Relatório de Implementação do Banco de Dados: Plataforma de Cursos Online

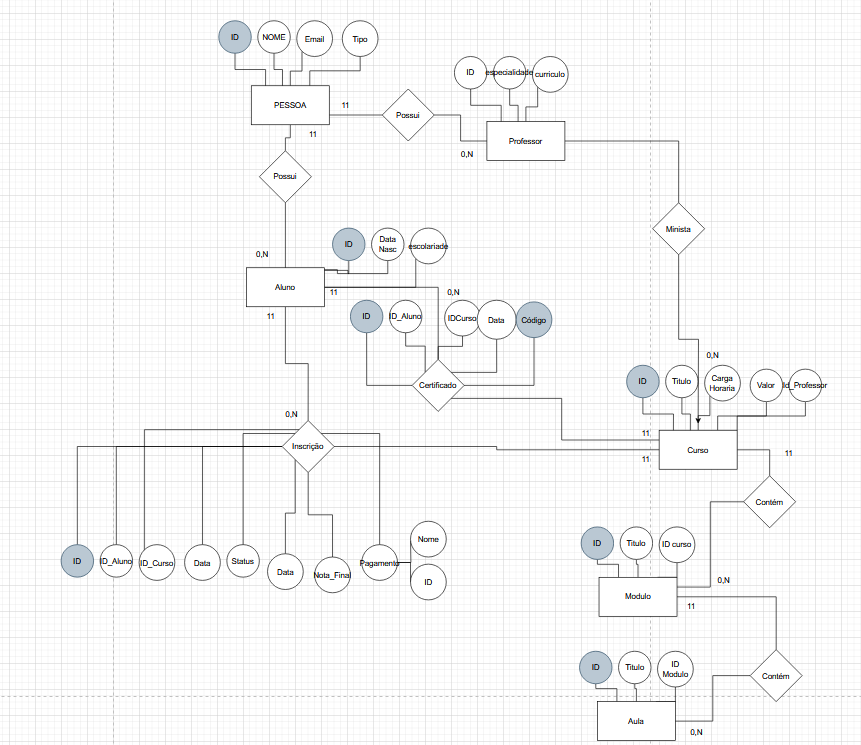
# 1. Contexto e Problema Definido:

O objetivo deste projeto é modelar e implementar um sistema de banco de dados para uma **plataforma de cursos online**. O sistema foi projetado para gerenciar informações essenciais relacionadas a:

* **Alunos:** Estudantes inscritos na plataforma.
* **Professores:** Instrutores que ministram os cursos.
* **Cursos:** Detalhes dos conteúdos oferecidos, organizados em módulos e aulas.
* **Inscrições:** Registros dos alunos nos cursos, incluindo informações sobre status e forma de pagamento.
* **Certificados:** Documentos emitidos após a conclusão bem-sucedida dos cursos.

O objetivo é simular o funcionamento de um sistema de ensino à distância, garantindo a organização e a integridade dos dados.

# 2. Modelo Conceitual



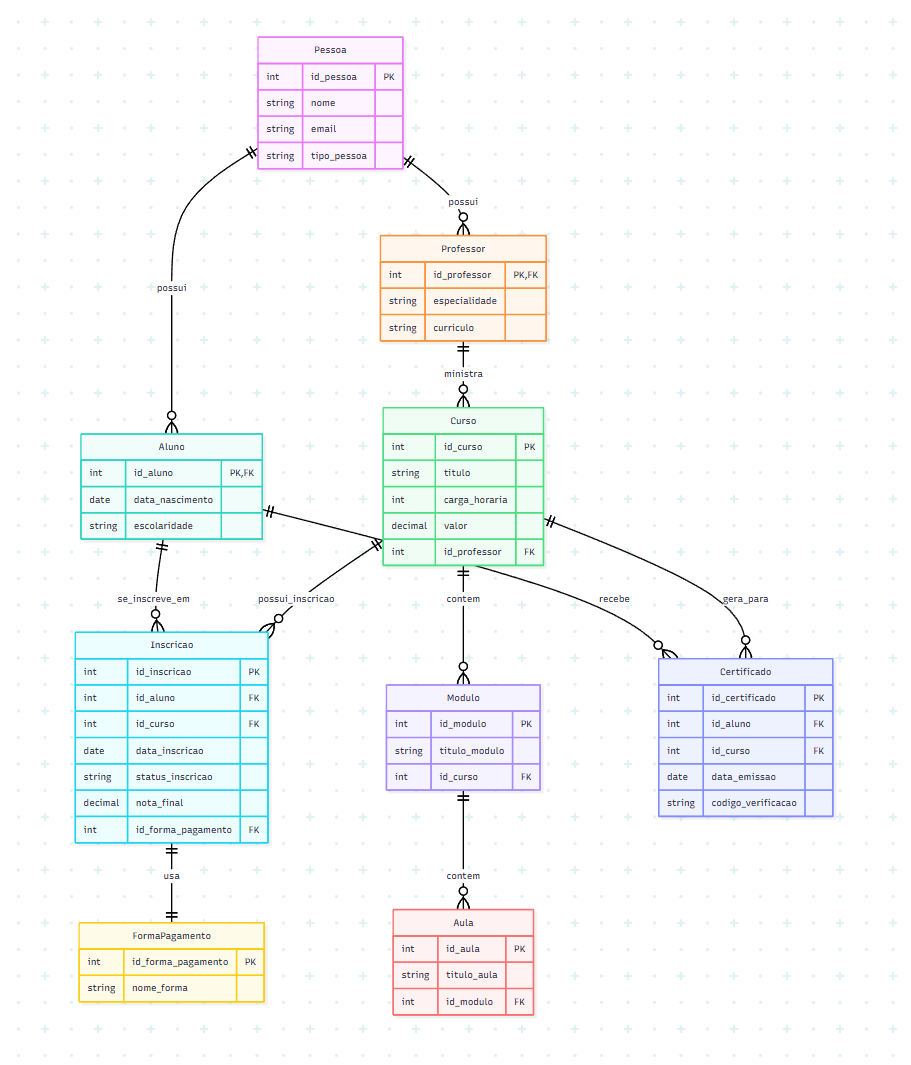
# 3. Modelo Lógico

O modelo conceitual oferece uma visão de alto nível das entidades do domínio da plataforma de cursos e como elas se relacionam, abstraindo detalhes de implementação. Ele foi visualizado utilizando um Diagrama de Entidade-Relacionamento (ERD) na sintaxe Mermaid.

|  |
| --- |
| Snippet de código  erDiagram  direction TB    %% Grupo 1: Pessoa, Aluno, Professor  Pessoa {  int id\_pessoa PK ""   string nome ""   string email ""   string tipo\_pessoa ""   }  Aluno {  int id\_aluno PK,FK ""   date data\_nascimento ""   string escolaridade ""   }  Professor {  int id\_professor PK,FK ""   string especialidade ""   string curriculo ""   }   %% Certificado ANTES DE CURSO  Certificado {  int id\_certificado PK ""   int id\_aluno FK ""   int id\_curso FK ""   date data\_emissao ""   string codigo\_verificacao ""   }   %% Curso e componentes  Curso {  int id\_curso PK ""   string titulo ""   int carga\_horaria ""   decimal valor ""   int id\_professor FK ""   }  Modulo {  int id\_modulo PK ""   string titulo\_modulo ""   int id\_curso FK ""   }  Aula {  int id\_aula PK ""   string titulo\_aula ""   int id\_modulo FK ""   }   %% Inscrição, FormaPagamento  Inscricao {  int id\_inscricao PK ""   int id\_aluno FK ""   int id\_curso FK ""   date data\_inscricao ""   string status\_inscricao ""   decimal nota\_final ""   int id\_forma\_pagamento FK ""   }  FormaPagamento {  int id\_forma\_pagamento PK ""   string nome\_forma ""   }   %% Relações  Pessoa ||--o{ Aluno : "possui"  Pessoa ||--o{ Professor : "possui"  Professor ||--o{ Curso : "ministra"   Aluno ||--o{ Certificado : "recebe"  Curso ||--o{ Certificado : "gera\_para"   Curso ||--o{ Modulo : "contem"  Modulo ||--o{ Aula : "contem"  Aluno ||--o{ Inscricao : "se\_inscreve\_em"  Curso ||--o{ Inscricao : "possui\_inscricao"  Inscricao ||--|| FormaPagamento : "usa" |

## Principais Elementos e Relacionamentos Identificados:

* **Entidades:** Pessoa, Aluno, Professor, Curso, Modulo, Aula, FormaPagamento, Inscricao, Certificado.
* **Especialização (Herança):** A entidade Pessoa atua como um super-tipo, generalizando características comuns para os subtipos Aluno e Professor. Isso permite que ambos compartilhem atributos como nome e email, enquanto possuem atributos específicos de suas funções.
* **Relacionamento 1:N (Um para Muitos):**
  + Um Professor pode ministrar muitos Cursos.
  + Um Curso contem muitos Modulos.
  + Um Modulo contem muitas Aulas.
  + Um Aluno pode receber muitos Certificados.
* **Relacionamento N:N (Muitos para Muitos):**
  + A relação entre Aluno e Curso é de muitos para muitos (um aluno pode se inscrever em vários cursos, e um curso pode ter vários alunos inscritos). Este relacionamento é resolvido através da entidade associativa Inscricao, que captura detalhes específicos da interação como data\_inscricao e status\_inscricao.
* **Relacionamento 1:1 (Um para Um):**
  + Uma Inscricao usa uma única FormaPagamento.



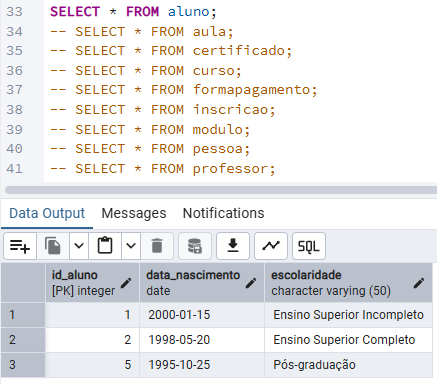
**Fonte:** Próprio Autor (2025

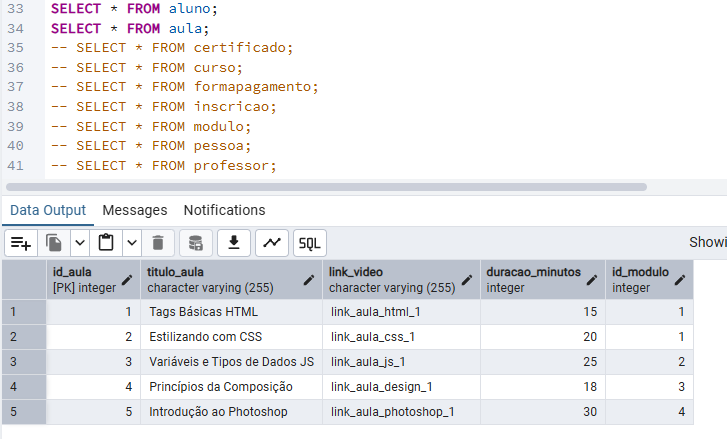
# 4. Comandos SQL de Implementação:

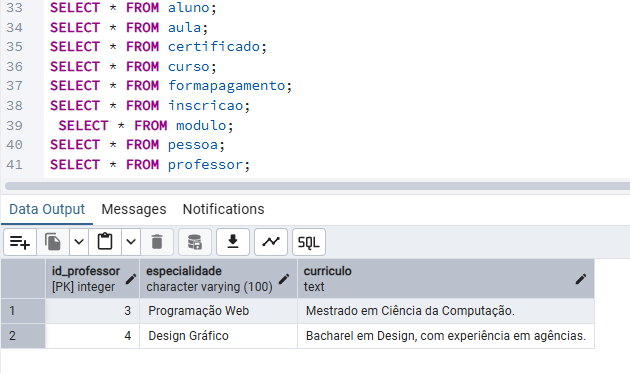
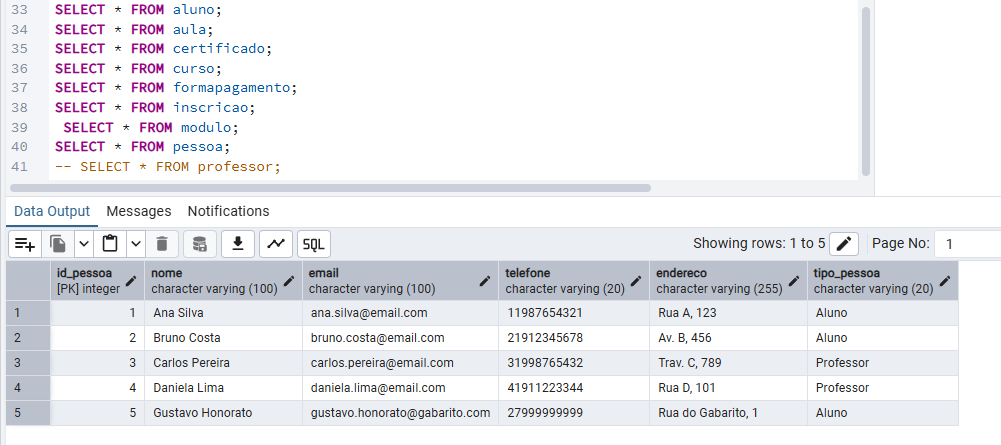
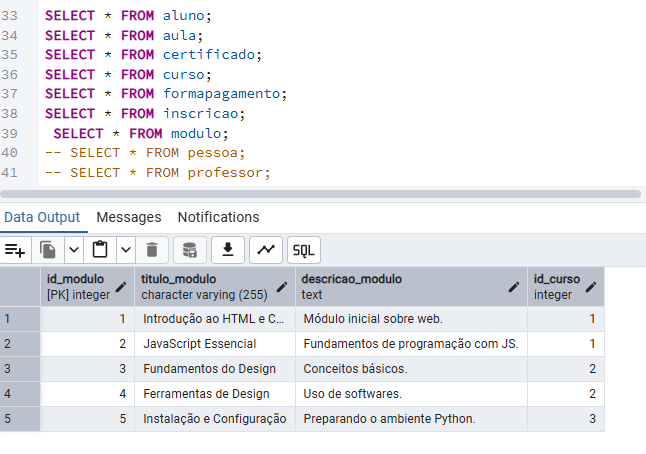
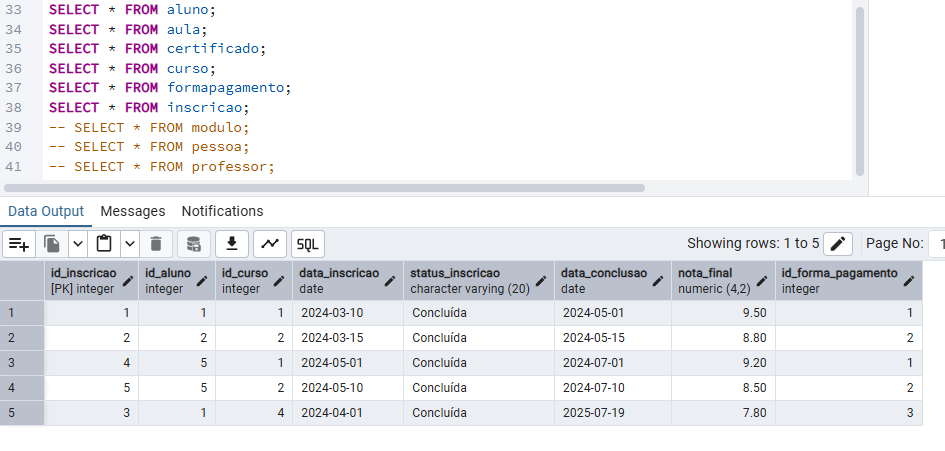
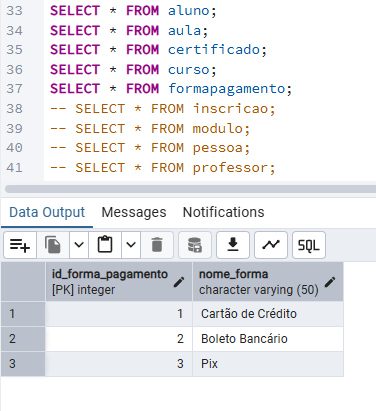
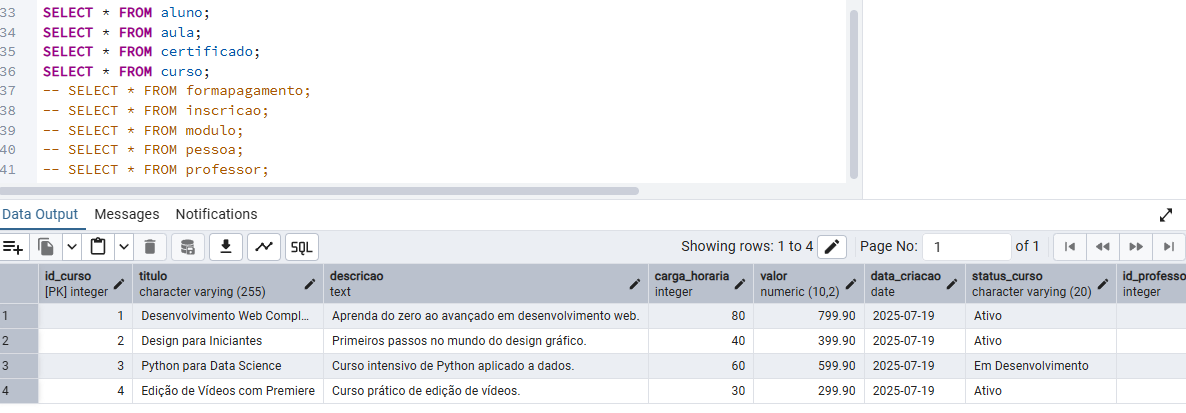
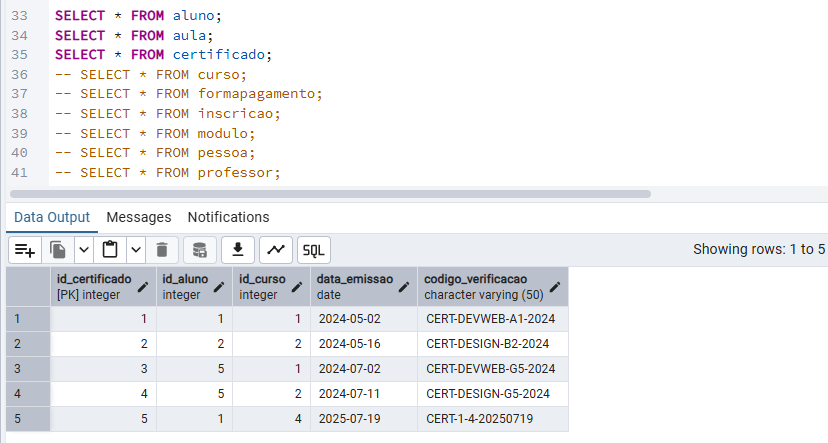
|  |
| --- |
| -- # ETAPA 1: CRIAÇÃO DAS TABELAS # -- # Descrição: Define a estrutura de cada tabela, suas colunas, # -- # tipos de dados (PostgreSQL), chaves primárias e # -- # estrangeiras, além de restrições de unicidade e # -- # validação de dados (CHECK constraints). #  -- Tabela Pessoa: Super-tipo para Aluno e Professor. Contém dados comuns. -- - id\_pessoa: Chave primária auto-incrementável (SERIAL). -- - email: Único para cada pessoa, garantindo identificação exclusiva. -- - tipo\_pessoa: Enumeração para garantir que seja 'Aluno' ou 'Professor'. CREATE TABLE Pessoa (  id\_pessoa SERIAL PRIMARY KEY,  nome VARCHAR(100) NOT NULL,  email VARCHAR(100) UNIQUE NOT NULL,  telefone VARCHAR(20),  endereco VARCHAR(255),  tipo\_pessoa VARCHAR(20) NOT NULL CHECK (tipo\_pessoa IN ('Aluno', 'Professor')) );  -- Tabela Aluno: Informações específicas do estudante. -- - id\_aluno: Chave primária que também é uma chave estrangeira, referenciando Pessoa.id\_pessoa. CREATE TABLE Aluno (  id\_aluno INT PRIMARY KEY REFERENCES Pessoa(id\_pessoa),  data\_nascimento DATE,  escolaridade VARCHAR(50) );  -- Tabela Professor: Informações específicas do instrutor. -- - id\_professor: Chave primária que também é uma chave estrangeira, referenciando Pessoa.id\_pessoa. CREATE TABLE Professor (  id\_professor INT PRIMARY KEY REFERENCES Pessoa(id\_pessoa),  especialidade VARCHAR(100),  curriculo TEXT );  -- Tabela Curso: Detalhes dos cursos oferecidos na plataforma. -- - id\_curso: Chave primária auto-incrementável. -- - carga\_horaria, valor: Obrigatórios para definir o curso. -- - data\_criacao: Preenchido automaticamente com a data atual (CURRENT\_DATE). -- - status\_curso: Enumeração para o estado do curso (Ativo, Inativo, Em Desenvolvimento). -- - id\_professor: Chave estrangeira para o professor que ministra o curso. CREATE TABLE Curso (  id\_curso SERIAL PRIMARY KEY,  titulo VARCHAR(255) NOT NULL,  descricao TEXT,  carga\_horaria INT NOT NULL,  valor NUMERIC(10, 2) NOT NULL,  data\_criacao DATE DEFAULT CURRENT\_DATE,  status\_curso VARCHAR(20) NOT NULL CHECK (status\_curso IN ('Ativo', 'Inativo', 'Em Desenvolvimento')),  id\_professor INT NOT NULL REFERENCES Professor(id\_professor) );  -- Tabela Modulo: Divisão de um curso em unidades temáticas. -- - id\_modulo: Chave primária auto-incrementável. -- - id\_curso: Chave estrangeira para o curso ao qual o módulo pertence. CREATE TABLE Modulo (  id\_modulo SERIAL PRIMARY KEY,  titulo\_modulo VARCHAR(255) NOT NULL,  descricao\_modulo TEXT,  id\_curso INT NOT NULL REFERENCES Curso(id\_curso) );  -- Tabela Aula: Unidade de conteúdo dentro de um módulo. -- - id\_aula: Chave primária auto-incrementável. -- - id\_modulo: Chave estrangeira para o módulo ao qual a aula pertence. CREATE TABLE Aula (  id\_aula SERIAL PRIMARY KEY,  titulo\_aula VARCHAR(255) NOT NULL,  link\_video VARCHAR(255),  duracao\_minutos INT,  id\_modulo INT NOT NULL REFERENCES Modulo(id\_modulo) );  -- Tabela FormaPagamento: Opções de pagamento disponíveis. -- - id\_forma\_pagamento: Chave primária auto-incrementável. -- - nome\_forma: Único para cada forma de pagamento (ex: 'Cartão de Crédito'). CREATE TABLE FormaPagamento (  id\_forma\_pagamento SERIAL PRIMARY KEY,  nome\_forma VARCHAR(50) UNIQUE NOT NULL );  -- Tabela Inscricao: Registro da matrícula de um aluno em um curso. -- Representa o relacionamento N:N entre Aluno e Curso. -- - id\_aluno, id\_curso: Chaves estrangeiras combinadas que formam a chave candidata para unicidade da inscrição. -- - data\_inscricao: Preenchida automaticamente. -- - status\_inscricao: Enumeração para o estado da inscrição (Pendente, Concluída, Cancelada). -- - data\_conclusao, nota\_final: Podem ser nulos até a conclusão do curso. -- - id\_forma\_pagamento: Chave estrangeira para a forma de pagamento utilizada. CREATE TABLE Inscricao (  id\_inscricao SERIAL PRIMARY KEY,  id\_aluno INT NOT NULL REFERENCES Aluno(id\_aluno),  id\_curso INT NOT NULL REFERENCES Curso(id\_curso),  data\_inscricao DATE DEFAULT CURRENT\_DATE,  status\_inscricao VARCHAR(20) NOT NULL CHECK (status\_inscricao IN ('Pendente', 'Concluída', 'Cancelada')),  data\_conclusao DATE,  nota\_final NUMERIC(4, 2),  id\_forma\_pagamento INT NOT NULL REFERENCES FormaPagamento(id\_forma\_pagamento) );  -- Tabela Certificado: Registro dos certificados emitidos. -- - id\_certificado: Chave primária auto-incrementável. -- - id\_aluno, id\_curso: Chaves estrangeiras para o aluno e o curso do certificado. -- - data\_emissao: Preenchida automaticamente. -- - codigo\_verificacao: Único e serve para a validação do certificado. CREATE TABLE Certificado (  id\_certificado SERIAL PRIMARY KEY,  id\_aluno INT NOT NULL REFERENCES Aluno(id\_aluno),  id\_curso INT NOT NULL REFERENCES Curso(id\_curso),  data\_emissao DATE DEFAULT CURRENT\_DATE,  codigo\_verificacao VARCHAR(50) UNIQUE NOT NULL ); |



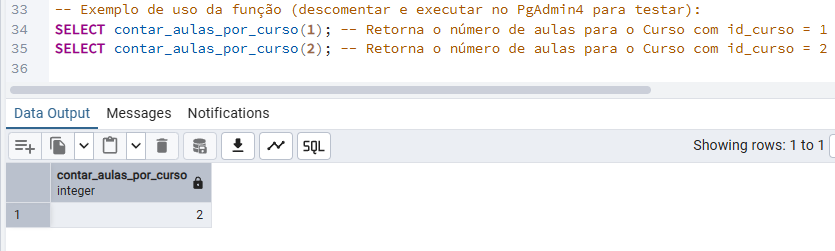
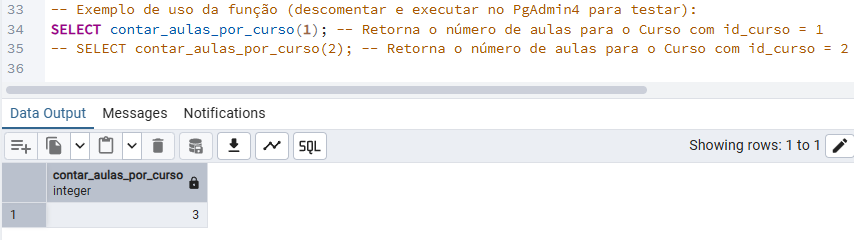
|  |
| --- |
| -- # ETAPA 2: INSERÇÃO DE DADOS DE EXEMPLO # -- # Descrição: Popula as tabelas recém-criadas com dados de exemplo # -- # para facilitar testes, demonstrações e validações. #   -- Inserção de dados na tabela Pessoa INSERT INTO Pessoa (nome, email, telefone, endereco, tipo\_pessoa) VALUES ('Ana Silva', 'ana.silva@email.com', '11987654321', 'Rua A, 123', 'Aluno'), ('Bruno Costa', 'bruno.costa@email.com', '21912345678', 'Av. B, 456', 'Aluno'), ('Carlos Pereira', 'carlos.pereira@email.com', '31998765432', 'Trav. C, 789', 'Professor'), ('Daniela Lima', 'daniela.lima@email.com', '41911223344', 'Rua D, 101', 'Professor'), ('Gustavo Honorato', 'gustavo.honorato@gabarito.com', '27999999999', 'Rua do Gabarito, 1', 'Aluno');  -- Inserção de dados na tabela Aluno (referenciando IDs de Pessoa) INSERT INTO Aluno (id\_aluno, data\_nascimento, escolaridade) VALUES (1, '2000-01-15', 'Ensino Superior Incompleto'), (2, '1998-05-20', 'Ensino Superior Completo'), (5, '1995-10-25', 'Pós-graduação');  -- Inserção de dados na tabela Professor (referenciando IDs de Pessoa) INSERT INTO Professor (id\_professor, especialidade, curriculo) VALUES (3, 'Programação Web', 'Mestrado em Ciência da Computação.'), (4, 'Design Gráfico', 'Bacharel em Design, com experiência em agências.');  -- Inserção de dados na tabela FormaPagamento INSERT INTO FormaPagamento (nome\_forma) VALUES ('Cartão de Crédito'), ('Boleto Bancário'), ('Pix');  -- Inserção de dados na tabela Curso INSERT INTO Curso (titulo, descricao, carga\_horaria, valor, id\_professor, status\_curso) VALUES ('Desenvolvimento Web Completo', 'Aprenda do zero ao avançado em desenvolvimento web.', 80, 799.90, 3, 'Ativo'), ('Design para Iniciantes', 'Primeiros passos no mundo do design gráfico.', 40, 399.90, 4, 'Ativo'), ('Python para Data Science', 'Curso intensivo de Python aplicado a dados.', 60, 599.90, 3, 'Em Desenvolvimento'), ('Edição de Vídeos com Premiere', 'Curso prático de edição de vídeos.', 30, 299.90, 4, 'Ativo');  -- Inserção de dados na tabela Modulo INSERT INTO Modulo (titulo\_modulo, descricao\_modulo, id\_curso) VALUES ('Introdução ao HTML e CSS', 'Módulo inicial sobre web.', 1), ('JavaScript Essencial', 'Fundamentos de programação com JS.', 1), ('Fundamentos do Design', 'Conceitos básicos.', 2), ('Ferramentas de Design', 'Uso de softwares.', 2), ('Instalação e Configuração', 'Preparando o ambiente Python.', 3);  -- Inserção de dados na tabela Aula INSERT INTO Aula (titulo\_aula, link\_video, duracao\_minutos, id\_modulo) VALUES ('Tags Básicas HTML', 'link\_aula\_html\_1', 15, 1), ('Estilizando com CSS', 'link\_aula\_css\_1', 20, 1), ('Variáveis e Tipos de Dados JS', 'link\_aula\_js\_1', 25, 2), ('Princípios da Composição', 'link\_aula\_design\_1', 18, 3), ('Introdução ao Photoshop', 'link\_aula\_photoshop\_1', 30, 4);  -- Inserção de dados na tabela Inscricao (Algumas já concluídas para testes, uma pendente para teste da trigger) INSERT INTO Inscricao (id\_aluno, id\_curso, data\_inscricao, status\_inscricao, id\_forma\_pagamento) VALUES (1, 1, '2024-03-10', 'Concluída', 1), (2, 2, '2024-03-15', 'Concluída', 2), (1, 4, '2024-04-01', 'Pendente', 3), -- Inscrição pendente que será usada para testar a trigger (5, 1, '2024-05-01', 'Concluída', 1), (5, 2, '2024-05-10', 'Concluída', 2);  -- Atualização de notas e datas de conclusão para inscrições já concluídas no momento da inserção inicial UPDATE Inscricao SET nota\_final = 9.5, data\_conclusao = '2024-05-01' WHERE id\_aluno = 1 AND id\_curso = 1; UPDATE Inscricao SET nota\_final = 8.8, data\_conclusao = '2024-05-15' WHERE id\_aluno = 2 AND id\_curso = 2; UPDATE Inscricao SET nota\_final = 9.2, data\_conclusao = '2024-07-01' WHERE id\_aluno = 5 AND id\_curso = 1; UPDATE Inscricao SET nota\_final = 8.5, data\_concluso = '2024-07-10' WHERE id\_aluno = 5 AND id\_curso = 2;  -- Inserção de dados na tabela Certificado (para as inscrições que já estavam concluídas no início) INSERT INTO Certificado (id\_aluno, id\_curso, data\_emissao, codigo\_verificacao) VALUES (1, 1, '2024-05-02', 'CERT-DEVWEB-A1-2024'), (2, 2, '2024-05-16', 'CERT-DESIGN-B2-2024'), (5, 1, '2024-07-02', 'CERT-DEVWEB-G5-2024'), (5, 2, '2024-07-11', 'CERT-DESIGN-G5-2024'); |



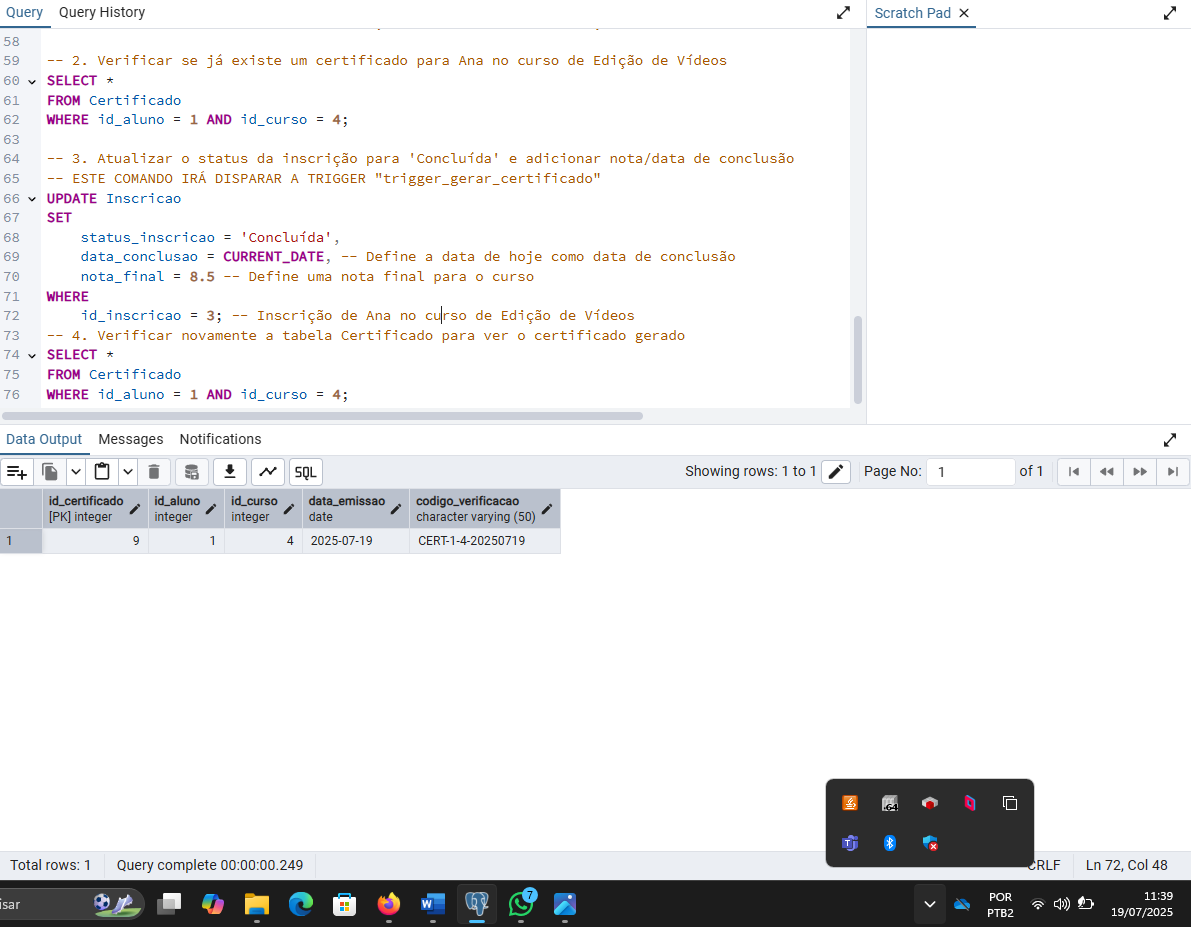


  
Fonte: Próprio Autor (2025)

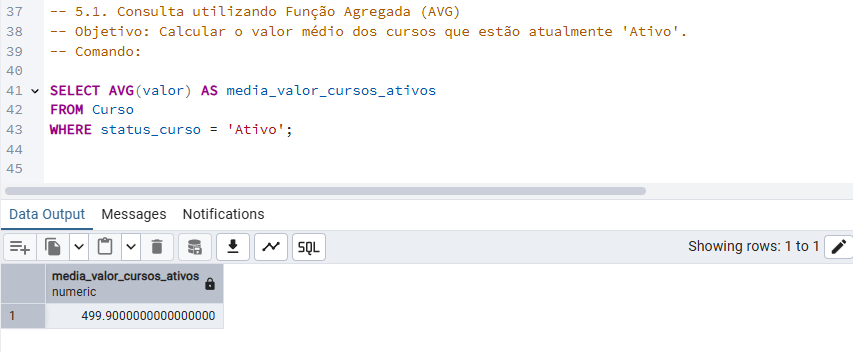
|  |
| --- |
| -- # ETAPA 3: FUNÇÕES # -- # Descrição: Cria blocos de código reutilizáveis (Stored Functions)# -- # para encapsular lógicas específicas no banco de dados.#   -- Função: contar\_aulas\_por\_curso -- Objetivo: Calcular e retornar o número total de aulas em um curso específico. -- - Parâmetro: `curso\_id` (inteiro), que é o identificador do curso. CREATE OR REPLACE FUNCTION contar\_aulas\_por\_curso(curso\_id INT) RETURNS INT AS $$ DECLARE  total\_aulas INT; BEGIN  SELECT COUNT(A.id\_aula) INTO total\_aulas  FROM Aula A  JOIN Modulo M ON A.id\_modulo = M.id\_modulo  WHERE M.id\_curso = curso\_id;  RETURN total\_aulas; END; $$ LANGUAGE plpgsql; -- Define a linguagem procedural usada (PL/pgSQL)  -- Exemplo de uso da função (descomentar e executar no PgAdmin4 para testar): -- SELECT contar\_aulas\_por\_curso(1); -- Retorna o número de aulas para o Curso com id\_curso = 1 -- SELECT contar\_aulas\_por\_curso(2); -- Retorna o número de aulas para o Curso com id\_curso = 2 |

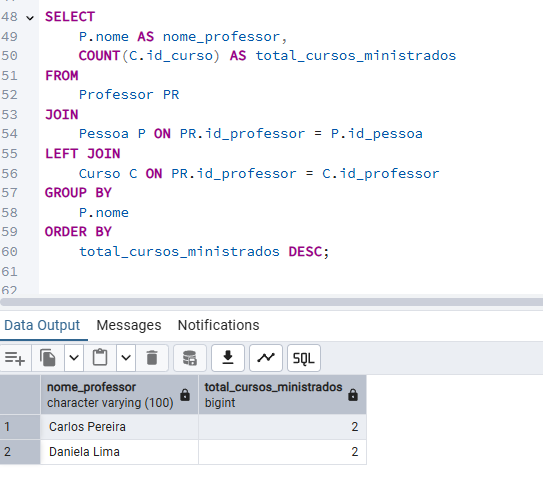
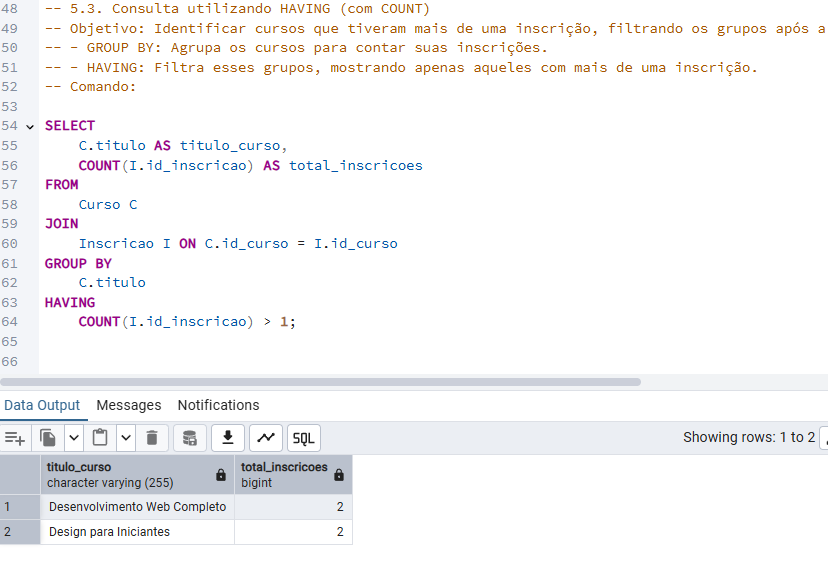
  
  
Fonte: Próprio Autor (2025)

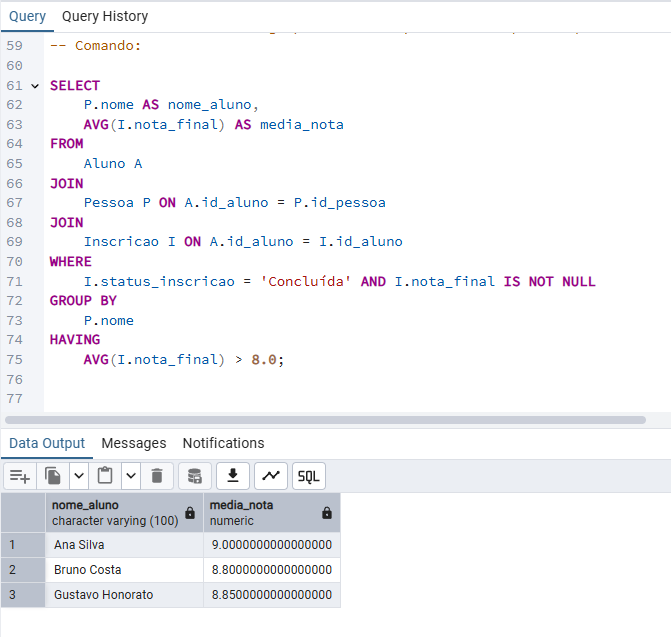
|  |
| --- |
| -- # ETAPA 4: TRIGGERS # -- # Descrição: Automatiza ações no banco de dados em resposta a # -- # eventos definidos (INSERT, UPDATE, DELETE). #   -- 4.1. Definição da Função da Trigger (`gerar\_certificado\_inscricao`) -- Objetivo: Esta função será executada pela trigger. Ela verifica se uma inscrição -- foi atualizada para o status 'Concluída' e, se sim, insere um novo -- registro na tabela `Certificado`, evitando duplicidades. CREATE OR REPLACE FUNCTION gerar\_certificado\_inscricao() RETURNS TRIGGER AS $$ BEGIN  -- Condição para disparar a geração do certificado:  -- NEW.status\_inscricao: Refere-se ao novo valor do status\_inscricao após o UPDATE.  -- NEW.data\_conclusao IS NOT NULL AND NEW.nota\_final IS NOT NULL: Garante que os dados essenciais para o certificado estão presentes.  IF NEW.status\_inscricao = 'Concluída' AND NEW.data\_conclusao IS NOT NULL AND NEW.nota\_final IS NOT NULL THEN  -- Verifica se um certificado para esta combinação ALUNO-CURSO JÁ EXISTE:  -- Isso impede a criação de certificados duplicados se a mesma inscrição for atualizada para 'Concluída' mais de uma vez.  IF NOT EXISTS (SELECT 1 FROM Certificado WHERE id\_aluno = NEW.id\_aluno AND id\_curso = NEW.id\_curso) THEN  -- Se não existir, insere um novo registro na tabela Certificado.  INSERT INTO Certificado (id\_aluno, id\_curso, data\_emissao, codigo\_verificacao)  VALUES (NEW.id\_aluno, NEW.id\_curso, NEW.data\_conclusao,  -- Geração de um código de verificação único e legível.  'CERT-' || NEW.id\_aluno || '-' || NEW.id\_curso || '-' || TO\_CHAR(NEW.data\_conclusao, 'YYYYMMDD'));  END IF;  END IF;  RETURN NEW; -- Retorna a nova linha para que a operação de UPDATE na tabela Inscricao prossiga normalmente. END; $$ LANGUAGE plpgsql;  -- 4.2. Criação da Trigger (`trigger\_gerar\_certificado`) -- Objetivo: Ligar a função `gerar\_certificado\_inscricao` à tabela `Inscricao` para que ela seja executada automaticamente. -- - `AFTER UPDATE ON Inscricao`: A trigger será ativada \*\*DEPOIS\*\* de qualquer operação de \*\*UPDATE\*\* na tabela `Inscricao`. -- - `FOR EACH ROW`: A função da trigger será executada para \*\*CADA LINHA\*\* que for modificada pelo comando UPDATE. -- - `EXECUTE FUNCTION gerar\_certificado\_inscricao()`: Especifica qual função PL/pgSQL deve ser executada. CREATE TRIGGER trigger\_gerar\_certificado AFTER UPDATE ON Inscricao FOR EACH ROW EXECUTE FUNCTION gerar\_certificado\_inscricao();  -- 4.3. Teste da Trigger -- Objetivo: Demonstrar o funcionamento automatizado da trigger ao mudar o status de uma inscrição para 'Concluída'.  -- 1. Consultar o estado inicial da inscrição com ID 3 (Ana no curso 4): -- Ela deve estar como 'Pendente'. -- SELECT id\_inscricao, id\_aluno, id\_curso, status\_inscricao, data\_conclusao, nota\_final FROM Inscricao WHERE id\_inscricao = 3;  -- 2. Verificar se já existe um certificado para Ana no curso 4 antes da atualização (NÃO DEVE HAVER): -- SELECT \* FROM Certificado WHERE id\_aluno = 1 AND id\_curso = 4;  -- 3. Executar o comando UPDATE que irá disparar a trigger: -- O status será mudado para 'Concluída', e a trigger agirá. -- UPDATE Inscricao -- SET status\_inscricao = 'Concluída', data\_conclusao = CURRENT\_DATE, nota\_final = 7.8 -- WHERE id\_inscricao = 3;  -- 4. Consultar novamente a tabela Certificado para verificar a criação automática (AGORA DEVE APARECER O REGISTRO): -- SELECT \* FROM Certificado WHERE id\_aluno = 1 AND id\_curso = 4; |

  
Fonte: Próprio Autor (2025)

|  |
| --- |
| -- # ETAPA 5: EXEMPLOS DE CONSULTAS SQL # -- # Descrição: Demonstrações de como consultar os dados utilizando # -- # funções agregadas e a cláusula HAVING. #  -- 5.1. Consulta utilizando Função Agregada (AVG) -- Objetivo: Calcular o valor médio dos cursos que estão atualmente 'Ativo'. -- Comando: /\* SELECT AVG(valor) AS media\_valor\_cursos\_ativos FROM Curso WHERE status\_curso = 'Ativo'; \*/  -- 5.2. Consulta utilizando Função Agregada (COUNT) -- Objetivo: Contar quantos cursos cada professor ministra. -- - Utiliza JOIN para combinar dados de Pessoa, Professor e Curso. -- - GROUP BY: Agrupa os resultados por nome de professor para contar os cursos de cada um. -- - LEFT JOIN: Garante que professores sem cursos também sejam listados (com contagem 0). -- Comando: /\* SELECT  P.nome AS nome\_professor,  COUNT(C.id\_curso) AS total\_cursos\_ministrados FROM  Professor PR JOIN  Pessoa P ON PR.id\_professor = P.id\_pessoa LEFT JOIN  Curso C ON PR.id\_professor = C.id\_professor GROUP BY  P.nome ORDER BY  total\_cursos\_ministrados DESC; \*/  -- 5.3. Consulta utilizando HAVING (com COUNT) -- Objetivo: Identificar cursos que tiveram mais de uma inscrição, filtrando os grupos após a contagem. -- - GROUP BY: Agrupa os cursos para contar suas inscrições. -- - HAVING: Filtra esses grupos, mostrando apenas aqueles com mais de uma inscrição. -- Comando: /\* SELECT  C.titulo AS titulo\_curso,  COUNT(I.id\_inscricao) AS total\_inscricoes FROM  Curso C JOIN  Inscricao I ON C.id\_curso = I.id\_curso GROUP BY  C.titulo HAVING  COUNT(I.id\_inscricao) > 1; \*/  -- 5.4. Consulta utilizando HAVING (com AVG) -- Objetivo: Encontrar alunos cuja nota média em cursos concluídos foi superior a 8.0. -- - WHERE: Filtra as inscrições para apenas as 'Concluídas' e com nota. -- - GROUP BY: Agrupa os alunos para calcular a média de suas notas. -- - HAVING: Filtra esses grupos de alunos, mostrando apenas aqueles com média acima de 8.0. -- Comando: /\* SELECT  P.nome AS nome\_aluno,  AVG(I.nota\_final) AS media\_nota FROM  Aluno A JOIN  Pessoa P ON A.id\_aluno = P.id\_pessoa JOIN  Inscricao I ON A.id\_aluno = I.id\_aluno WHERE  I.status\_inscricao = 'Concluída' AND I.nota\_final IS NOT NULL GROUP BY  P.nome HAVING  AVG(I.nota\_final) > 8.0; \*/ |







Fonte: Próprio Autor (2025)