Trabalho G1 - INF1383 - 3WA

Daniel Santana Souza - 2310995

---

1 - Texto-Enunciado

O Laboratório de Caracterização de Águas (LabÁguas), vinculado ao Departamento de Química da PUC-Rio, realiza análises laboratoriais de qualidade de águas para clientes diversos, sejam eles pessoas físicas ou jurídicas. O laboratório deseja implementar um sistema de gerenciamento de dados que permita um controle completo e rastreável do ciclo de vida das amostras analisadas, substituindo planilhas e processos manuais atualmente utilizados.

No sistema proposto, cada cliente é identificado exclusivamente pelo seu endereço de e-mail. Além do e-mail, o laboratório precisa armazenar informações adicionais sobre cada cliente, incluindo seu nome completo, o nome da pessoa responsável pelas amostras enviadas, endereço detalhado e telefone ou fax para contato. Cada cliente pode enviar uma ou mais amostras para análise ao laboratório e para cada envio é registrada a data da efetuação.

Cada amostra recebida possui um identificador único gerado automaticamente pelo sistema (**IDAmostra**) e também um código interno também único atribuído pelo laboratório (**Código PUC**), e está associada a exatamente um cliente. Para cada amostra, o sistema precisa registrar o ponto exato de coleta, a data da coleta, o nome do coletor, o tipo de frasco utilizado, a matriz da água (por exemplo, água mineral, água bruta etc.), o tipo específico da amostra e a forma pela qual a amostra foi entregue ao laboratório (presencialmente, Correios ou outro método). O recebimento de cada amostra é feito por exatamente um funcionário do laboratório, que fica responsável pelo cadastro inicial da amostra no sistema. O sistema deve registrar também a data e hora exata desse recebimento. Sobre cada funcionário, o sistema precisa armazenar seu número de identificação único (**IDFuncionario**), nome completo e detalhes de sua habilitação técnica específica.

Cada amostra gera obrigatoriamente um único requerimento de análise, identificado por um número próprio (**IDRequerimento**). Neste requerimento são armazenadas informações financeiras e administrativas relevantes para a prestação do serviço, incluindo o valor cobrado pela análise, a data em que o pagamento foi efetuado, o número da nota fiscal (quando pedida) e a data-limite acordada para a entrega do laudo final ao cliente.

As análises executadas pelo laboratório incluem diversos parâmetros físico-químicos (como aspecto visual, odor, cor aparente, cor real, pH, turbidez, condutividade e sólidos totais dissolvidos). Podem também abranger análises de concentrações de íons (tais como cloreto, nitrato, sulfato, entre outros) e metais (por exemplo, ferro, manganês, chumbo etc.), bem

como análises radiológicas e isotópicas, conforme solicitação específica do cliente. Cada parâmetro analisado deve ter registrado no sistema uma descrição clara e única.

Cada associação específica entre parâmetro e requerimento caracteriza um ensaio. Cada ensaio realizado possui um valor obtido como resultado da análise, a unidade de medida utilizada, a metodologia aplicada (ex.: potenciométrica, espectrofotométrica, etc), os valores-límite permitidos (VMP), o limite de quantificação (LQ), uma indicação clara sobre se o resultado está ou não dentro dos limites estabelecidos, além de eventuais observações técnicas adicionais.

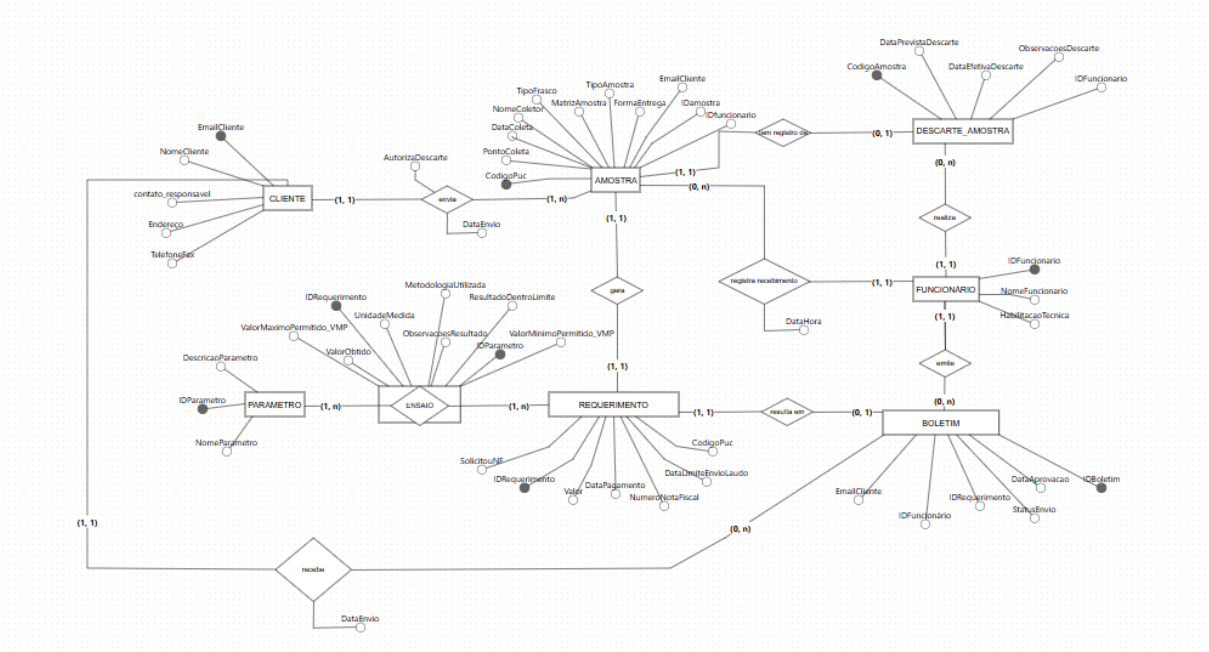
Esses resultados (ensaios) são consolidados em um único boletim de análise associado a cada requerimento, que será obrigatoriamente elaborado e enviado ao cliente pelo laboratório. Para cada boletim, o sistema deve armazenar um identificador único (**IDBoletim**), o status do envio, a data de aprovação do boletim e o funcionário tecnicamente habilitado e responsável pela validação e emissão do boletim. Quando o boletim é enviado, o sistema registra a data efetiva do envio.

Finalmente, após a emissão e envio do boletim final ao cliente, o sistema precisa controlar o processo de descarte da amostra analisada. Para isso, devem ser registrados se o cliente autoriza formalmente o descarte, a data inicialmente prevista para o descarte, a data efetiva em que o descarte foi realizado, eventuais observações adicionais relevantes sobre o procedimento, bem como o funcionário responsável por executar o descarte da amostra.

Esse conjunto detalhado de informações e processos garantirá que o LabÁguas mantenha controle rigoroso, completo e plenamente rastreável em todas as etapas do ciclo de vida das amostras analisadas, garantindo qualidade técnica, integridade e conformidade normativa do início ao fim do processo laboratorial.

2- Modelo de Entidades e Relacionamentos (MER)

Link para modelagem - Modelo ER



CLIENTE:

- **EmailCliente** (varchar, PK) — Endereço de e-mail único do cliente, utilizado como principal identificador e meio de contato.
- **NomeCliente** (varchar) — Nome completo da pessoa física ou razão social da pessoa jurídica que é cliente do laboratório.
- **ContatoResponsável** (varchar) — Nome da pessoa de contato designada pelo cliente, responsável pelas amostras enviadas e pelo acompanhamento dos processos.
- **Endereco** (varchar) - local de correspondência ou cobrança.
- **TelefoneFax** (varchar) — Número de telefone ou fax principal para contato.

FUNCIONARIO:

- **IDFuncionario** (int ou varchar, PK) — Identificador único do funcionário (ex: matrícula).
- **NomeFuncionario** (varchar) — Nome completo do funcionário.
- **HabilitacaoTecnica** (varchar) — Detalhes da qualificação ou habilitação técnica do funcionário (ex: número de registro profissional, especializações).

AMOSTRA:

- **CodigoPUC** (varchar, PK) — Código interno atribuído pelo laboratório à amostra.
- **IDAmostra** (int ou varchar) — Identificador único gerado pelo sistema para cada amostra.
- **PontoColeta** (varchar) — Descrição detalhada do local onde a amostra foi coletada.
- **DataColeta** (date) — Data em que a amostra foi coletada (formato AAAA-MM-DD).

- **DataEnvioCliente**(date) - Data que o cliente enviou a amostra
- **NomeColetor** (varchar) — Nome da pessoa responsável pela coleta da amostra.
- **TipoFrasco** (varchar) — Descrição do tipo de recipiente utilizado para armazenar a amostra.
- **MatrizAmostra** (varchar) — Tipo de material do qual a amostra é composta (ex: Água Bruta, Água Tratada).
- **TipoAmostra** (varchar) — Classificação da amostra (ex: Simples, Composta).
- **FormaEntrega** (varchar) — Método pelo qual a amostra foi entregue ao laboratório.
- ***EmailCliente*** (varchar, FK para CLIENTE.EmailCliente) — Referência o cliente que enviou a amostra.
- **ID_Funcionario**(int ou varchar, FK para funcionario)-Referencia o funcionário que recebeu a amostra
- **DataHoraRecebimento** (timestamp)-data e hora que o funcionário recebeu

REQUERIMENTO_ANALISE:

- **IDRequerimento** (varchar ou int, PK) — Identificador único do requerimento de análise.
- **DataRequerimento** (date) — Data em que o requerimento foi formalizado.
- **Valor** (float) — Custo total do serviço de análise para este requerimento.
- **DataPagamento** (date) — Data em que o pagamento pelo serviço foi efetuado.
- **NumeroNotaFiscal** (varchar) — Número da nota fiscal emitida para o requerimento.
- **DataLimiteEnvioLaudo** (date) — Prazo final para a entrega do laudo ao cliente.
- ***CodigoPuc*** (Mesmo tipo de AMOSTRA.CodigoPuc, FK para AMOSTRA.CodigoPuc) — Referência a amostra associada.

PARAMETRO:

- **IDParametro** (varchar ou int, PK) — Código único para o parâmetro de análise.
- **NomeParametro** (varchar) — Nome descritivo do parâmetro (ex: pH, Turbidez).
- **DescricaoParametro** (varchar) — Explicação breve sobre o parâmetro.
- **UnidadeMedidaPadrao** (varchar) — Unidade padrão para o resultado deste parâmetro.
- **ValorMinimoReferencia** (float) — Valor de referência mínimo padrão.
- **ValorMaximoReferencia** (float) — Valor de referência máximo padrão.

ENSAIO:

- ***NumeroRequerimento*** (PK, FK para REQUERIMENTO_ANALISE.NumeroRequerimento) — Identificador do requerimento.

- ***IDParametro*** (PK, FK para PARAMETRO.IDParametro) — Identificador do parâmetro analisado.
- **ValorObtido** (float) — Resultado medido da análise.
- **UnidadeMedida** (varchar) — Unidade do valor obtido.
- **MetodologiaUtilizada** (varchar) — Técnica ou norma usada na análise.
- **ValorMaximoPermitido_VMP** (float) — Valor máximo aceitável por norma para este ensaio específico (opcional).
- **ValorMinimoPermitido_VMP** (float) — Valor mínimo aceitável por norma para este ensaio específico (opcional).
- **ResultadoDentroLimite** (boolean) — Indica se o valor está dentro do VMP.
- **ObservacoesResultado** (varchar) — Comentários ou observações adicionais sobre o ensaio.

BOLETIM:

- **IDBoletim** (int ou varchar, PK) — Identificador único do laudo/boletim.
- **DataEnvioCliente** (DATA) — Data em que o boletim foi enviado ao cliente (opcional).
- **StatusEnvio** (varchar) — Situação atual do envio (ex: "Pendente", "Enviado").
- ***IDRequerimento***(FK para ENSAIO.IDRequerimento) — Requerimento relacionado.
- ***IDFuncionario*** (FK para FUNCIONARIO.IDFuncionario) — Técnico responsável pelo laudo.
- **DataAprovacao** (DATA) — Data em que o boletim foi aprovado pelo responsável técnico.

DESCARTE_AMOSTRA:

- ***CodigoAmostra*** (PK, FK para AMOSTRA.CodigoPUC) — Referência à amostra descartada.
- **DataPrevistaDescarte** (date) — Data planejada para o descarte.
- **DataEfetivaDescarte** (date) — Data real do descarte (opcional).
- **ObservacoesDescarte** (varchar) — Anotações sobre o descarte (opcional).
- ***IDFuncionarioDescarte*** (FK para FUNCIONARIO.IDFuncionario) — Responsável pelo descarte.

A modelagem do sistema **LabÁguas** foi feita para refletir de forma organizada e eficiente o fluxo de trabalho e as informações essenciais descritas no texto-enunciado. A criação de entidades distintas e seus relacionamentos visa separar logicamente os diferentes conjuntos de dados, evitando redundância e garantindo a integridade referencial.

Cada **CLIENTE** é único, identificado pelo **EmailCliente**, e armazena informações de contato e identificação. A entidade **AMOSTRA** é central, pois representa o objeto principal do laboratório. Cada amostra está vinculada a um **CLIENTE** (quem enviou) e a um **FUNCIONARIO** (quem a recebeu), capturando detalhes da coleta e recebimento. Opta-se pela separação da amostra de seu requerimento para que os dados da amostra sejam registrados antes da formalização completa de todos os serviços a serem realizados.

A entidade **REQUERIMENTO_ANALISE** surge da necessidade de formalizar o pedido de análise para uma amostra específica. Ela contém dados administrativos e financeiros do serviço, estando diretamente ligada a uma única **AMOSTRA** através de uma relação um-para-um, pois cada amostra gera obrigatoriamente um único requerimento principal. Manter **REQUERIMENTO_ANALISE** como uma entidade separada de **AMOSTRA** permite uma visualização de aspectos não-técnicos (parâmetros analisados) da amostra.

A entidade **PARAMETRO** funciona como um catálogo dos diversos tipos de análises que o laboratório pode realizar (pH, turbidez, concentração de chumbo, etc.). Ter uma entidade separada para parâmetros evita a repetição de nomes e descrições de análises em cada resultado individual.

Um único requerimento de análise (para uma amostra) geralmente envolve a medição de múltiplos parâmetros. Por outro lado, um mesmo tipo de parâmetro (como "pH") será analisado em diversos requerimentos diferentes. Esta é uma relação (N:N). Para modelar adequadamente essa relação N:N, introduzimos a entidade associativa **ENSAIO**.

A entidade ENSAIO representa cada análise específica realizada. Cada registro em ENSAIO representa a medição de um **PARAMETRO** específico para um determinado **REQUERIMENTO_ANALISE**. Sua chave primária é composta pelas chaves estrangeiras de **REQUERIMENTO_ANALISE** e **PARAMETRO**, garantindo que cada combinação seja única. Nela se armazenam os dados dinâmicos e específicos de cada teste, como o **ValorObtido**, a **UnidadeMedida**, etc. Sem a entidade **ENSAIO**, seria impossível registrar de forma estruturada os múltiplos resultados de diferentes parâmetros para um único requerimento.

A entidade **BOLETIM** armazena os dados do laudo final, que consolida os resultados dos ensaios de um requerimento. Ela tem uma relação de zero-para-um com **REQUERIMENTO_ANALISE**, pois cada requerimento pode resultar em um único boletim, porém se por exemplo o cliente ainda estiver com pagamento pendente ou cancelado, o boletim não será emitido pelo funcionário. Esta separação permite gerenciar os dados específicos do boletim, como data de envio, status e o responsável técnico pela sua emissão.

A entidade **DESCARTE_AMOSTRA** registra informações sobre sua o final da amostra. A relação 1:1 com **AMOSTRA** (onde **CodigoAmostra** é PK e FK) garante que cada amostra tenha, no máximo, um registro de descarte associado, ao entendimento que o cliente não autorizando o descarte a tupla não vai para o registro **DESCARTE_AMOSTRA**.

3- Regras semânticas

1. **Envio do laudo condicionado ao pagamento integral**
   Um laudo final só pode ser liberado (status de envio == TRUE) se o requerimento de análise correspondente estiver com o campo data de pagamento preenchido. Ao entendimento que este campo é preenchido por meio de um procedimento interno que

analisa a relação entre o valor pago e o total devido. Caso contrário, a tentativa de envio deve ser bloqueada e sinalizada ao usuário

IF (TotalDevido > ValorPago) DataPagamento != NULL, logo StatusEnvio = “Enviado”.

ELSE StatusEnvio = “Pendente”

2. **Regra semântica de descarte**
   No estágio de descarte de amostra, definimos que o atributo AutorizouDescarte é obrigatório e indica se o cliente deu ou não consentimento para tal atividade. Caso conteúdo == False, a tupla não é enviada para o descarte. Garantindo a eficiência e integridade do modelo.

3. **Regra validação nota fiscal**

   Para a entidade **REQUERIMENTO:**

   IF (SolicitouNF == FALSE) NumeroNotaFiscal = NULL

   ELSE NumeroNotaFiscal != NULL

---

4 - Esquema Relacional

- **CLIENTE** (EmailCliente, NomeCliente, ContatoResponsavel, Endereco, TelefoneFax)
  PK: EmailCliente

- **AMOSTRA** (CodigoPuc, IDamostra, AutorizouDescarte, PontoColeta, DataColeta, NomeColetor, TipoFrasco, MatrizAmostra, TipoAmostra, FormaEntrega, EmailCliente, DataEnvio,DataHoraRecebimento,ID_funcionario)
  PK: (CodigoPuc, IDamostra)
  FK: EmailCliente → **CLIENTE**
  FK: ID_funcionario→**FUNCIONARIO**

- **FUNCIONARIO** (IDFuncionario, NomeFuncionario, HabilitacaoTecnica)
  PK: IDFuncionario

- **DESCARTE_AMOSTRA** (CodigoAmostra, DataPrevistaDescarte, DataEfetivaDescarte, ObservacoesDescarte, IDFuncionario)
  PK: CodigoAmostra
  FK: CodigoAmostra → **AMOSTRA**
  FK: IDFuncionario → **FUNCIONARIO**

- **REQUERIMENTO** (IDRequerimento, DataRequerimento, ValorCobrado, DataPagamento, SolicitouNF, NumeroNotaFiscal, DataLimiteEnvioLaudo, CodigoPuc)
  PK: IDRequerimento
  FK: CodigoPuc → **AMOSTRA**

- **PARAMETRO** (IDParametro, NomeParametro, DescricaoParametro, UnidadeMedidaPadrao, ValorMinimoReferencia, ValorMaximoReferencia)
  PK: IDParametro

- **ENSAIO** (IDRequerimento, IDparametro, ValorObtido, UnidadeMedida, MetodologiaUtilizada, ValorMinimoPermitido_VMP, ValorMaximoPermitido_VMP, ResultadoDentroLimite)
  PK: (IDRequerimento, IDParametro)
  FK: IDRequerimento → **REQUERIMENTO**
  FK: IDParametro → **PARAMETRO**

- **BOLETIM** (IDBoletim, IDRequerimento, IDFuncionario, DataAprovacao, DataEnvio, StatusEnvio)
  PK: IDBoletim
  FK: IDRequerimento → **REQUERIMENTO**
  FK: IDFuncionario → **FUNCIONARIO**

Mapeamento ER → Relacional (usando as regras vistas em aula)

CLIENTE — (1,1) ENVIA (1,N) — AMOSTRA : “Relacionamento 1:N -> FK no lado N”
FK EmailCliente, DataEnvio e AutorizouDescarte incluído em **AMOSTRA**

AMOSTRA — (0,N)RECEBIMENTO (1,1) — FUNCIONÁRIO :
“Relacionamento N:1 -> FK no lado N”
FK IDFuncionário, DataHoraRecebimento incluídos em **AMOSTRA**

AMOSTRA — (1,1) TEM REGISTRO DE (0,1) — DESCARTE_AMOSTRA:“Relacionamento 1:1, lado opcional recebe FK&PK”
PK igual à PK de AMOSTRA implementa dependência total;

FUNCIONÁRIO — (1,1) EMITE (0,N) — BOLETIM: “Relacionamento 1:N -> FK no lado N”
FK IDFuncionário incluída em **BOLETIM**

REQUERIMENTO — (1,1) RESULTA_EM (0,1) — BOLETIM:“Relacionamento 1:1 -> lado opcional recebe FK&PK ”
FK IDrequerimento incluída em **BOLETIM**

AMOSTRA — (1,1) GERA (1,1) — REQUERIMENTO:“Relacionamento 1:1 -> lado opcional recebe FK&PK ”
FK de amostra em REQUERIMENTO
FK CodigoAmostra incluída em **REQUERIMENTO**

PARAMETRO — (1,N) ENSAIO (1,N) — REQUERIMENTO: “Relacionamento N:N -> tabela associativa”
Decidimos colocar ENSAIO como classe associativa, que associa PARAMETRO com requerimento, todos os parâmetros avaliados em ensaio vão para o requerimento, portanto nesse caso como criamos essa entidade associativa resultante da relação entre parâmetro e requerimento, a PK de ENSAIO deve ser {FK de PARAMETRO, FK de REQUERIMENTO}
FK IDRequerimento incluída em **ENSAIO como parte da PK**
FK IDParametro incluída em **ENSAIO como parte da PK**

—----------

5 - Criação de tabelas (SQL - DDL)

```
CREATE TABLE Cliente (
   EmailCliente          VARCHAR(120) PRIMARY KEY,
   NomeCliente           VARCHAR(120) NOT NULL,
   contato_responsavel               VARCHAR(120),
   Endereco              VARCHAR(255),
   TelefoneFax           VARCHAR(25)
);

CREATE TABLE Funcionario (
   IDFuncionario     VARCHAR(20) PRIMARY KEY,
   NomeFuncionario   VARCHAR(100) NOT NULL,
   HabilitacaoTecnica VARCHAR(80)
```

```
);

CREATE TABLE Amostra (
   CodigoPUC      VARCHAR(20) NOT NULL,
   IDAmostra      VARCHAR(20) NOT NULL,
   AutorizouDescarte BOOLEAN     NOT NULL,
   PontoColeta    VARCHAR(255),
   DataEnvioCliente      DATE   NOT NULL,
   DataColeta     DATE,
   NomeColetor    VARCHAR(100),
   TipoFrasco     VARCHAR(50),
   MatrizAmostra  VARCHAR(50),
   TipoAmostra    VARCHAR(50),
   FormaEntrega   VARCHAR(50),
   DataHoraRecebimento TIMESTAMP WITHOUT TIME,
   ID_Funcionario VARCHAR(20),
   CONSTRAINT FK_Amostra_Funcionario
      FOREIGN KEY (ID_Funcionario)
      REFERENCES Funcionario (IDFuncionario)
   EmailCliente   VARCHAR(120) NOT NULL,
   CONSTRAINT FK_Amostra_Cliente
      FOREIGN KEY (EmailCliente)
      REFERENCES Cliente (EmailCliente)
);

ALTER TABLE Amostra
   ADD CONSTRAINT PK_Amostra
      PRIMARY KEY (CodigoPUC);

CREATE TABLE Requerimento_Analise (
   IDRequerimento       VARCHAR(20) PRIMARY KEY,
   DataRequerimento     DATE    NOT NULL,
   Valor                NUMERIC(12,2),
   DataPagamento        DATE,
   NumeroNotaFiscal     VARCHAR(30),
   DataLimiteEnvioLaudo DATE,
   CodigoPUC            VARCHAR(20) NOT NULL,
   CONSTRAINT FK_Requ_Amostra
      FOREIGN KEY (CodigoPUC)
      REFERENCES Amostra (CodigoPUC)
);

CREATE TABLE Parametro (
   IDParametro          VARCHAR(20) PRIMARY KEY,
```

```
    NomeParametro         VARCHAR(60)  NOT NULL,
    DescricaoParametro    TEXT,
    UnidadeMedidaPadrao   VARCHAR(20),
    ValorMinimoReferencia NUMERIC(14,4),
    ValorMaximoReferencia NUMERIC(14,4)
);

CREATE TABLE Ensaio (
    NumeroRequerimento       VARCHAR(20) NOT NULL,
    IDParametro              VARCHAR(20) NOT NULL,
    ValorObtido              NUMERIC(14,4),
    UnidadeMedida            VARCHAR(20),
    MetodologiaUtilizada     VARCHAR(255),
    ValorMaximoPermitido_VMP NUMERIC(14,4),
    ValorMinimoPermitido_VMP NUMERIC(14,4),
    ResultadoDentroLimite    BOOLEAN,
    ObservacoesResultado     TEXT,
    CONSTRAINT PK_Ensaio PRIMARY KEY (NumeroRequerimento, IDParametro),
    CONSTRAINT FK_Ensaio_Requerimento
        FOREIGN KEY (NumeroRequerimento)
        REFERENCES Requerimento_Analise (IDRequerimento),
    CONSTRAINT FK_Ensaio_Parametro
        FOREIGN KEY (IDParametro)
        REFERENCES Parametro (IDParametro)
);

CREATE TABLE Boletim (
    IDBoletim         VARCHAR(20) PRIMARY KEY,
    DataEnvioCliente  DATE,
    StatusEnvio       VARCHAR(20),
    IDRequerimento    VARCHAR(20) NOT NULL,
    IDFuncionario     VARCHAR(20) NOT NULL,
    DataAprovacao     DATE,
    CONSTRAINT FK_Boletim_Requerimento
        FOREIGN KEY (IDRequerimento)
        REFERENCES Requerimento_Analise (IDRequerimento),
    CONSTRAINT FK_Boletim_Funcionario
        FOREIGN KEY (IDFuncionario)
        REFERENCES Funcionario (IDFuncionario)
);

CREATE TABLE Descarte_Amostra (
    CodigoAmostra            VARCHAR(20) PRIMARY KEY,
    DataPrevistaDescarte     DATE,
```

```
    DataEfetivaDescarte      DATE,
    ObservacoesDescarte      TEXT,
    IDFuncionarioDescarte    VARCHAR(20) NOT NULL,
    CONSTRAINT FK_Descarte_Amostra
        FOREIGN KEY (CodigoAmostra)
        REFERENCES Amostra (CodigoPUC)
);

ALTER TABLE Descarte_Amostra
    ADD CONSTRAINT FK_Descarte_Funcionario
        FOREIGN KEY (IDFuncionarioDescarte)
        REFERENCES Funcionario (IDFuncionario);
```

6 - Inserção de tuplas nas tabelas (SQL - DML)

Para **cada tabela**, o comando de inserção de exemplo e, na sequência, um comando que **dispara violação** de restrição (PK ou FK)

CLIENTE:
Inserção válida :

```
INSERT INTO Cliente (
  EmailCliente,
  NomeCliente,
  ContatoResponsavel,
  Endereco,
  TelefoneFax
) VALUES (
  'contato@aguasexemplo.com.br',
  'Águas Exemplo Ltda.',
  'Maria souza',
  'Rua das Flores, 123 – Rio de Janeiro/RJ',
  '(21) 3527-1814'
);
```

```
INSERT INTO Cliente (EmailCliente, NomeCliente, contato_responsavel, Endereco, TelefoneFax) VALUES
('novaempresa@cliente.com', 'Nova Empresa S/A', 'REBECCA ROCHA', 'Rua Z, 101, Petrópolis, RJ', '(24) 1357-2468');
```

Inserção violando(repetindo o mesmo EMAIL)

```
INSERT INTO Cliente (
  EmailCliente, NomeCliente
) VALUES (
  'contato@aguasexemplo.com.br', 'Outro Cliente'
);
```

```
ERROR:  duplicate key value violates unique constraint "cliente_pkey"
DETAIL:  Key (emailcliente)=(contato@aguasexemplo.com.br) already exists.
```

FUNCIONARIO:
Inserção válida :

```
INSERT INTO Funcionario (
  IDFuncionario,
  NomeFuncionario,
  HabilitacaoTecnica
) VALUES (
  'F001',
  'José Godoy',
  'Química Analítica'
);
INSERT INTO Funcionario (IDFuncionario, NomeFuncionario, HabilitacaoTecnica)
VALUES
('F018', 'Ricardo Nunes', 'Técnico de Campo');
```

Inserção violando(repetindo o ID)

```
INSERT INTO Funcionario (
  IDFuncionario, NomeFuncionario
) VALUES (
  'F001', 'Duplicado Silva'
);
```

```
ERROR:  duplicate key value violates unique constraint "funcionario_pkey"
DETAIL:  Key (idfuncionario)=(F001) already exists.
```

FUNCIONARIO:
Inserção válida :

```
INSERT INTO Funcionario (
```

```
  IDFuncionario,
  NomeFuncionario,
  HabilitacaoTecnica
) VALUES (
  'F001',
  'José Godoy',
  'Química Analítica'
);
INSERT INTO Funcionario (IDFuncionario, NomeFuncionario, HabilitacaoTecnica)
VALUES
('F018', 'Ricardo Nunes', 'Técnico de Campo');
```

Inserção violando(repetindo o ID)

```
INSERT INTO Funcionario (
  IDFuncionario, NomeFuncionario
) VALUES (
  'F001', 'Duplicado Silva'
);
```

AMOSTRA:

Inserção válida:

```
-- Inserção válida
INSERT INTO Amostra (
  CodigoPUC,
  IDAmostra,
  PontoColeta,
  DataColeta,
  NomeColetor,
  TipoFrasco,
  MatrizAmostra,
  TipoAmostra,
  FormaEntrega,
  EmailCliente,
  autorizadescarte,
  DataEnvioCliente
  DataHoraRecebimento
) VALUES (
  'LB-0001-23',
  'A01',
  'Capacete I – Rio Branco',
  '2025-05-20',
  'Antônio Silva',
  'Vidro 500 mL',
  'Água de Consumo',
```

'Fonte’,
'Coleta Própria',
'contato@aguasexemplo.com.br',
TRUE,
'2025-05-26'
'2025-06-24 15:45:00'
);

Inserção violando(cliente inexistente):

```
INSERT INTO Amostra (
  CodigoPUC, IDAmostra, EmailCliente
) VALUES (
  'LB-0002-23','A02','naoexiste@cliente.com'
);
```

```
ERROR:  insert or update on table "amostra" violates foreign key constraint "fk_amostra_cliente"
DETAIL:  Key (emailcliente)=(naoexiste@cliente.com) is not present in table "cliente".
```

Inserção violando(2)(mesmo CodigoPUC):

```
INSERT INTO Amostra (CodigoPUC, IDAmostra, EmailCliente)
VALUES ('LB-0001-23','A03','contato@aguasexemplo.com.br');
```

```
ERROR:  duplicate key value violates unique constraint "pk_amostra"
DETAIL:  Key (codigopuc)=(LB-0001-23) already exists.
```

REQUERIMENTO_ANALISE:

Inserção válida:

```
INSERT INTO Requerimento_Analise (
  IDRequerimento,
  DataRequerimento,
  Valor,
  DataPagamento,
  NumeroNotaFiscal,
  DataLimiteEnvioLaudo,
  CodigoPUC
) VALUES (
  'R1005',
  '2025-05-21',
  450.00,
  NULL,
  NULL,
  '2025-05-28',
  'LB-0001-23'
);
```

Inserção violando (amostra inexistente):

```
INSERT INTO Requerimento_Analise (IDRequerimento, DataRequerimento, CodigoPUC)
VALUES ('R1002','2025-05-22','LB-9999-23');
```

```
ERROR:  insert or update on table "requerimento_analise" violates foreign key constraint "fk_requ_amostra"
DETAIL:  Key (codigopuc)=(LB-9999-23) is not present in table "amostra".
```

Inserção violando(2)(mesmo IDRequerimento):

INSERT INTO Requerimento_Analise (IDRequerimento, DataRequerimento, CodigoPUC)
VALUES ('R1005','2025-05-22','LB-0001-23');

PARAMETRO:

Inserção válida:

INSERT INTO Parametro (
IDParametro,
NomeParametro,
DescricaoParametro,
UnidadeMedidaPadrao,
ValorMinimoReferencia,
ValorMaximoReferencia
) VALUES (
'P_H',
'pH',
'Potencial hidrogeniônico',
'adimensional',
6.0,
9.0
);

Inserção violando(mesmo IDParametro):

INSERT INTO Parametro (IDParametro, NomeParametro)
VALUES ('P_H','Duplicado pH');

```
ERROR:  duplicate key value violates unique constraint "parametro_pkey"
DETAIL:  Key (idparametro)=(P_H) already exists.
```

ENSAIO:

Inserção válida:

INSERT INTO Ensaio (
NumeroRequerimento,
IDParametro,
ValorObtido,
UnidadeMedida,
MetodologiaUtilizada,
ValorMaximoPermitido_VMP,
ValorMinimoPermitido_VMP,
ResultadoDentroLimite,
ObservacoesResultado
) VALUES (

'R1001',
'P_H',
7.4,
'adimensional',
'Potenciometria',
9.0,
6.0,
TRUE,
NULL
);

Inserção violando (requerimento inexistente):

INSERT INTO Ensaio (NumeroRequerimento, IDParametro)
VALUES ('R9999','P_H');

```
ERROR:  insert or update on table "ensaio" violates foreign key constraint "fk_ensaio_requerimento"
DETAIL:  Key (numerorequerimento)=(R9999) is not present in table "requerimento_analise".
```

Inserção violando(2) (mesma dupla):

INSERT INTO Ensaio (NumeroRequerimento, IDParametro)
VALUES ('R1001','P_H');

```
ERROR:  duplicate key value violates unique constraint "pk_ensaio"
DETAIL:  Key (numerorequerimento, idparametro)=(R1001, P_H) already exists.
```

BOLETIM:

Inserção válida:

INSERT INTO Boletim (
IDBoletim,
DataEnvioCliente,
StatusEnvio,
IDRequerimento,
IDFuncionario,
DataAprovacao
) VALUES (
'B2025001',
NULL,
'Pendente',
'R1001',
'F001',
'2025-05-22'
);

Inserção violando (Funcionário inexistente):

INSERT INTO Boletim (IDBoletim, IDRequerimento, IDFuncionario)
VALUES ('B2025002','R1001','F999');

```
ERROR:  insert or update on table "boletim" violates foreign key constraint "fk_boletim_funcionario"
DETAIL:  Key (idfuncionario)=(F999) is not present in table "funcionario".
```

Inserção violando(2) (mesmo IDBoletim)

INSERT INTO Boletim (IDBoletim, IDRequerimento, IDFuncionario)
VALUES ('B2025001','R1001','F001');

```
ERROR:  duplicate key value violates unique constraint "boletim_pkey"
DETAIL:  Key (idboletim)=(B2025001) already exists.
```

DESCARTE_AMOSTRA:

Inserção válida:

```
INSERT INTO Descarte_Amostra (
  CodigoAmostra,
  DataPrevistaDescarte,
  DataEfetivaDescarte,
  ObservacoesDescarte,
  IDFuncionarioDescarte
) VALUES (
  'LB-0001-23',
  '2025-06-05',
  NULL,
  NULL,
  'F001'
);
```

Inserção violando (amostra inexistente):

```
INSERT INTO Descarte_Amostra (
  CodigoAmostra, IDFuncionarioDescarte
) VALUES (
  'LB-9999-23', 'F001'
);
```

```
ERROR:  insert or update on table "descarte_amostra" violates foreign key constraint "fk_descarte_amostra"
DETAIL:  Key (codigoamostra)=(LB-9999-23) is not present in table "amostra".
```

Inserção violando (2) (funcionário inexistente):

```
INSERT INTO Descarte_Amostra (
  CodigoAmostra, IDFuncionarioDescarte
) VALUES (
  'LB-0001-23', 'F999'
);
```

```
ERROR:  duplicate key value violates unique constraint "descarte_amostra_pkey"
DETAIL:  Key (codigoamostra)=(LB-0001-23) already exists.
```

LINK PARA TODAS AS INSERÇÕES:

Inserções para o banco LabÁguas

7- Consultas Álgebra Relacional

Consulta 1

Texto em Português:

Listar o e-mail e o nome de todos os clientes cujos boletins ainda estão com StatusEnvio = ‘Pendente’.

Solução em Álgebra Relacional:

R1 ← σ_StatusEnvio='Pendente'(Boletim)

R2 ← Amostra ⋈ Requerimento_Analise

R3 ← R2 ⋈ R1

R4 ← Cliente ⋈ R3

RESPOSTA1 ← π_EmailCliente, NomeCliente(R4)

Consulta 2

Texto em Português:

Exibir o código PUC da amostra, o IDParâmetro e o valor obtido de todos os ensaios cujo **ValorObtido > ValorMaximoPermitido_VMP**.

Solução em Álgebra Relacional:

R1 ← σ_ValorObtido > ValorMaximoPermitido_VMP(Ensaio)

R2 ← Amostra ⋈ Requerimento_Analise

R3 ← R2 ⋈ R1

R4 ← R3 ⋈ Parametro

RESPOSTA2 ← π_CodigoPUC,IDParametro,ValorObtido(R4)

Consulta 3

Texto em Português:

Listar todas as informações que um boletim necessita para ser feito, ou seja,EmailCliente, NomeCliente, CodigoPUC, IDAmostra, DataColeta, DataRequerimento, IDParametro, ValorObtido, UnidadeMedida, MetodologiaUtilizada, ValorMinimoPermitido_VMP, ValorMaximoPermitido_VMP, IDBoletim, DataEnvioCliente, StatusEnvio, NomeFuncionario

Solução em Álgebra Relacional:

A0 ← σCodigoPUC='LB-0001-23'(Amostra)

R1 ← Cliente ⋈ A0

R2 ← R1 ⋈ Requerimento_Analise

R3 ← R2 ⋈ Ensaio

R4 ← R3 ⋈ Boletim

R5 ← R4 ⋈ Funcionario

BOLETIM_LB000123 ← π

EmailCliente,

NomeCliente,

CodigoPUC,

IDAmostra,

DataColeta,

DataRequerimento,

IDParametro,

ValorObtido,

UnidadeMedida,

MetodologiaUtilizada,

ValorMinimoPermitido_VMP,

ValorMaximoPermitido_VMP,

IDBoletim,

DataEnvioCliente,

StatusEnvio,

NomeFuncionario

(R5)

Consulta 4:

Texto em Português:

Listar o e-mail de todos os clientes que já tiveram **ensaios** para **todos** os parâmetros cadastrados.

Solução em Álgebra Relacional:

R1a ← Cliente ⋈ Amostra

R1b ← R1a ⋈ Requerimento_Analise

R1c ← R1b ⋈ Ensaio

R1 ← π_EmailCliente,IDParametro(R1c)

R2 ← π_IDParametro(Parametro)

RESPOSTA4 ← R1 ÷ R2

→Resultado: conjunto de EmailCliente que aparecem com *todos* os IDParametro em R2

Consulta 5:
Texto em Português:

Listar o código PUC de todas as amostras com as quais o funcionário de código 'F001' esteve diretamente envolvido, seja no **recebimento** inicial da amostra ou no seu **descarte** final.

Solução em Álgebra Relacional:

π CodigoPUC (σ IDFuncionario = 'F001' (Amostra))
∪
π CodigoPUC (ρ CodigoPUC/CodigoAmostra (σ IDFuncionarioDescarte='F001' (Descarte_Amostra)))

8- Consultas (SQL - DML)

Consulta 1 - Listar todas as amostras cuja matriz seja 'Água Bruta', que foram coletadas durante o primeiro semestre do ano de 2025 e cuja forma de entrega ao laboratório tenha sido 'Coleta Presencial'. Para cada uma dessas amostras, exibir seu **CodigoPUC**, o **PontoColeta**, a **DataColeta** e, buscando na tabela de clientes, o **NomeCliente** e o **ContatoResponsavel** do cliente que enviou a amostra. Os resultados devem ser ordenados pela data de coleta da amostra, da mais antiga para a mais recente.

SQL:

```
SELECT
   a.CodigoPUC,
   a.PontoColeta,
   a.DataColeta,
   c.NomeCliente,
   c.contato_responsavel
FROM
   Amostra a
INNER JOIN
   Cliente c ON a.EmailCliente = c.EmailCliente
WHERE
   a.MatrizAmostra = 'Água Bruta'
   AND a.DataColeta >= '2025-01-01' AND a.DataColeta <= '2025-06-30'
   AND a.FormaEntrega = 'Coleta Presencial'
ORDER BY
   a.DataColeta ASC;
```

Álgebra relacional:

$A1 \leftarrow \sigma(\text{seleção})\{\text{MatrizAmostra} = \text{'Água Bruta'} \wedge \text{DataColeta} \geq \text{'2025-01-01'} \wedge \text{DataColeta} \leq \text{'2025-06-30'} \wedge \text{FormaEntrega} = \text{'Presencialmente'}\}\ (\text{AMOSTRA})$

$A2 \leftarrow A1 \bowtie(\text{junção natural})\{A1.\text{EmailCliente} = \text{CLIENTE.EmailCliente}\}\ \text{CLIENTE}$

$A3 \leftarrow \pi(\text{projeção})\{\text{CodigoPUC, PontoColeta, DataColeta, NomeCliente, contato\_responsavel}\}\ (A2)$

$\text{Resultado} \leftarrow \tau\{\text{DataColeta ASC}\}\ (A3)$

τ = sort (ordenação)

Consulta 2 - Identificar os parâmetros (exibindo o **NomeParametro**) que mais frequentemente apresentaram ensaios com o **ResultadoDentroLimite** igual a **FALSE** (ou seja, não conformes) durante o ano de 2025. Para cada um desses parâmetros, listar o nome do parâmetro e a contagem

total de ensaios não conformes associados a ele. Os resultados devem ser ordenados em ordem decrescente pela contagem de ensaios não conformes, exibindo primeiro os parâmetros com maior número de falhas. Considerar apenas os requerimentos de análise que foram formalizados (cuja **DataRequerimento** se encontra) no ano de 2025.

SQL:

```
SELECT
   p.NomeParametro,
   COUNT(*) AS TotalNaoConforme
FROM
   Ensaio e
INNER JOIN
   Requerimento_Analise ra ON e.NumeroRequerimento = ra.IDRequerimento
INNER JOIN
   Parametro p ON e.IDParametro = p.IDParametro
WHERE
   e.ResultadoDentroLimite = FALSE
   AND EXTRACT(YEAR FROM ra.DataRequerimento) = 2025
GROUP BY
   p.NomeParametro
ORDER BY
   TotalNaoConforme DESC;
```

Álgebra Relacional:

R1 ← σ(seleção){ENSAIO.ResultadoDentroLimite = FALSE ∧ ano(REQUERIMENTO_ANALISE.DataRequerimento) = 2025} (ENSAIO ⋈(junção natural){ENSAIO.NumeroRequerimento = REQUERIMENTO_ANALISE.IDRequerimento} REQUERIMENTO_ANALISE)

R2 ← R1 ⋈ (junção natural){ENSAIO.IDParametro = PARAMETRO.IDParametro} PARAMETRO

R3 ← γ(agrupamento){PARAMETRO.NomeParametro, count(*) → TotalNaoConforme} (R2)

Resultado <-- τ_{TotalNaoConforme DESC} (R3)

Consulta 3 - Listar todos os funcionários que possuem a habilitação técnica especificada como 'Química Analítica' e que são atualmente responsáveis pelo descarte de pelo menos uma amostra para a qual a autorização de descarte por parte do cliente foi negada. Para cada funcionário que satisfaça essas condições, exibir seu **IDFuncionario**, **NomeFuncionario** e o **CodigoAmostra** da(s) amostra(s) pendente(s) de autorização de descarte pelas quais ele é responsável. Se um mesmo funcionário for responsável por múltiplas amostras nesta situação, todas devem ser listadas.

SQL:

```
SELECT
  f.IDFuncionario,
  f.NomeFuncionario,
```

```
  da.CodigoAmostra
FROM
  Funcionario AS f
  JOIN Descarte_Amostra AS da
    ON f.IDFuncionario = da.IDFuncionarioDescarte
  JOIN Amostra AS a
    ON a.CodigoPUC = da.CodigoAmostra
WHERE
  f.HabilitacaoTecnica = 'Química Analítica'
  AND a.autorizadescarte = FALSE;
```

Álgebra Relacional:

```
π IDFuncionario, NomeFuncionario, CodigoAmostra (
    (‘
      σ HabilitacaoTecnica='Química Analítica' (FUNCIONARIO)
      ⋈ IDFuncionario=IDFuncionarioDescarte
      DESCARTE_AMOSTRA
    )
    ⋈ CodigoPUC=CodigoAmostra
    (σ autorizadescarte=FALSE (AMOSTRA))
)
```

9- Revisão parte G1

1. Texto enunciado: Não fizemos nenhuma alteração significativa ou que mudasse a estrutura do trabalho. Apenas nos atentamos para a coesão geral do texto, portanto, corrigimos detalhes.
2. Modelo MER: Incluímos a chave IDfuncionario para a entidade **AMOSTRA** no modelo gráfico. Alteramos a descrição textual, adicionando dois novos atributo a tabela **AMOSTRA**, sendo eles “DataHoraRecebimento” e “IdFuncionario” (FK para **FUNCIONARIO**). Não estavam presentes e se faziam necessários.
3. Restrições semânticas: Sem alterações. Todas coerentes com o projeto.
4. Esquema lógico-relacional: Seguindo a lógica da alteração do tópico (2), alteramos o esquema na parte “AMOSTRA-RECEBIMENTO-FUNCIONARIO”.
5. SQL DDL: Incluímos DataHoraRecebimento em **AMOSTRA** com o domínio TIMESTAMP para dar mais precisão a esse dado, com base no contexto do trabalho.
6. SQL DML: Mínimas alterações e todas seguindo a lógica passada nos tópicos anteriores.
7. Consultas Álgebra Relacional: Alteramos apenas a consulta 5, à consulta era considerada sem contexto e com pouca utilidade, portanto trocamos ela para uma consulta mais coerente e útil para o nosso contexto.

8. Consultas SQL: Sem alterações. Todas consultas coerentes e funcionando perfeitamente.

10- Novas consultas (SQL)

1) **Listar cada cliente e o número total de ensaios que apresentaram resultado fora dos limites, exibindo apenas clientes com ao menos um ensaio não conforme, ordenados do maior para o menor número de não conformidades.**

```
SELECT c.nomecliente,
    COUNT(*) AS ensaios_fora_limite
FROM   cliente        c
JOIN   amostra        a ON a.emailcliente   = c.emailcliente
JOIN   requerimento_analise r ON r.codigopuc = a.codigopuc
JOIN   ensaio         e ON e.numerorequerimento = r.idrequerimento
WHERE  e.resultadodentrolimite = FALSE
GROUP  BY c.nomecliente
ORDER  BY ensaios_fora_limite DESC;
```

2) **Listar os funcionários que ainda não foram responsáveis pela aprovação de nenhum boletim de laudo.**

```
SELECT f.idfuncionario,
    f.nomefuncionario
FROM   funcionario f
WHERE  NOT EXISTS (
        SELECT 1
        FROM   boletim b
        WHERE  b.idfuncionario = f.idfuncionario
          AND  b.dataaprovacao IS NOT NULL
      );
```

3) **Calcular o faturamento mensal do laboratório, exibindo para cada ano e mês a soma total dos valores dos requerimentos de análise cujo pagamento já foi registrado.**

```
SELECT EXTRACT(YEAR  FROM datapagamento) AS ano,
    EXTRACT(MONTH FROM datapagamento) AS mes,
    SUM(valor)                  AS valor_total
FROM   requerimento_analise
WHERE  datapagamento IS NOT NULL
GROUP  BY ano, mes
ORDER  BY ano, mes;
```

4) **Mostra quantas amostras cada cliente já enviou ao laboratório, em ordem decrescente.Só devem ser exibidos os clientes que ja enviaram pelo menos 2 amostras, além disso ordene de forma decrescente o total de amostras calculado.**

```
SELECT c.nomecliente,
      COUNT(*) AS total_amostras
FROM   cliente c
JOIN   amostra a ON a.emailcliente = c.emailcliente
GROUP  BY c.nomecliente
HAVING COUNT(*) > 1
ORDER  BY total_amostras DESC;
```

5) **Identifica requerimentos cujo prazo já passou e que ainda não têm boletim enviado e exiba o id a data limite e o nome do cliente.**

```
WITH boletins_enviados AS (
   SELECT idrequerimento
   FROM   boletim
   WHERE  statusenvio = 'Enviado'
)
SELECT r.idrequerimento,
      r.datalimiteenviolaudo,
      c.nomecliente
FROM   requerimento_analise r
JOIN   amostra  a ON a.codigopuc    = r.codigopuc
JOIN   cliente  c ON c.emailcliente = a.emailcliente
WHERE  r.datalimiteenviolaudo < CURRENT_DATE
       AND      r.idrequerimento NOT IN (SELECT idrequerimento FROM
boletins_enviados);
```

6) **Para cada ponto de coleta, calcula média, mínimo e máximo dos valores de pH medidos.**

```
SELECT a.pontocoleta,
    ROUND(AVG(e.valorobtido::NUMERIC),2) AS media_ph,
    MIN(e.valorobtido::NUMERIC)        AS min_ph,
    MAX(e.valorobtido::NUMERIC)        AS max_ph
FROM   amostra            a
JOIN   requerimento_analise r ON r.codigopuc        = a.codigopuc
JOIN   ensaio             e ON e.numerorequerimento = r.idrequerimento
WHERE  e.idparametro = 'P001'        -- pH
GROUP  BY a.pontocoleta
ORDER  BY media_ph DESC;
```

11-

a)Criação de views

VIEW 1

Agrupa todos os valores de requerimentos ainda não pagos por cliente, para facilitar relatórios de cobrança.

```
CREATE OR REPLACE VIEW vw_cliente_saldo AS
SELECT
  c.emailcliente,
  c.nomecliente,
  SUM(r.valor) AS saldo_pendente
FROM   cliente c
JOIN   amostra a ON a.emailcliente = c.emailcliente
JOIN   requerimento_analise r ON   r.codigopuc = a.codigopuc
WHERE  r.datapagamento IS NULL
GROUP  BY c.emailcliente, c.nomecliente;
```

Utilidade: por exemplo agora, se eu quiser saber o saldo_pendente do cliente com email aquaprime@cliente.com basta eu fazer a seguinte consulta:

```
SELECT *
FROM vw_cliente_saldo
WHERE emailcliente = 'aquaprime@cliente.com';
```

emailcliente	nomecliente	saldo_pendente
aquaprime@cliente.com	AquaPrime Saneamento	192.50

VIEW 2

Lista todos os requerimentos cujo prazo de envio de laudo já venceu, com o nome do cliente e dias de atraso.

```
CREATE OR REPLACE VIEW vw_reqs_atrasados AS
SELECT
  r.idrequerimento,
  a.codigopuc,
  c.nomecliente,
  r.datalimiteenviolaudo,
  (CURRENT_DATE - r.datalimiteenviolaudo) AS dias_atraso
FROM   requerimento_analise r
JOIN   amostra a ON a.codigopuc = r.codigopuc
JOIN   cliente c ON c.emailcliente = a.emailcliente
WHERE  r.datalimiteenviolaudo < CURRENT_DATE;
```

Utilidade : Exibir os três requerimentos mais atrasados (em dias) e o cliente responsável.

Consulta:

```
SELECT
  idrequerimento,
  nomecliente,
  dias_atraso
FROM   vw_reqs_atrasados
ORDER  BY dias_atraso DESC
LIMIT  3;
```

idrequerimento	nomecliente	dias_atraso
R1005	Águas Exemplo Ltda.	28
R001	Águas Exemplo Ltda.	20
R002	ASA EMPRESA	19

b) View com Check Option

```
CREATE VIEW v_boletins_para_emissao AS
SELECT IDBoletim, IDRequerimento, IDFuncionario, DataAprovacao, StatusEnvio
FROM Boletim
WHERE StatusEnvio = 'Pendente'
WITH CHECK OPTION;
```

1. Caso de sucesso:

INSERT INTO V_Boletins_Para_Emissao (IDBoletim, IDRequerimento, IDFuncionario, StatusEnvio)
VALUES ('B2025002', 'R1005', 'F001', 'Pendente');

2. Caso de erro violação CHECK OPTION:

INSERT INTO V_Boletins_Para_Emissao (IDBoletim, IDRequerimento, IDFuncionario, StatusEnvio)
VALUES ('B2025003', 'R1005', 'F001', 'Enviado');

12-

a) Criação de funções

- **FUNÇÃO 1 - QUANTIDADE DE PARAMETROS FORA DO LIMITE**

Entrada e saída de dados:
Recebe um único parâmetro de entrada (cod_puc), que é o código da amostra.
Devolve um valor integer sendo a quantidade de parâmetros (ensaios) fora do limite.

Funcionamento:
Cria a variável qtd, inicializada com 0, que armazenará o total.
Executa um SELECT COUNT(*) contando todas as linhas de ENSAIO para o requerimento associado à amostra cod_puc cujo resultado não está dentro do limite.
Coloca esse número em qtd e, em seguida, retorna qtd.

Comando para criar função 2:

```
CREATE OR REPLACE FUNCTION qtd_parametros_fora_limite(cod_puc varchar)
RETURNS integer AS $$
DECLARE
   qtd integer := 0;
BEGIN
   SELECT COUNT(*) INTO qtd
   FROM ENSAIO e
   JOIN REQUERIMENTO_ANALISE r ON e.NumeroRequerimento = r.IDRequerimento
   WHERE r.CodigoPuc = cod_puc
     AND (e.ResultadoDentroLimite = false OR e.ResultadoDentroLimite IS NULL);
   RETURN qtd;
END;
$$ LANGUAGE plpgsql;
```

Testando função 1:

```
SELECT
 'LB-0101-23' AS amostra,
 qtd_parametros_fora_limite('LB-0101-23') AS qtd_fora
UNION ALL
SELECT
 'LB-0102-23',
 qtd_parametros_fora_limite('LB-0102-23');
```

amostra	qtd_fora
LB-0101-23	0
LB-0102-23	1

- **FUNÇÃO 2 - Soma de valores dos requerimentos em dívida**

Entrada e saída de dados:
Recebe um único parâmetro de entrada (p_email) e retorna um número com duas casas decimais representando a soma dos valores de todos os requerimentos ainda não pagos.

Funcionamento:
Cria total_pendente para armazenar temporariamente o resultado do SUM.
JOINs: atravessa cliente > amostra > requerimento_analise para pegar todos os requerimentos daquele cliente.
Filtro: r.datapagamento IS NULL garante que só some valores de serviços não pagos.
O SUM(r.valor) retorna NULL caso não haja nenhum registro; caso contrário, retorna a soma.
Transforma NULL em 0.00 para que a função nunca retorne valor nulo.

Comando para criar função 2:

```
CREATE OR REPLACE FUNCTION valor_pendente_cliente(
        p_email VARCHAR)
RETURNS NUMERIC(12,2) AS $$
DECLARE
   total_pendente NUMERIC(12,2);
BEGIN
   SELECT SUM(r.valor)
     INTO total_pendente
   FROM   cliente             c
   JOIN   amostra             a ON a.emailcliente = c.emailcliente
   JOIN   requerimento_analise r ON r.codigopuc    = a.codigopuc
   WHERE  c.emailcliente = p_email
     AND  r.datapagamento IS NULL;

   IF total_pendente IS NULL THEN
      total_pendente := 0;
   END IF;

   RETURN total_pendente;
END;
$$ LANGUAGE plpgsql;
```

Testando função 2:

```
SELECT valor_pendente_cliente('hydrosys@cliente.com') AS pendente_hydrosys;
```

pendente_hydrosys
195.50

b) Procedimento Armazenado

Descrição do procedimento de registrar pagamento:

Este procedimento armazenado tem a função de registrar ou atualizar a data de pagamento de um determinado requerimento de análise no sistema do laboratório. Através dele, é possível alterar o campo datapagamento da tabela requerimento_analise, garantindo o controle adequado dos pagamentos efetuados pelos serviços de análise de amostras.

O procedimento recebe dois parâmetros:

- p_requerimento: identificador do requerimento de análise a ser atualizado.
- p_data_pagamento: data em que o pagamento foi realizado.

Comando para criar procedimento:

```
CREATE OR REPLACE PROCEDURE registrar_pagamento(
   p_requerimento   VARCHAR,
   p_data_pagamento DATE
)
AS $$
BEGIN
  UPDATE requerimento_analise
    SET datapagamento = p_data_pagamento
  WHERE idrequerimento = p_requerimento;
END;
$$ LANGUAGE plpgsql;
```

TESTANDO:

1. Estado anterior:

```
SELECT idrequerimento, datapagamento
FROM   requerimento_analise
WHERE  idrequerimento = 'R102';
```

idrequerimento	datapagamento
R102	*NULL*

2. Chamada:

```
CALL registrar_pagamento('R102', '2025-07-20');
```

3. Estado posterior:

```
SELECT idrequerimento, datapagamento
FROM   requerimento_analise
WHERE  idrequerimento = 'R102';
```

idrequerimento	datapagamento
R102	2025-07-20

c) TRIGGER

Explicação regra de negócio escolhida:

Um laudo só pode ser marcado como enviado (statusenvio = 'Enviado') se a data de pagamento do requerimento estiver preenchido (!NULL). Caso contrário, a tentativa de inserção ou atualização deve ser abortada.

Tabelas envolvidas : BOLETIM, REQUERIMENTO_ANALISE

Momento do disparo : BEFORE INSERT OR UPDATE OF statusenvio em boletim

Lógica:

- Só valida quando statusenvio = 'Enviado'.
- Busca datapagamento no requerimento_analise vinculado.
- Se for NULL, lança erro abortando a operação.

Comando para criar função associada:

```
CREATE OR REPLACE FUNCTION trg_validar_envio_apos_pagamento()
RETURNS TRIGGER AS $$
DECLARE
   v_pagamento DATE;
BEGIN
   IF NEW.statusenvio = 'Enviado' THEN
      SELECT datapagamento
        INTO v_pagamento
      FROM requerimento_analise
      WHERE idrequerimento = NEW.idrequerimento;

      IF v_pagamento IS NULL THEN
         RAISE EXCEPTION
          'Não é possível enviar laudo para requerimento % sem pagamento.',
          NEW.idrequerimento;
      END IF;
   END IF;

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Comando para criar trigger:

```
CREATE TRIGGER validar_envio_apos_pagamento
  BEFORE INSERT OR UPDATE OF statusenvio
  ON boletim
  FOR EACH ROW
  EXECUTE FUNCTION trg_validar_envio_apos_pagamento();
```

Verificando:

Tentativa de envio com sucesso sem pagamento efetuado:

```
SELECT b.idboletim, b.statusenvio, r.datapagamento
FROM boletim b
JOIN requerimento_analise r ON r.idrequerimento = b.idrequerimento
WHERE b.idboletim = 'B002';
```

idboletim	statusenvio	datapagamento
B002	Pendente	*NULL*

```
UPDATE boletim
  SET statusenvio = 'Enviado'
WHERE idboletim = 'B002';
```

```
ERROR:  Não é possível enviar laudo para requerimento R002 sem pagamento.
CONTEXT:  PL/pgSQL function trg_validar_envio_apos_pagamento() line 12 at RAISE
```

13 -

a) Índice primário

Um exemplo de índice primário existente no banco de dados é o atributo **EmailCliente** na tabela **CLIENTE**. Esse índice foi criado quando a tabela **Cliente** foi definida com **EmailCliente** como **PRIMARY KEY** no comando de criação de tabela:

```
CREATE TABLE Cliente (
        EmailCliente VARCHAR(120) PRIMARY KEY,
        NomeCliente VARCHAR(120) NOT NULL,
        contato_responsavel VARCHAR(120),
        Endereco VARCHAR(255),
        TelefoneFax VARCHAR(25)
);
```

O índice está sendo utilizado. Ele é usado para garantir a unicidade de cada endereço de e-mail do cliente e para otimizar as operações de busca e junção que envolvem a coluna **EmailCliente** , como na inserção válida do cliente:

```
INSERT INTO Cliente (
        EmailCliente,
        NomeCliente,
        ContatoResponsavel,
        Endereco,
        TelefoneFax
 ) VALUES (
        'contato@aguasexemplo.com.br',
        'Águas Exemplo Ltda.',
        'Maria souza',
        'Rua das Flores, 123 – Rio de Janeiro/RJ',
        '(21) 3527-1814'
);
```

Além disso, a tentativa de inserção de um cliente com o mesmo e-mail resulta em um erro de violação de chave primária, confirmando o uso do índice para manter a unicidade. Isso é mostrado na inserção abaixo:

```
INSERT INTO Cliente (
        EmailCliente,
        NomeCliente
) VALUES (
        'contato@aguasexemplo.com.br',
        'Outro Cliente'
)
```

A qual retorna o seguinte erro:

```
ERROR:  duplicate key value violates unique constraint "cliente_pkey"
DETAIL:  Key (emailcliente)=(contato@aguasexemplo.com.br) already exists.
```

b) Índice secundário

1. Índice na coluna NomeCliente da tabela CLIENTE:

Criação do índice em SQL:

CREATE INDEX idx_cliente_nomecliente ON Cliente (NomeCliente);

Esse índice vai facilitar casos de **busca frenqunte,** pois, apesar de **EmailCliente** seja a chave primária e o identificador único do cliente, é comum que usuários ou funcionários do laboratório busquem clientes pelo nome (**NomeCliente**). Sem um índice, essas buscas exigiram uma varredura completa da tabela **CLIENTE**, o que seria ineficiente em uma base de dados com muitos clientes. Além disso, consultas que ordenam ou agrupam resultados por **NomeCliente** (por exemplo, "listar clientes em ordem alfabética" ou "contar amostras por cliente") se beneficiam da existência deste índice, acelerando essas operações.

2. Índice na coluna DataColeta da tabela AMOSTRA:

Criação do Índice em SQL:

CREATE INDEX idx_amostra_datacoleta ON Amostra (DataColeta);

Primeiramente**,** o laboratório frequentemente realiza análises para amostras coletadas em um determinado período. Consultas que filtram amostras por
DataColeta (ex: WHERE DataColeta BETWEEN '2025-01-01' AND '2025-06-30') seriam significativamente mais rápidas com este índice, pois ele permitiria ao SGBD localizar diretamente os registros relevantes sem escanear toda a tabela.

3. Forçar o otimizador a usar os índices

Mesmo com um índice criado, o otimizador de consultas do PostgreSQL pode decidir não usá-lo se ele acreditar que uma varredura sequencial da tabela (full table scan) seria mais eficiente. Para incentivar o uso do índice **idx_amostra_datacoleta** em um cenário onde ele seria útil (por exemplo, uma consulta que busca por um pequeno intervalo de datas em uma tabela **AMOSTRA** razoavelmente grande), pode-se tentar as seguintes abordagens:

Forçar o Uso do Índice:

Em PostgreSQL, pode-se temporariamente desabilitar varreduras sequenciais para uma sessão, forçando o otimizador a considerar apenas planos que usam índices (se disponíveis).

SQL:

```
SET enable_seqscan TO OFF;

SELECT

    a.CodigoPUC,

    a.PontoColeta,

    a.DataColeta,

    c.NomeCliente,

    c.contato_responsavel

FROM

    Amostra a

INNER JOIN

    Cliente c ON a.EmailCliente = c.EmailCliente

WHERE

    a.MatrizAmostra = 'Água Bruta'

    AND a.DataColeta = '2025-05-20' -- Usando um critério mais seletivo

ORDER BY

    a.DataColeta ASC;

SET enable_seqscan TO ON; -- Lembre-se de reabilitar
```

Ao desabilitar **enable_seqscan**, estamos informando ao PostgreSQL que a varredura sequencial é "mais cara" do que o normal, o que o empurra a considerar planos baseados em índices. Em cenários reais, se o otimizador ainda preferir a varredura sequencial, geralmente significa que, para a cardinalidade e seletividade dos dados da sua tabela, a varredura sequencial é de fato mais rápida. Isso é mais para diagnóstico e para entender o comportamento do otimizador do que para uma solução permanente. Além disso, outra forma de fazer o otimizador utilizar o índice é garantir que a consulta seja escrita de forma que o otimizador a considere eficiente.