySQL Community Server

2.2. – ETAPA 1: CRIAÇÃO DA ESTRUTURA DO BANCO DE DADOS E TABELAS

Como ponto de partida, iniciamos criando a estrutura da nossa base de dados chamada "Loja". Esta etapa inaugural estabelece os alicerces do nosso sistema de gerenciamento de banco de dados, definindo a estrutura primordial que sustentará todas as operações subsequentes.

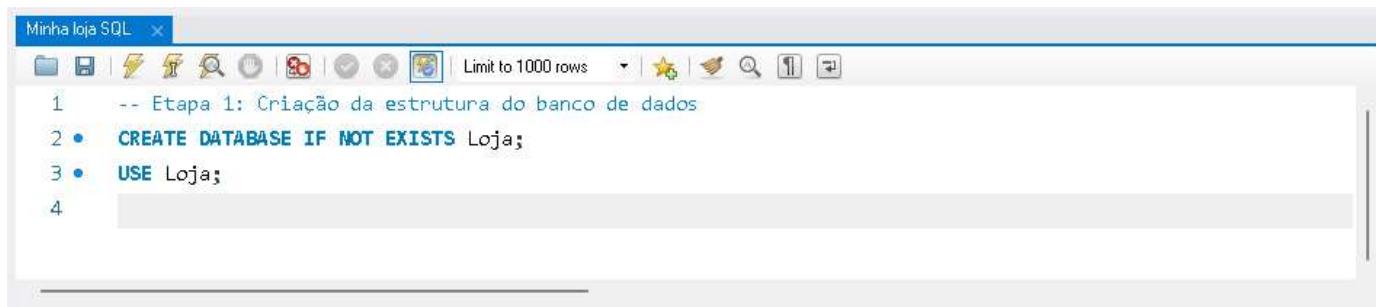


Figura 4. | Criação da estrutura do banco de dados

Na imagem acima, empregamos o comando "CREATE DATABASE" para criar a base de dados, complementado pelo uso do "IF NOT EXISTS", prevenindo erros e exceções que podem surgir caso tentemos criar um banco de dados ou tabela que já exista. Embora não seja uma prática obrigatória, é comum e altamente recomendável em muitos contextos, especialmente ao escrevermos scripts SQL ou instruções DDL (Data Definition Language) que serão executadas repetidamente. Isso confere robustez e segurança aos nossos scripts SQL, evitando interrupções indesejadas no fluxo de trabalho decorrente de tentativas de criação de objetos duplicados.

Continuando com o nosso processo, avançaremos para a elaboração das tabelas. Inicialmente, começaremos com a tabela "Estado", destinada a armazenar informações sobre os estados, tais como um ID único para cada estado, o nome do estado e a sua sigla (UF). É crucial considerar a ordem das tabelas para garantir que todas as restrições de chaves estrangeiras sejam respeitadas durante a criação do banco de dados.

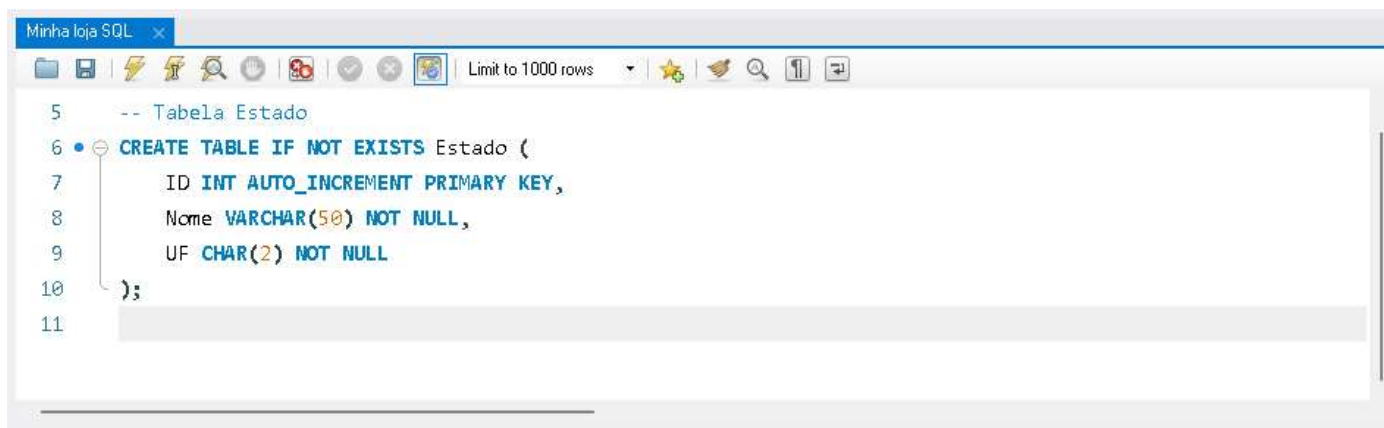


Figura 5. | Tabela Estado

RELATÓRIO DE AULA PRÁTICA

Programação e Desenvolvimento de Banco de Dados

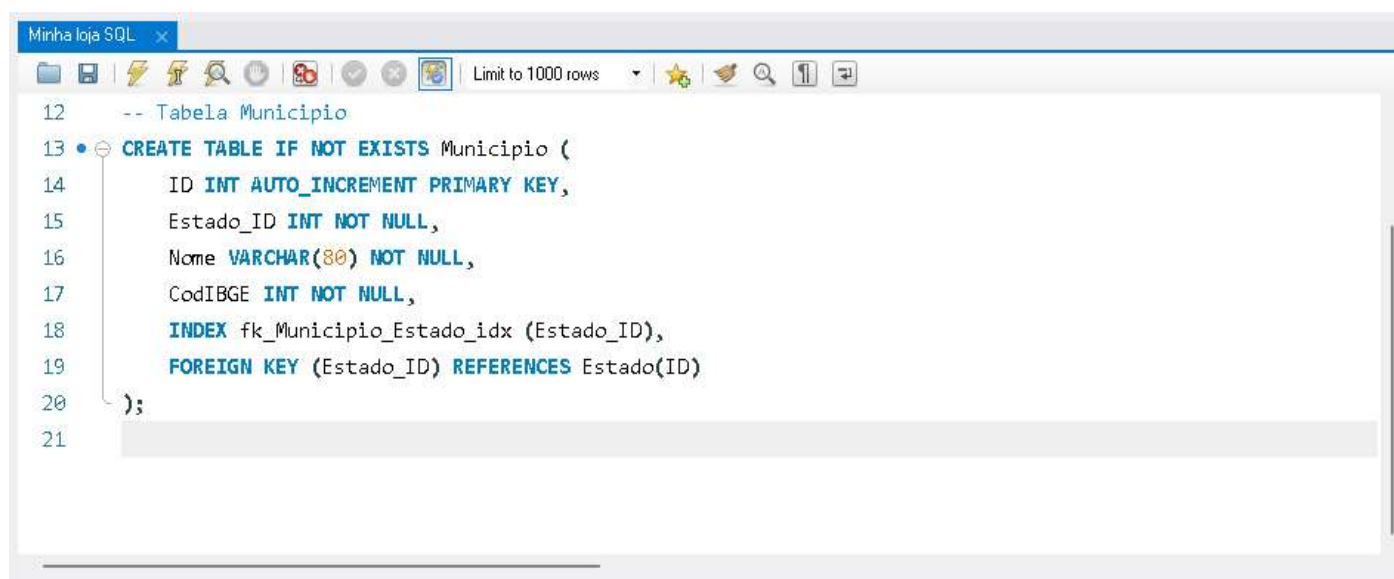
Seguindo as diretrizes da tarefa proposta, iniciamos a criação da tabela "Estado", utilizando o comando "CREATE TABLE IF NOT EXISTS". Abaixo, apresentamos os principais aspectos da definição desta tabela:

"ID INT AUTO_INCREMENT PRIMARY KEY": Definimos a coluna "ID" como chave primária do tipo INT, onde os valores são incrementados automaticamente para garantir a unicidade.

"Nome VARCHAR(50) NOT NULL": Definimos a coluna "Nome" para armazenar o nome do estado, com um limite de até 50 caracteres e sem valores nulos.

"UF CHAR(2) NOT NULL": Definimos a coluna "UF" para armazenar a sigla do estado, com um comprimento fixo de dois caracteres e sem permitir valores nulos.

Posteriormente, estabelecemos a tabela "Municipio", aderindo aos princípios observados na criação da tabela "Estado", conforme descrito nos seguintes tópicos abaixo:



```
12  -- Tabela Municipio
13  CREATE TABLE IF NOT EXISTS Municipio (
14      ID INT AUTO_INCREMENT PRIMARY KEY,
15      Estado_ID INT NOT NULL,
16      Nome VARCHAR(80) NOT NULL,
17      CodIBGE INT NOT NULL,
18      INDEX fk_Municipio_Estado_idx (Estado_ID),
19      FOREIGN KEY (Estado_ID) REFERENCES Estado(ID)
20  );
21
```

Figura 6.1 Tabela Municipio

"ID INT AUTO_INCREMENT PRIMARY KEY": Definimos a coluna "ID" como chave primária do tipo INT, incrementada automaticamente para garantir a unicidade.

"Estado_ID INT NOT NULL": Definimos a coluna "Estado_ID" do tipo INT para armazenar o ID do estado correspondente, sem aceitar valores nulos.

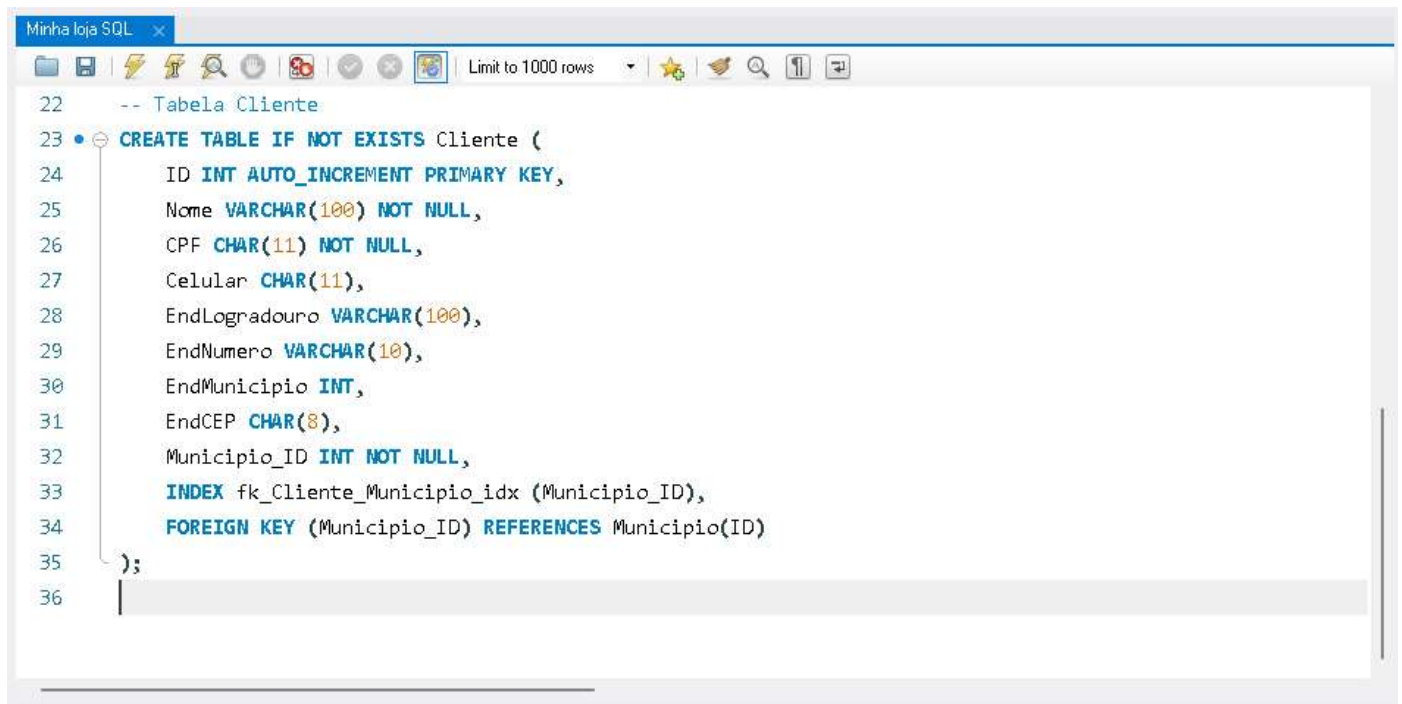
"Nome VARCHAR(80) NOT NULL": Definimos a coluna "Nome" para armazenar o nome do município, com limite de 80 caracteres e sem valores nulos.

"CodIBGE INT NOT NULL": Definimos a coluna "CodIBGE" do tipo INT para armazenar o código do município do IBGE, sem aceitar valores nulos.

"INDEX fk_Municipio_Estado_idx (Estado_ID)": Criamos um índice na coluna "Estado_ID" para otimizar consultas.

"FOREIGN KEY (Estado_ID) REFERENCES Estado(ID)": Estabelecemos uma chave estrangeira para garantir a integridade referencial entre as tabelas "Municipio" e "Estado".

Continuando, elaboramos a tabela "Cliente", seguindo os mesmos preceitos adotados nas tabelas anteriores, conforme exposto nos seguintes tópicos:



```
22  -- Tabela Cliente
23  CREATE TABLE IF NOT EXISTS Cliente (
24      ID INT AUTO_INCREMENT PRIMARY KEY,
25      Nome VARCHAR(100) NOT NULL,
26      CPF CHAR(11) NOT NULL,
27      Celular CHAR(11),
28      EndLogradouro VARCHAR(100),
29      EndNumero VARCHAR(10),
30      EndMunicipio INT,
31      EndCEP CHAR(8),
32      Municipio_ID INT NOT NULL,
33      INDEX fk_Cliente_Municipio_idx (Municipio_ID),
34      FOREIGN KEY (Municipio_ID) REFERENCES Municipio(ID)
35  );
36
```

Figura 7. | Tabela Cliente

ID INT AUTO_INCREMENT PRIMARY KEY: Definimos a coluna "ID" como chave primária do tipo INT, onde os valores são incrementados automaticamente para garantir a unicidade.

Nome VARCHAR(100) NOT NULL: Definimos a coluna "Nome" para armazenar o nome do cliente, com um máximo de 100 caracteres e sem permitir valores nulos.

CPF CHAR(11) NOT NULL: Definimos a coluna "CPF" para armazenar o CPF do cliente, com um comprimento fixo de 11 caracteres e sem permitir valores nulos.

Celular CHAR(11): Definimos a coluna "Celular" para armazenar o número de celular do cliente, com um comprimento fixo de 11 caracteres e permitindo valores nulos.

EndLogradouro VARCHAR(100): Definimos a coluna "EndLogradouro" para armazenar o logradouro do endereço do cliente, com um máximo de 100 caracteres e permitindo valores nulos.

EndNumero VARCHAR(10): Definimos a coluna "EndNumero" para armazenar o número do endereço do cliente, com um máximo de 10 caracteres e permitindo valores nulos.

EndMunicipio INT: Definimos a coluna "EndMunicipio" do tipo INT para armazenar o ID do município do endereço do cliente e permitindo valores nulos.

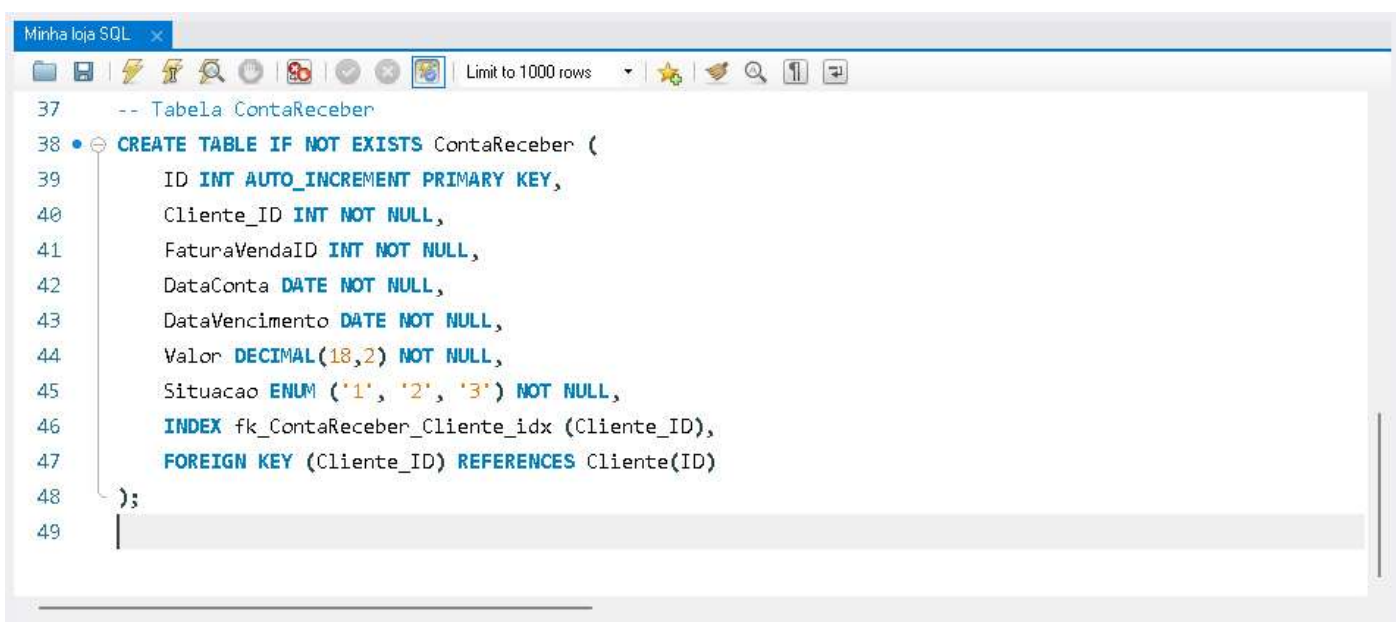
EndCEP CHAR(8): Definimos a coluna "EndCEP" para armazenar o CEP do endereço do cliente, com um comprimento fixo de 8 caracteres e permitindo valores nulos.

Municipio_ID INT NOT NULL: Definimos a coluna "Municipio_ID" do tipo INT para armazenar o ID do município do cliente, sem permitir valores nulos.

INDEX fk_Cliente_Municipio_idx (Municipio_ID): Criamos um índice na coluna "Municipio_ID" para otimizar consultas.

FOREIGN KEY (Municipio_ID) REFERENCES Municipio(ID): Estabelecemos uma chave estrangeira na coluna "Municipio_ID", referenciando a coluna "ID" da tabela "Municipio" para garantir a integridade referencial.

Por fim, implementamos a tabela "ContaReceber", mantendo a consistência com os princípios e padrões adotados nas tabelas precedentes, como detalhado a seguir:



```
37 -- Tabela ContaReceber
38 CREATE TABLE IF NOT EXISTS ContaReceber (
39     ID INT AUTO_INCREMENT PRIMARY KEY,
40     Cliente_ID INT NOT NULL,
41     FaturaVendaID INT NOT NULL,
42     DataConta DATE NOT NULL,
43     DataVencimento DATE NOT NULL,
44     Valor DECIMAL(18,2) NOT NULL,
45     Situacao ENUM ('1', '2', '3') NOT NULL,
46     INDEX fk_ContaReceber_Cliente_idx (Cliente_ID),
47     FOREIGN KEY (Cliente_ID) REFERENCES Cliente(ID)
48 );
49
```

Figura 8. | Tabela ContaReceber

ID INT AUTO_INCREMENT PRIMARY KEY: Definimos a coluna "ID" como a chave primária do tipo INT, com valores incrementados automaticamente para garantir a unicidade.

Cliente_ID INT NOT NULL: Armazenamos o ID do cliente do tipo INT associado à conta a receber, não permitindo valores nulos.

FaturaVendaID INT NOT NULL: Guardamos o ID da fatura de venda do tipo INT relacionada à conta, também sem aceitar valores nulos.

DataConta DATE NOT NULL: Registramos a data em que a conta foi registrada, sem permitir valores nulos.

DataVencimento DATE NOT NULL: Armazenamos a data de vencimento da conta, sem aceitar valores nulos.

Valor DECIMAL(18,2) NOT NULL: Guardamos o valor da conta, permitindo números decimais e sem permitir valores nulos.

Situacao ENUM ('1', '2', '3') NOT NULL: Representamos a situação da conta (registrada, cancelada, paga) usando ENUM, sem permitir valores nulos.

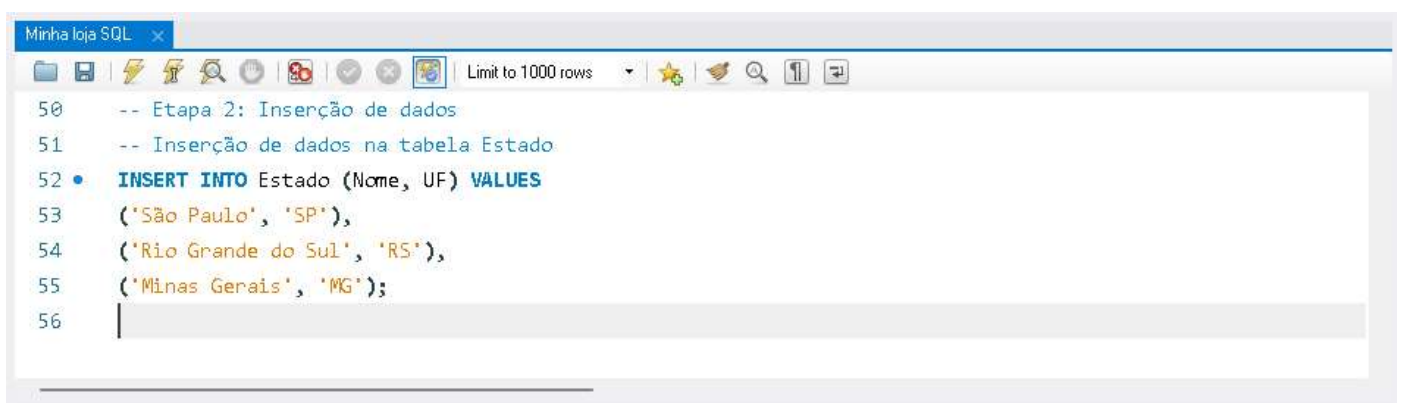
INDEX fk_ContaReceber_Cliente_idx (Cliente_ID): Criamos um índice na coluna "Cliente_ID" para otimização de consultas.

FOREIGN KEY (Cliente_ID) REFERENCES Cliente(ID): Estabelecemos uma chave estrangeira na coluna "Cliente_ID", garantindo integridade referencial com a tabela "Cliente".

2.3. – ETAPA 2: INSERÇÃO DE DADOS

Nesta etapa, dedicamos-nos à crucial tarefa de inserir dados nas tabelas do banco de dados. Essa etapa é fundamental, pois é aqui que os dados reais ganham vida dentro do sistema, representando entidades do mundo real conforme exigido, com um mínimo de três registros por tabela. O código fornecido consiste em uma série de instruções SQL de inserção para cada tabela, onde os valores são meticulosamente fornecidos para cada coluna especificada.

Na tabela "Estado", os registros dos estados brasileiros são inseridos utilizando o comando "INSERT INTO", atribuindo seus nomes e siglas (UF) como valores.



```
50  -- Etapa 2: Inserção de dados
51  -- Inserção de dados na tabela Estado
52  • INSERT INTO Estado (Nome, UF) VALUES
53    ('São Paulo', 'SP'),
54    ('Rio Grande do Sul', 'RS'),
55    ('Minas Gerais', 'MG');
56  |
```

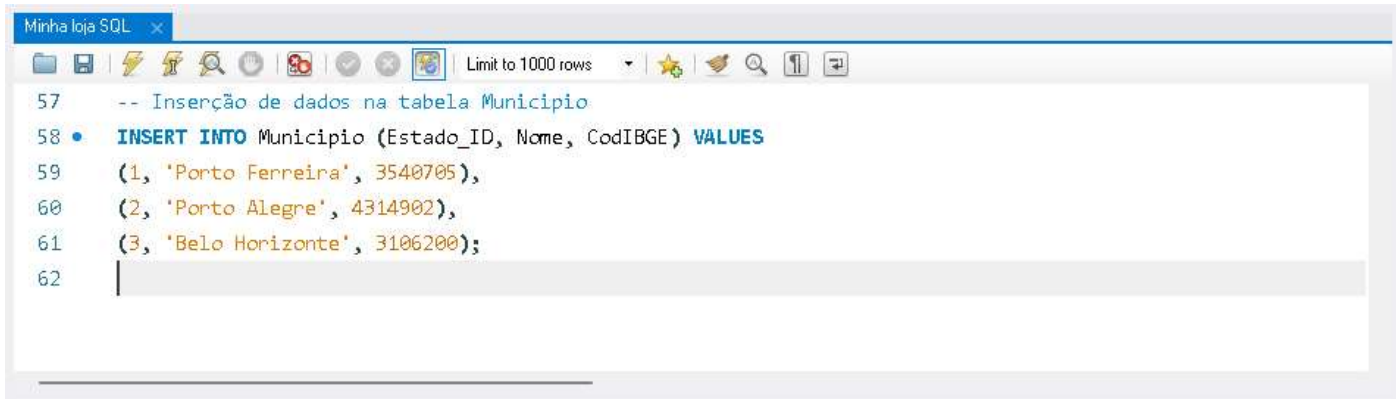
Figura 9. | Inserção de dados na tabela Estado

Esses registros são fundamentais para o funcionamento do sistema, pois permitem a identificação e a associação de outras entidades, como municípios e clientes, aos estados correspondentes. Além disso, a inserção desses dados é um passo importante para a validação e verificação da integridade do banco de dados, garantindo que as operações de consulta e manipulação futuras sejam precisas e confiáveis.

RELATÓRIO DE AULA PRÁTICA

Programação e Desenvolvimento de Banco de Dados

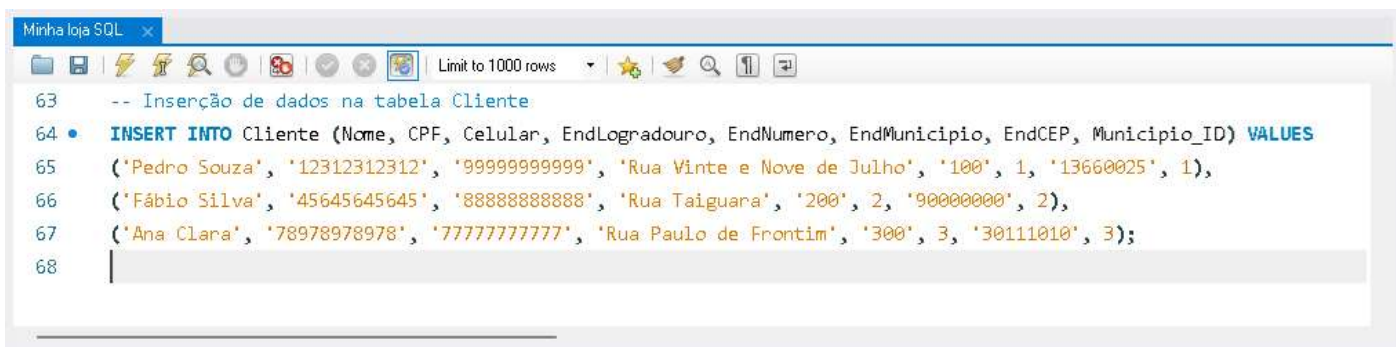
Em seguida, na tabela "Municipio", seguindo os princípios de inserção de dados da tabela "Estado", foram adicionados dados relacionados aos municípios. Isso incluiu o ID do estado correspondente, o nome do município e o código do IBGE.



```
57 -- Inserção de dados na tabela Municipio
58 • INSERT INTO Municipio (Estado_ID, Nome, CodIBGE) VALUES
59   (1, 'Porto Ferreira', 3540705),
60   (2, 'Porto Alegre', 4314902),
61   (3, 'Belo Horizonte', 3106200);
62 |
```

Figura 10. | Inserção de dados na tabela Municipio

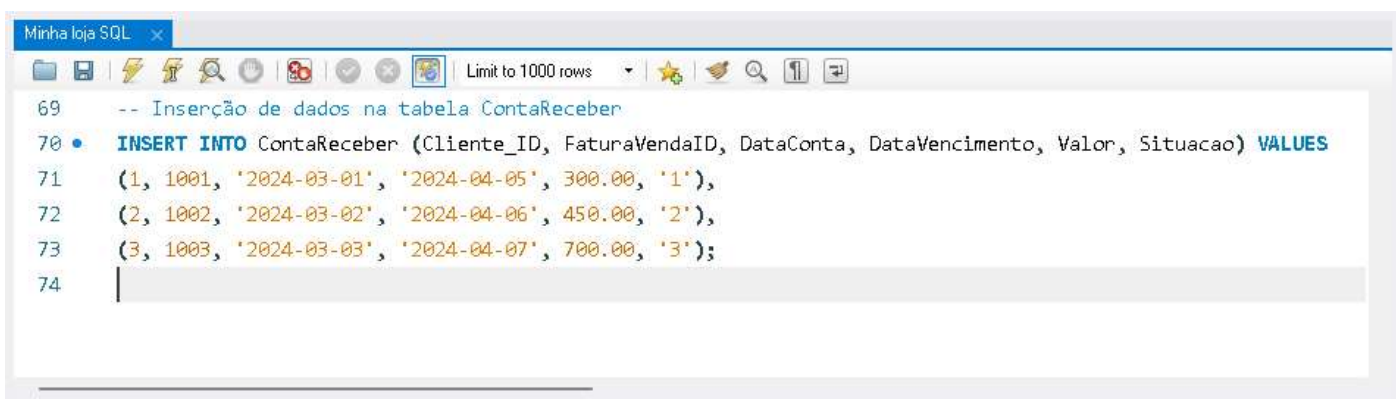
Continuando, na tabela "Cliente", mantendo a consistência com as práticas de inserção das tabelas anteriores, foram adicionadas informações detalhadas sobre os clientes, abrangendo nome, CPF, número de celular, endereço completo e a identificação do município de residência por meio do seu ID.



```
63 -- Inserção de dados na tabela Cliente
64 • INSERT INTO Cliente (Nome, CPF, Celular, EndLogradouro, EndNumero, EndMunicipio, EndCEP, Municipio_ID) VALUES
65   ('Pedro Souza', '12312312312', '99999999999', 'Rua Vinte e Nove de Julho', '100', 1, '13660025', 1),
66   ('Fábio Silva', '45645645645', '88888888888', 'Rua Taiguara', '200', 2, '90000000', 2),
67   ('Ana Clara', '78978978978', '77777777777', 'Rua Paulo de Frontim', '300', 3, '30111010', 3);
68 |
```

Figura 11. | Inserção de dados na tabela Cliente

Finalizando, na tabela "ContaReceber", registramos os dados referentes às contas a receber, incluindo o ID do cliente associado, ID da fatura de venda, datas de emissão e vencimento, valor da conta e sua situação atual.



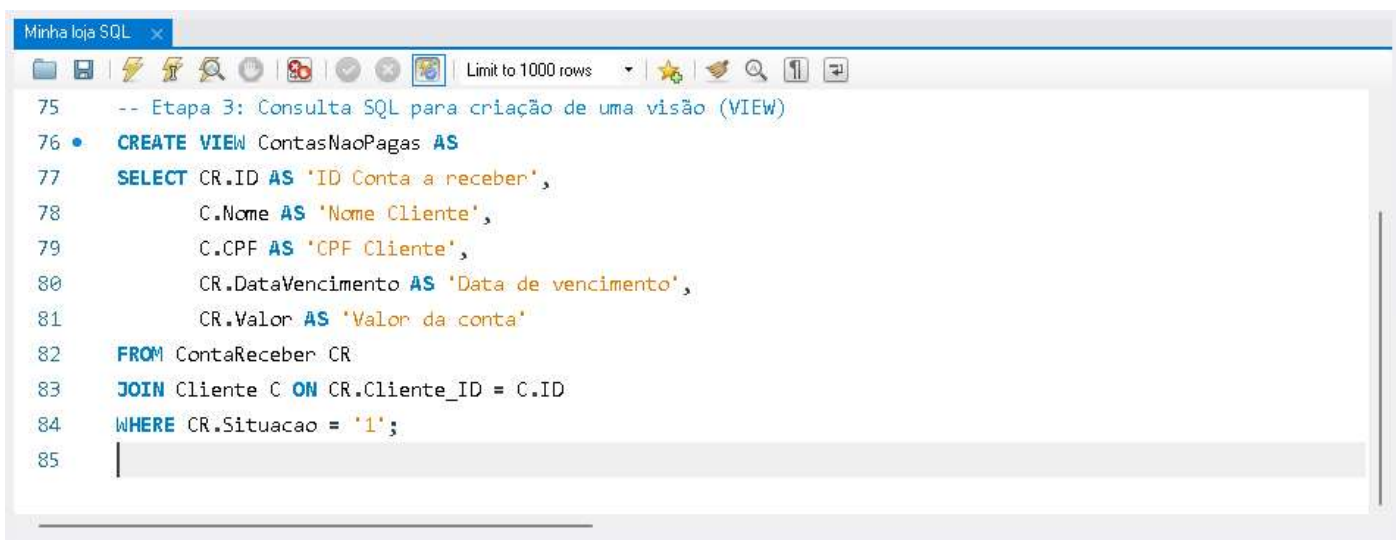
```
69 -- Inserção de dados na tabela ContaReceber
70 • INSERT INTO ContaReceber (Cliente_ID, FaturaVendaID, DataConta, DataVencimento, Valor, Situacao) VALUES
71   (1, 1001, '2024-03-01', '2024-04-05', 300.00, '1'),
72   (2, 1002, '2024-03-02', '2024-04-06', 450.00, '2'),
73   (3, 1003, '2024-03-03', '2024-04-07', 700.00, '3');
74 |
```

Figura 12. | Inserção de dados na tabela ContaReceber

Essas inserções de dados nas tabelas do nosso banco de dados são fundamentais para criar um ambiente simulado que reflita a realidade e seja funcional no banco de dados. Elas garantem que consultas e operações futuras possam ser conduzidas com informações relevantes e atualizadas.

2.4. – ETAPA 3: CRIAÇÃO DA VISÃO (VIEW)

Para concluir nosso banco de dados, nessa última etapa, utilizando o comando "CREATE" criamos uma visão (VIEW), que é uma consulta SQL armazenada que pode ser tratada como uma tabela virtual no banco de dados. A visão "ContasNaoPagas" foi criada para fornecer informações específicas sobre contas que ainda não foram pagas, ou a receber.



```
75  -- Etapa 3: Consulta SQL para criação de uma visão (VIEW)
76  • CREATE VIEW ContasNaoPagas AS
77  SELECT CR.ID AS 'ID Conta a receber',
78         C.Nome AS 'Nome Cliente',
79         C.CPF AS 'CPF Cliente',
80         CR.DataVencimento AS 'Data de vencimento',
81         CR.Valor AS 'Valor da conta'
82  FROM ContaReceber CR
83  JOIN Cliente C ON CR.Cliente_ID = C.ID
84  WHERE CR.Situacao = '1';
85  |
```

Figura 13. | Criação da VIEW

No código criado, a visão é construída a partir de uma consulta que seleciona campos relevantes das tabelas "ContaReceber" e "Cliente", filtrando os resultados com base na situação das contas a receber. Isso inclui o ID da conta a receber, o nome e o CPF do cliente, a data de vencimento da conta e o valor associado. Essa visão facilita a consulta e análise de dados específicos sobre contas a receber, proporcionando uma perspectiva clara e concisa dessas informações dentro do contexto do banco de dados.

Além disso, a consulta utiliza uma junção (JOIN) com a tabela "Cliente" para obter o nome e o CPF do cliente associado à conta. Essa junção é realizada com base na correspondência entre os IDs de cliente nas tabelas "ContaReceber" (CR.Cliente_ID) e "Cliente" (C.ID). A cláusula WHERE é utilizada para filtrar apenas as contas com situação igual a '1', que neste caso representa as contas não pagas.

RELATÓRIO DE AULA PRÁTICA

Programação e Desenvolvimento de Banco de Dados

3. RESULTADOS

Após a conclusão de todas as etapas, selecionamos o script completo do código e o executamos. Assim, nosso banco de dados foi adicionado à plataforma do sistema MySQL, ficando hospedado em "SCHEMAS", como ilustrado na figura abaixo.

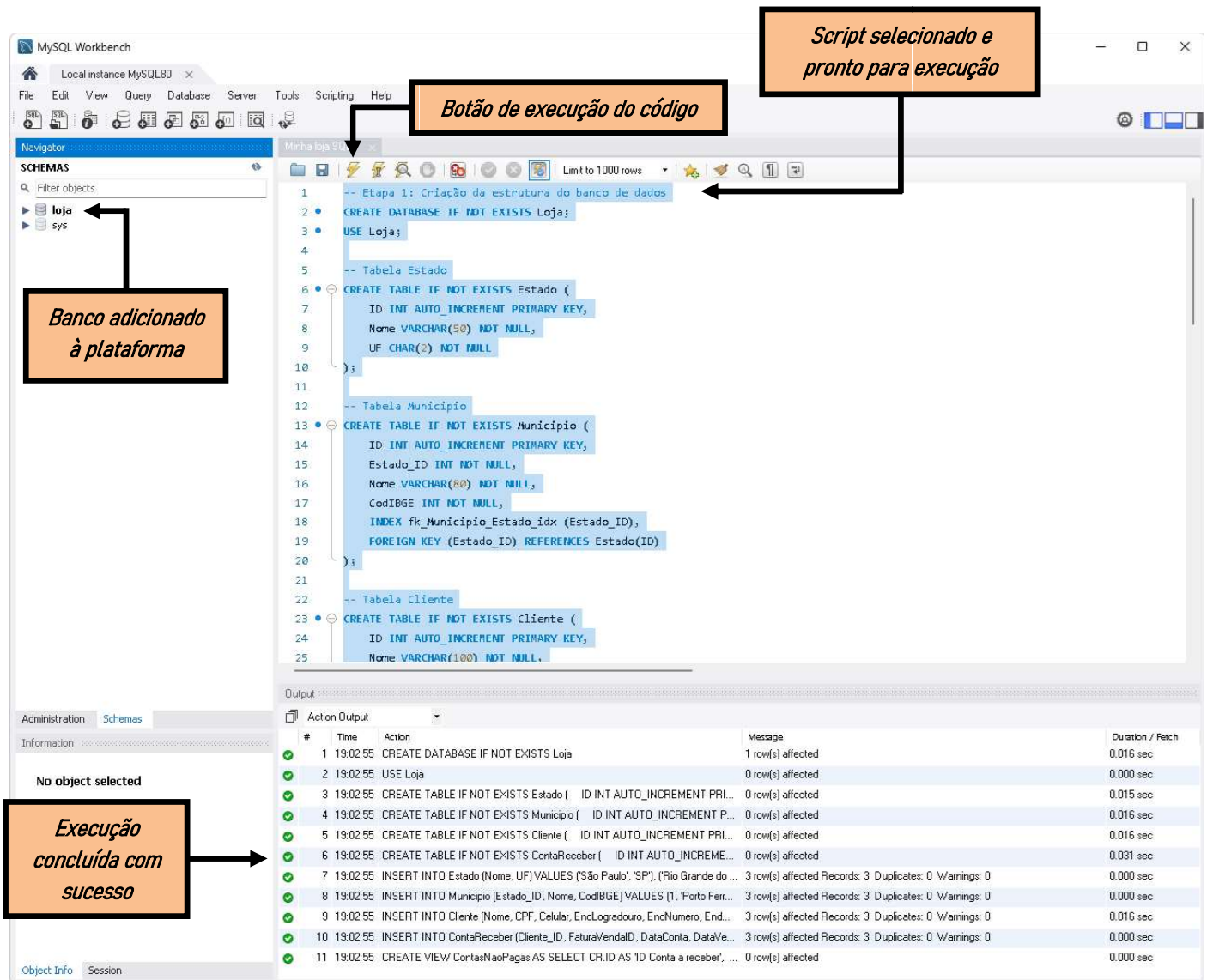


Figura 14. | Adicionando o Banco de Dados na Plataforma

É importante ressaltar que o processo de implementação foi realizado sem intercorrências, o que refletiu em uma execução eficiente e bem-sucedida do projeto. A estrutura do banco de dados demonstra-se totalmente organizada e funcional, com uma base sólida para realização de análises e operações, como veremos adiante.

RELATÓRIO DE AULA PRÁTICA

Programação e Desenvolvimento de Banco de Dados

Com o banco da loja ativado, nesta etapa, observaremos todos os testes feitos. Utilizando o comando **"SELECT * FROM ContaReceber;"**, podemos ver todas as contas a receber.

The screenshot shows the SQL Enterprise Manager interface. On the left, the 'Navigator' pane displays the database schema, including a table named 'ContaReceber'. The main window, titled 'Minha loja SQL', contains a query editor with the following SQL code:

```
83 JOIN Cliente C ON CR.Cliente_ID = C.ID
84 WHERE CR.Situacao = '1';
85
86 -- Exibir resultados da visão (VIEW)
87 SELECT * FROM ContaReceber;
```

Below the query editor, the 'Result Grid' displays the results of the query. The grid has columns: ID, Cliente_ID, FaturaVendaID, DataConta, DataVencimento, Valor, and Situacao. The results are as follows:

ID	Cliente_ID	FaturaVendaID	DataConta	DataVencimento	Valor	Situacao
1	1	1001	2024-03-01	2024-04-05	300.00	1
2	2	1002	2024-03-02	2024-04-06	450.00	2
3	3	1003	2024-03-03	2024-04-07	700.00	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Below the result grid, the 'Output' pane shows the execution details of the query, including the time taken and the number of rows returned.

Figura 15. | Consulta de contas a receber

A criação da visão "ContasNaoPagas" facilitou significativamente essa análise, e agora seguiremos para os demais testes. Para visualizar todos os clientes cadastrados no sistema, utilizamos o comando **"SELECT * FROM Cliente;"** e executamos a operação.

The screenshot shows the SQL Enterprise Manager interface. On the left, the 'Navigator' pane displays the database schema, including a table named 'Cliente'. The main window, titled 'Minha loja SQL', contains a query editor with the following SQL code:

```
85
86 -- Exibir resultados da visão (VIEW)
87 SELECT * FROM ContaReceber;
88 SELECT * FROM Cliente;
89
```

Below the query editor, the 'Result Grid' displays the results of the query. The grid has columns: ID, Nome, CPF, Celular, EndLogradouro, EndNumero, EndMunicipio, EndCEP, and Municipio_ID. The results are as follows:

ID	Nome	CPF	Celular	EndLogradouro	EndNumero	EndMunicipio	EndCEP	Municipio_ID
1	Pedro Souza	12312312312	99999999999	Rua Vinte e Nove de Julho	100	1	13660025	1
2	Fábio Silva	45645645645	88888888888	Rua Taiguara	200	2	90000000	2
3	Ana Clara	78978978978	77777777777	Rua Paulo de Frontin	300	3	30111010	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Below the result grid, the 'Output' pane shows the execution details of the query, including the time taken and the number of rows returned.

Figura 16. | Consulta de clientes cadastrados

Uma observação importante, vale lembrar que o uso de comandos corretos é fundamental para garantir o sucesso e a eficácia da consulta. Isso é essencial para compreender melhor o propósito da consulta e obter os resultados desejados.

RELATÓRIO DE AULA PRÁTICA

Programação e Desenvolvimento de Banco de Dados

Utilizando o comando **"SELECT * FROM Estado;"**, podemos verificar em qual estado cada cliente reside ou, dependendo do sistema de banco de dados de uma rede de lojas, em qual estado ele está cadastrado.

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'loja' schema with tables and views. The right pane shows a query window with the following SQL code:

```
85
86 -- Exibir resultados da visão (VIEW)
87 SELECT * FROM ContaReceber;
88 SELECT * FROM Cliente;
89 SELECT * FROM Estado;
```

The 'Result Grid' displays the following data:

ID	Nome	UF
1	São Paulo	SP
2	Rio Grande do Sul	RS
3	Minas Gerais	MG
NULL	NULL	NULL

The 'Output' pane shows the execution log for the query:

#	Time	Action	Message	Duration / Fetch
1	19:44:24	SELECT * FROM ContaReceber LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
2	19:48:26	SELECT * FROM Cliente LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
3	19:54:36	SELECT * FROM Estado LIMIT 0, 1000	3 row(s) returned	0.016 sec / 0.000 sec

Figura 17. | Consulta de Estado

Da mesma forma, utilizando o comando **"SELECT * FROM Municipio;"**, podemos verificar em qual município cada cliente reside ou, em qual, ele está cadastrado.

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'loja' schema with tables and views. The right pane shows a query window with the following SQL code:

```
83 JOIN Cliente C ON CR.Cliente_ID = C.ID
84 WHERE CR.Situacao = '1';
85
86 -- Exibir resultados da visão (VIEW)
87 SELECT * FROM ContaReceber;
88 SELECT * FROM Cliente;
89 SELECT * FROM Estado;
90 SELECT * FROM Municipio;
```

The 'Result Grid' displays the following data:

ID	Estado_ID	Nome	CodIBGE
1	1	Porto Ferreira	3540705
2	2	Porto Alegre	4314902
3	3	Belo Horizonte	3106200
NULL	NULL	NULL	NULL

The 'Output' pane shows the execution log for the query:

#	Time	Action	Message	Duration / Fetch
1	19:44:24	SELECT * FROM ContaReceber LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
2	19:48:26	SELECT * FROM Cliente LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
3	19:54:36	SELECT * FROM Estado LIMIT 0, 1000	3 row(s) returned	0.016 sec / 0.000 sec
4	20:01:38	SELECT * FROM Municipio LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

Figura 18. | Consulta de Município

RELATÓRIO DE AULA PRÁTICA

Programação e Desenvolvimento de Banco de Dados

Como vimos, a linguagem SQL oferece uma variedade de comandos para realizar diversas análises, operações e tipos de consultas. Por exemplo, se quisermos saber o nome e número de celular de um cliente, podemos digitar o seguinte comando: ***"SELECT Nome AS 'Nome do Cliente', Celular AS 'Telefone de Contato' FROM Cliente WHERE ID = 1;"***. Tudo depende das informações que estamos buscando. Para concluir, exportamos a imagem do diagrama do nosso banco de dados.

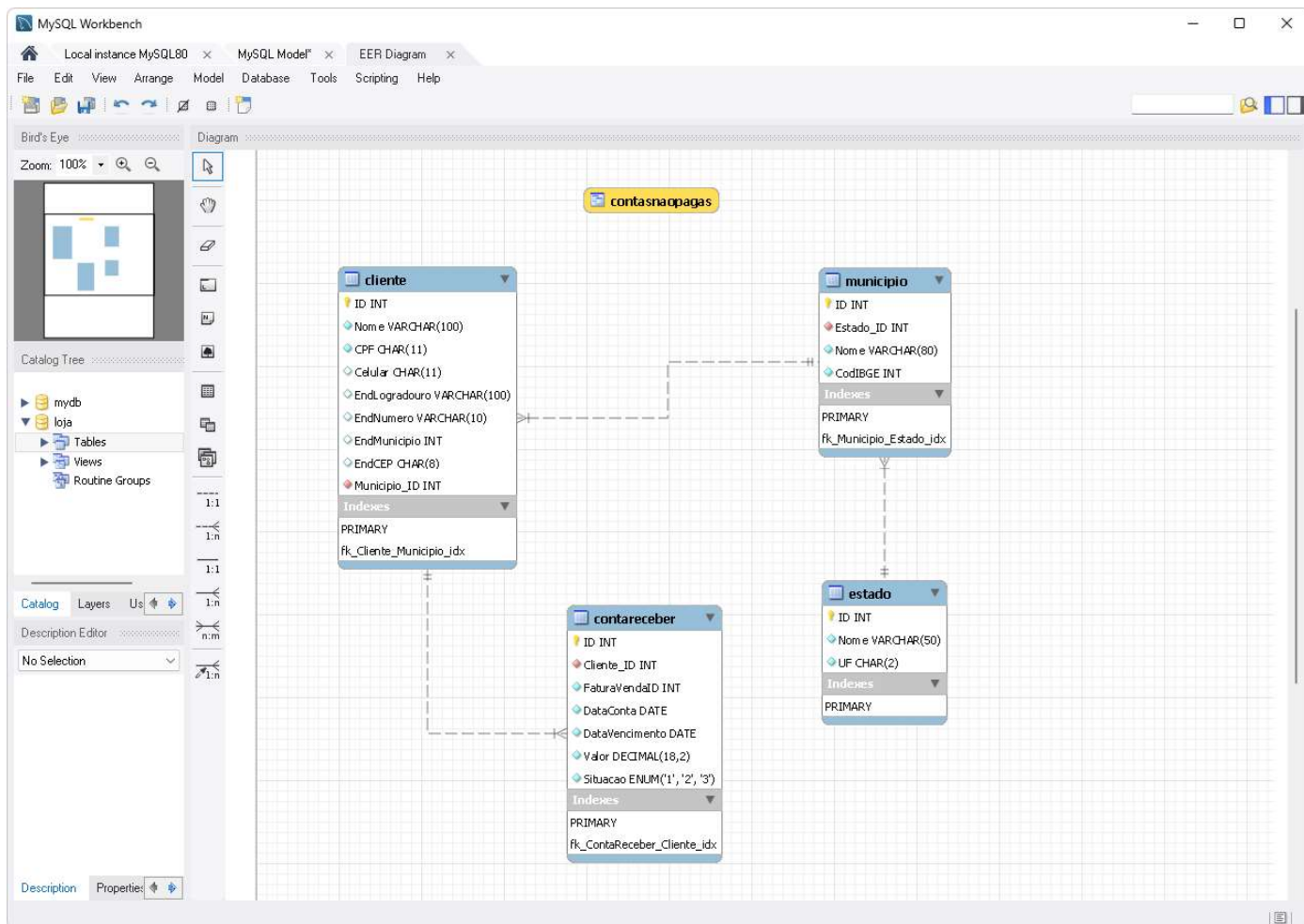


Figura 19. | Diagrama do Banco de Dados

4. CONCLUSÃO

Durante este projeto, alcançamos com sucesso a criação e execução do banco de dados, demonstrando uma implementação eficaz. Dividimos a construção do código em três etapas distintas: "Criação da estrutura do banco de dados", "Inserção de dados" e "Consulta SQL por meio de uma visão (VIEW)".

Destacamos a eficiência na estruturação do banco de dados, com tabelas bem definidas e relacionamentos apropriados entre elas. A inserção de dados seguiu as diretrizes estabelecidas, mantendo a integridade referencial entre as tabelas e fornecendo uma base sólida para análises e operações. Isso resultou em uma organização lógica dos dados, facilitando consultas e aprimorando os processos internos da loja.

A criação da visão "ContasNaoPagas" foi estratégica, confirmada pela exibição precisa dos resultados, garantindo a execução correta das consultas e a disponibilidade dos dados para acesso e análise.

Para garantir o sucesso da execução do nosso banco de dados, contamos com duas ferramentas essenciais: o MySQL Community Server e o MySQL Workbench. Enquanto o MySQL Community Server proporcionou uma base robusta para armazenar e gerenciar os dados, o MySQL Workbench simplificou o desenvolvimento, modelagem e administração do banco de dados por meio de uma interface intuitiva e recursos avançados. Juntos, esses recursos foram fundamentais para a execução eficaz do banco de dados, garantindo sua confiabilidade e desempenho.

Em resumo, o banco de dados recém-criado possibilitou uma gestão mais eficiente da nossa loja, permitindo uma análise mais detalhada dos dados, tomadas de decisão embasadas e uma melhor compreensão do nosso sistema. A cada etapa do processo de desenvolvimento, serviu-nos como fonte de aprendizado sobre os fundamentos de ferramentas de criação e modelagem de banco de dados, bem como sobre o uso eficiente da linguagem SQL.

5. REFERÊNCIAS

LIVRO DIDÁTICO, **PROGRAMAÇÃO EM BANCO DE DADOS**, © 2018 por Editora e Distribuidora Educacional S.A.
NUNES, Sergio Eduardo / Cód. da atividade: 3936324

JUNIOR, Gilberto Fernandes, **PROGRAMAÇÃO EM BANCO DE DADOS**, **Teleaulas**, Cód. Atividades: 3936316, 3936318, 3936320 e 3936322.

Programa **Workbench MySQL** - MySQL Community Server <<https://dev.mysql.com/downloads/workbench/>>

Plataforma **MIMO**: Curso SQL <<https://mimo.org/>>

Material Acessado em 2024.