Explicação do Código

Sistema de Gerenciamento de Biblioteca Digital

Raissa Caroline e Gabriel Nunes

DDL · DML · Consultas SQL (RC001–RC010)

Análise de cada decisão de projeto

1. Visão Geral do Projeto

O código está organizado em três scripts SQL independentes, que devem ser executados em sequência. Cada arquivo tem responsabilidade única, seguindo o princípio de separação entre definição de estrutura, carga de dados e consultas.

Arquivo	Comandos SQL	Responsabilidade
DDL.sql	CREATE DATABASE/TABLE	Estrutura, constraints, índices
DML.sql	INSERT INTO	Carga de dados de teste
Consultas.sql	SELECT, CREATE VIEW	Consultas analíticas RC001–RC010

2. DDL.sql - Definição das Estruturas

2.1 Criação do Banco de Dados

O script começa garantindo um estado limpo com DROP DATABASE IF EXISTS, seguido da criação com charset utf8mb4 para suporte completo a Unicode (acentos, caracteres especiais).

```sql
DROP DATABASE IF EXISTS biblioteca_digital;
CREATE DATABASE biblioteca_digital
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;
USE biblioteca_digital;
```

> utf8mb4 é o charset correto para MySQL. O charset utf8 padrão do MySQL é, na verdade, utf8mb3 e não suporta todos os caracteres Unicode (como emojis). utf8mb4 é a escolha profissional.

2.2 Tabela Editora

Primeira tabela criada intencionalmente - ela é referenciada por Livro via FK, portanto precisa existir primeiro para garantir integridade referencial.

```sql
CREATE TABLE Editora (
    id_editora   INT           NOT NULL AUTO_INCREMENT,
    nome         VARCHAR(150)  NOT NULL,
    localizacao  VARCHAR(200),
    CONSTRAINT pk_editora PRIMARY KEY (id_editora)
) ENGINE=InnoDB;
```

- AUTO_INCREMENT: MySQL gera o ID automaticamente a cada INSERT
- NOT NULL em nome: garante que toda editora tenha nome (RN003)
- localizacao sem NOT NULL: campo opcional por não ser crítico para o negócio
- ENGINE=InnoDB: obrigatório para suporte a FOREIGN KEY no MySQL
- CONSTRAINT pk_editora: nomear constraints facilita diagnóstico de erros

2.3 Tabela Livro

O ISBN é usado como chave primária diretamente - não é um INT autoincrement - pois é um identificador universal e único por natureza (RN001).

```sql
CREATE TABLE Livro (
    isbn            VARCHAR(20)   NOT NULL,
    titulo          VARCHAR(300)  NOT NULL,
    ano_publicacao  YEAR,
    edicao          VARCHAR(20),
    id_editora      INT           NOT NULL,
    CONSTRAINT pk_livro PRIMARY KEY (isbn),
    CONSTRAINT fk_livro_editora
        FOREIGN KEY (id_editora) REFERENCES Editora(id_editora)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
) ENGINE=InnoDB;
```

- ISBN como PK: elimina redundância e garante identificação universal (RN001)
- ON UPDATE CASCADE: se o id_editora mudar, a FK em Livro se atualiza automaticamente
- ON DELETE RESTRICT: impede excluir uma editora que ainda tem livros cadastrados (RN002)
- YEAR: tipo nativo do MySQL para armazenar apenas o ano, sem dia/mês

2.4 Tabela Livro_Autor - Relacionamento N:M

Esta tabela implementa o relacionamento N:M entre Livro e Autor (RN004). Não possui atributos próprios - sua chave primária é composta pelos dois atributos FK.

```sql
CREATE TABLE Livro_Autor (
    isbn      VARCHAR(20)  NOT NULL,
    id_autor  INT          NOT NULL,
    CONSTRAINT pk_livro_autor  PRIMARY KEY (isbn, id_autor),
    CONSTRAINT fk_la_livro
        FOREIGN KEY (isbn)     REFERENCES Livro(isbn)
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT fk_la_autor
        FOREIGN KEY (id_autor) REFERENCES Autor(id_autor)
        ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB;
```

- Chave primária composta (isbn, id_autor): garante que cada par livro-autor seja único
- ON DELETE CASCADE no lado do Livro: se um livro for excluído, suas associações com autores são removidas automaticamente
- ON DELETE RESTRICT no lado do Autor: impede excluir um autor que ainda tem livros

2.5 Tabela Exemplar - Rastreamento Físico

A entidade Exemplar implementa o requisito RN103: separar o conceito de 'título' (Livro) do 'item físico' (Exemplar). Empréstimos e reservas são feitos no nível do Exemplar.

```SQL
CREATE TABLE Exemplar (
    id_exemplar        INT          NOT NULL AUTO_INCREMENT,
    isbn               VARCHAR(20)  NOT NULL,
    numero_tombamento  VARCHAR(50),
    status             ENUM('Disponivel','Emprestado','Reservado','Extraviado')
                       NOT NULL DEFAULT 'Disponivel',
    CONSTRAINT pk_exemplar PRIMARY KEY (id_exemplar),
    CONSTRAINT fk_exemplar_livro
        FOREIGN KEY (isbn) REFERENCES Livro(isbn)
        ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB;
```

- ENUM: restringe o campo status a apenas os valores válidos — mais seguro que VARCHAR livre
- DEFAULT 'Disponivel': todo exemplar recém-cadastrado começa disponível automaticamente
- número_tombamento: código de patrimônio físico da biblioteca, opcional

2.6 Tabela Emprestimo - Controle de Prazos

Esta tabela é o coração transacional do sistema. A coluna data_real_devolucao com valor NULL representa o estado 'ainda não devolvido', tornando qualquer consulta de empréstimos ativos direta e eficiente.

```SQL
CREATE TABLE Emprestimo (
    id_emprestimo            INT   NOT NULL AUTO_INCREMENT,
    id_exemplar              INT   NOT NULL,
    id_usuario               INT   NOT NULL,
    data_emprestimo          DATE  NOT NULL,
    data_prevista_devolucao  DATE  NOT NULL,
    data_real_devolucao      DATE  NULL DEFAULT NULL,
    CONSTRAINT pk_emprestimo PRIMARY KEY (id_emprestimo),
    CONSTRAINT fk_emp_exemplar
        FOREIGN KEY (id_exemplar) REFERENCES Exemplar(id_exemplar)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT fk_emp_usuario
        FOREIGN KEY (id_usuario)  REFERENCES Usuario(id_usuario)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT chk_datas_emprestimo
        CHECK (data_prevista_devolucao > data_emprestimo)
```

```
) ENGINE=InnoDB;
```

- NULL DEFAULT NULL em data_real_devolucao: NULL significa 'não devolvido'. Quando o livro é devolvido (RF004), este campo é preenchido com a data real
- CHECK constraint (RN006): o banco recusa qualquer INSERT ou UPDATE onde a data prevista seja anterior ou igual à data do empréstimo
- FK para Exemplar (não para Livro): permite rastrear qual exemplar físico foi emprestado, não apenas qual título

2.7 Tabela Reserva - Fila FIFO

A entidade Reserva implementa RN101 (entidade de reserva) e RN102 (fila FIFO). A prioridade FIFO não é implementada por uma coluna de ordem, mas sim pelo campo data_reserva com tipo DATETIME, permitindo ordenação cronológica natural.

```
SQL
CREATE TABLE Reserva (
    id_reserva    INT       NOT NULL AUTO_INCREMENT,
    id_exemplar   INT       NOT NULL,
    id_usuario    INT       NOT NULL,
    data_reserva  DATETIME  NOT NULL DEFAULT CURRENT_TIMESTAMP,
    status        ENUM('Ativa','Cancelada','Atendida') NOT NULL DEFAULT
'Ativa',
    ...
) ENGINE=InnoDB;
```

- DATETIME (não DATE): armazena hora, minuto e segundo — permite desempate em reservas feitas no mesmo dia
- DEFAULT CURRENT_TIMESTAMP: captura automaticamente o momento exato da reserva
- FIFO implementado nas queries: ORDER BY data_reserva ASC LIMIT 1 (RC010)

2.8 Índices Auxiliares

Índices são estruturas de acesso que aceleram consultas às custas de espaço em disco. Foram criados nos campos mais consultados:

```
SQL
CREATE INDEX idx_emprestimo_usuario   ON Emprestimo(id_usuario);
CREATE INDEX idx_emprestimo_exemplar  ON Emprestimo(id_exemplar);
CREATE INDEX idx_emprestimo_datas     ON Emprestimo(data_prevista_devolucao,
data_real_devolucao);
CREATE INDEX idx_reserva_status       ON Reserva(status, data_reserva);
CREATE INDEX idx_exemplar_status      ON Exemplar(status);
```

- idx_emprestimo_datas: índice composto para RC001 (filtra por duas colunas de data simultaneamente)

- idx_reserva_status: índice composto para RC009/RC010 (filtra por status='Ativa' e ordena por data)
- FKs já criam índices automaticamente no InnoDB; estes índices cobrem as colunas restantes mais consultadas

3. DML.sql - Carga de Dados

3.1 Estratégia dos Dados de Teste

Os dados não foram inseridos aleatoriamente. Cada registro foi planejado para garantir que todas as 10 consultas retornem resultados significativos e verificáveis.

Dado de Teste	Cenário Coberto	Query Beneficiada
Empréstimos com data_real_devolucao = NULL e data vencida	Atrasos detectáveis	
Múltiplos exemplares do mesmo ISBN	Disponibilidade parcial	
Livros com 2 e 4 autores na Livro_Autor	N:M funcional	
Reservas com datas diferentes para o mesmo exemplar	Fila FIFO com desempate	
Multas com pago=0 e pago=1	SUM condicional RC008	

3.2 Inserção em Ordem de Dependência

A ordem dos INSERTs é crítica e respeita as dependências de chaves estrangeiras: primeiro as tabelas sem FKs, depois as que referenciam outras.

```sql
-- Ordem obrigatória de inserção:
-- 1. Editora        (sem FKs)
-- 2. Livro          (FK → Editora)
-- 3. Autor          (sem FKs)
-- 4. Livro_Autor    (FK → Livro, FK → Autor)
-- 5. Exemplar       (FK → Livro)
-- 6. Usuario        (sem FKs)
-- 7. Emprestimo     (FK → Exemplar, FK → Usuario)
-- 8. Reserva        (FK → Exemplar, FK → Usuario)
-- 9. Multa          (FK → Emprestimo)
```

> Inverter esta ordem causaria erro de Foreign Key Constraint (errno 150). O MySQL rejeita inserir um registro que referencia uma chave que ainda não existe.

4. Consultas.sql - Análise Detalhada

RC001 - Livros em Atraso

Técnicas utilizadas: INNER JOIN em cadeia, função DATEDIFF, filtros IS NULL e comparação de datas.

```sql
SELECT
    u.nome                                      AS nome_usuario,
    l.titulo                                    AS titulo_livro,
    ex.numero_tombamento,
    DATEDIFF(CURDATE(), e.data_prevista_devolucao) AS dias_atraso
FROM Emprestimo e
    INNER JOIN Usuario  u  ON e.id_usuario  = u.id_usuario
    INNER JOIN Exemplar ex ON e.id_exemplar = ex.id_exemplar
    INNER JOIN Livro    l  ON ex.isbn       = l.isbn
WHERE
    e.data_real_devolucao IS NULL         -- ainda não devolvido
    AND e.data_prevista_devolucao < CURDATE()  -- prazo vencido
ORDER BY dias_atraso DESC;
```

- Emprestimo → Exemplar → Livro: como os empréstimos são feitos no nível do Exemplar (RN103), precisamos percorrer dois JOINs para chegar ao título
- DATEDIFF(CURDATE(), data_prevista): calcula quantos dias se passaram desde o vencimento. Valor positivo = em atraso
- IS NULL: em SQL, comparações com NULL exigem IS NULL (não = NULL)
- ORDER BY dias_atraso DESC: coloca o caso mais grave no topo

RC002 - Análise de Acervo

Técnicas: GROUP BY com COUNT, LEFT JOIN para incluir editoras sem livros, alias de coluna, ORDER BY DESC.

```sql
SELECT
    ed.nome          AS nome_editora,
    COUNT(l.isbn)    AS total_livros
FROM Editora ed
    LEFT JOIN Livro l ON ed.id_editora = l.id_editora
GROUP BY ed.id_editora, ed.nome
ORDER BY total_livros DESC;
```

- LEFT JOIN: incluiria editoras sem nenhum livro no acervo (COUNT retornaria 0). INNER JOIN as excluiria

- GROUP BY ed.id_editora, ed.nome: agrupa por ID (mais eficiente que por texto) e inclui nome na projeção
- COUNT(l.isbn): conta apenas linhas não-NULL do Livro. Se for COUNT(*), contaria a linha mesmo sem livros

RC003 - Disponibilidade

Duas abordagens para verificar ausência de registros. Ambas produzem o mesmo resultado; cada uma tem vantagens em cenários específicos.

```sql
-- Versão 1: NOT IN com subquery
SELECT DISTINCT l.isbn, l.titulo FROM Livro l
    INNER JOIN Exemplar ex ON l.isbn = ex.isbn
WHERE
    ex.id_exemplar NOT IN (
        SELECT id_exemplar FROM Emprestimo
        WHERE data_real_devolucao IS NULL
    )
    AND ex.status = 'Disponivel';

-- Versão 2: NOT EXISTS (geralmente mais eficiente)
SELECT DISTINCT l.isbn, l.titulo FROM Livro l
    INNER JOIN Exemplar ex ON l.isbn = ex.isbn
WHERE NOT EXISTS (
    SELECT 1 FROM Emprestimo e
    WHERE e.id_exemplar = ex.id_exemplar
      AND e.data_real_devolucao IS NULL
);
```

- NOT IN vs NOT EXISTS: NOT EXISTS é geralmente mais eficiente pois para ao encontrar o primeiro match. NOT IN carrega toda a subquery em memória
- NOT IN tem armadilha com NULL: se a subquery retornar qualquer NULL, NOT IN retorna nenhuma linha. NOT EXISTS não tem este problema
- SELECT 1 no NOT EXISTS: o valor 1 é irrelevante; o banco só verifica se a linha existe (true/false)
- DISTINCT: evita linhas duplicadas quando um título tem múltiplos exemplares disponíveis

RC005 - View V_EmprestimosAtivos

Uma VIEW é uma consulta nomeada e armazenada. Funciona como uma tabela virtual — pode ser consultada com SELECT, filtrada com WHERE e unida com JOIN. Não armazena dados duplicados.

```sql
CREATE VIEW V_EmprestimosAtivos AS
SELECT
    e.id_emprestimo,
    u.nome                                          AS nome_usuario,
    l.titulo                                        AS titulo_livro,
    e.data_emprestimo,
    e.data_prevista_devolucao,
    DATEDIFF(CURDATE(), e.data_prevista_devolucao)  AS dias_atraso,
```

```
    CASE
        WHEN e.data_prevista_devolucao < CURDATE() THEN 'Em Atraso'
        ELSE 'No Prazo'
    END                                             AS situacao
FROM Emprestimo e
    INNER JOIN Usuario  u  ON e.id_usuario  = u.id_usuario
    INNER JOIN Exemplar ex ON e.id_exemplar = ex.id_exemplar
    INNER JOIN Livro    l  ON ex.isbn       = l.isbn
WHERE e.data_real_devolucao IS NULL;
```

- CASE WHEN: expressão condicional SQL, similar ao if/else de outras linguagens
- dias_atraso calculado dinamicamente: a VIEW não armazena este valor — ele é recalculado cada vez que a VIEW é consultada, usando CURDATE() atual
- WHERE na definição da VIEW: filtra apenas empréstimos não devolvidos. Quem consultar a VIEW não precisa repetir este filtro

RC006 - Álgebra Relacional

Tradução da expressão formal: PiNome,Titulo(sigmaAno>2020(Usuario ⋈ Emprestimo ⋈ Livro))

```
SQL
-- Álgebra: Pi_Nome,Titulo( sigma_Ano>2020( Usuario ⋈ Emprestimo ⋈ Livro ) )
-- sigma = seleção (WHERE)
-- Pi = projeção (SELECT colunas)
-- ⋈ = junção natural (INNER JOIN)

SELECT DISTINCT
    u.nome   AS nome_usuario,   -- Pi: projetar Nome
    l.titulo AS titulo_livro    -- Pi: projetar Titulo
FROM Usuario u
    INNER JOIN Emprestimo e  ON u.id_usuario  = e.id_usuario   -- ⋈
    INNER JOIN Exemplar   ex ON e.id_exemplar = ex.id_exemplar -- ⋈
    INNER JOIN Livro      l  ON ex.isbn       = l.isbn         -- ⋈
WHERE l.ano_publicacao > 2020   -- sigma: seleção
ORDER BY u.nome, l.titulo;
```

- sigma (seleção) → WHERE: filtra as tuplas que satisfazem a condição
- Pi (projeção) → SELECT: mantém apenas as colunas especificadas (Nome, Titulo)
- ⋈ (junção natural) → INNER JOIN: combina tuplas com valores iguais nos atributos comuns
- Exemplar como tabela intermediária: necessário porque o modelo separa Livro de Exemplar (RN103)
- DISTINCT: a projeção em álgebra relacional remove duplicatas por definição; DISTINCT garante este comportamento

RC007 - Ranking com Janela Temporal Dinâmica

```
SQL
SELECT
    l.isbn,
```

```
    l.titulo                  AS titulo_livro,
    COUNT(e.id_emprestimo)    AS total_emprestimos_6meses
FROM Emprestimo e
    INNER JOIN Exemplar ex ON e.id_exemplar = ex.id_exemplar
    INNER JOIN Livro    l  ON ex.isbn       = l.isbn
WHERE
    e.data_emprestimo >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY l.isbn, l.titulo
ORDER BY total_emprestimos_6meses DESC
LIMIT 10;
```

- DATE_SUB(CURDATE(), INTERVAL 6 MONTH): função que subtrai 6 meses da data atual. Janela temporal completamente dinâmica
- LIMIT 10: retorna apenas os 10 livros mais populares (ranking)
- Resultado vazio nos dados de teste: esperado, pois os dados simulam 2024/2025 e a execução ocorreu em fev/2026

RC008 - SUM Condicional

```
SQL
SELECT
    u.nome                          AS nome_usuario,
    SUM(m.valor)                    AS total_multas_acumuladas,
    SUM(CASE WHEN m.pago = 1
             THEN m.valor ELSE 0
        END)                        AS multas_pagas,
    SUM(CASE WHEN m.pago = 0
             THEN m.valor ELSE 0
        END)                        AS multas_pendentes
FROM Multa m
    INNER JOIN Emprestimo e ON m.id_emprestimo = e.id_emprestimo
    INNER JOIN Usuario    u ON e.id_usuario    = u.id_usuario
GROUP BY u.id_usuario, u.nome
ORDER BY total_multas_acumuladas DESC;
```

- SUM(CASE WHEN): padrão 'pivot' em SQL — soma apenas os valores que satisfazem uma condição, zerando os demais
- Três SUMs em uma query: total, pagas e pendentes são calculados em uma única passagem pela tabela (eficiente)
- JOIN triplo: Multa → Emprestimo → Usuario (percurso de 2 FKs para chegar ao nome do usuário)

RC009 e RC010 - Reservas e FIFO

```
SQL
-- RC009: Quantidade na fila por título
SELECT l.titulo, COUNT(r.id_reserva) AS quantidade_na_fila
FROM Reserva r
    INNER JOIN Exemplar ex ON r.id_exemplar = ex.id_exemplar
    INNER JOIN Livro    l  ON ex.isbn       = l.isbn
WHERE r.status = 'Ativa'
```

```
GROUP BY l.isbn, l.titulo
ORDER BY quantidade_na_fila DESC;

-- RC010: Primeiro da fila para um ISBN (FIFO)
SELECT u.nome, r.data_reserva
FROM Reserva r
    INNER JOIN Usuario  u  ON r.id_usuario  = u.id_usuario
    INNER JOIN Exemplar ex ON r.id_exemplar = ex.id_exemplar
    INNER JOIN Livro    l  ON ex.isbn       = l.isbn
WHERE ex.isbn = '978-8543004792'
  AND r.status = 'Ativa'
ORDER BY r.data_reserva ASC
LIMIT 1;
```

- RC009 usa COUNT + GROUP BY: agrupa por título e conta as reservas ativas
- RC010 implementa FIFO: ORDER BY data_reserva ASC garante que a reserva mais antiga venha primeiro. LIMIT 1 retorna apenas o topo da fila
- data_reserva como DATETIME: captura hora e minuto, permitindo desempatar reservas feitas no mesmo dia

5. Boas Práticas Aplicadas no Código

Prática	Onde/Como Aplicada
Nomenclatura de constraints	Todas as PKs, FKs e CHECKs têm nomes descritivos (ex: fk_livro_editora)
Aliases em queries	Colunas renomeadas com AS para clareza (AS nome_usuario)
Comentários nos scripts	Cada seção tem comentários explicativos com os IDs dos requisitos
ON DELETE RESTRICT	Impede exclusões que quebrariam integridade referencial
ENUM para domínios fixos	Mais seguro que VARCHAR livre; o banco rejeita valores inválidos
DROP IF EXISTS antes do CREATE	Garante execução idempotente do script (pode rodar múltiplas vezes)
Índices compostos	Cobrem os dois filtros mais comuns das queries (status + data)
IS NULL para ausência de valor	Correto para verificar campos opcionais como data_real_devolucao

6. Resumo: O que cada arquivo entrega

> DDL.sql — Define a estrutura. Roda uma vez. Cria as 9 tabelas com todas as constraints e índices. Se rodar novamente, recria tudo do zero (DROP DATABASE IF EXISTS).

DML.sql — Popula o banco. Roda após o DDL. Insere os dados de teste em ordem de dependência. Projetado para que todas as 10 consultas retornem resultados não-triviais.

Consultas.sql — Valida o sistema. Roda após o DML. Cada bloco é independente e pode ser executado individualmente com Cmd+Enter no Workbench. RC005 cria a VIEW antes de consultá-la.