Banco de Dados

Profª. Ana Luiza Scharf
ana.scharf@ifsc.edu.br

LABORATÓRIO: Integração SQL-JPA - Parte 04

Objetivo da Parte 04

Esta etapa visa demonstrar a integração entre os scripts SQL (`schema.sql` e `data.sql`) e as entidades JPA, estabelecendo as regras e boas práticas para garantir a comunicação eficiente entre o banco de dados relacional e a camada de objetos do Java.

O Ciclo de Inicialização do Spring Boot

Ciclo de Inicialização Spring Boot

1. Aplicação Spring Boot inicia
2. Executa `schema.sql` (cria tabelas)
3. Executa `data.sql` (popula dados)
4. Hibernate valida entidades vs tabelas
5. Executa `main()` da aplicação
6. Roda `TesteComponent.rodarTestes()`

Figura 1: Ciclo de inicialização da aplicação Spring Boot

Configuração Essencial no `application.properties`

```
# ================================================
# 1. CONEXÃO COM O BANCO DE DADOS
# ================================================
spring.datasource.url=jdbc:mysql://localhost:3306/bcd
spring.datasource.username=aluno
spring.datasource.password=aluno
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# ================================================
# 2. CONTROLE DO HIBERNATE (DDL)
# ================================================
# 'none' = Desliga criação automática de tabelas
#          (usamos schema.sql manualmente)
spring.jpa.hibernate.ddl-auto=none

# ================================================
# 3. ATIVAÇÃO DOS SCRIPTS SQL
# ================================================
# 'always' = Sempre executa schema.sql e data.sql
spring.sql.init.mode=always

# ================================================
# 4. CONFIGURAÇÕES ADICIONAIS (OPCIONAIS)
# ================================================
# Mostra SQL gerado pelo Hibernate (para debug)
spring.jpa.show-sql=true

# Formata o SQL exibido no console
spring.jpa.properties.hibernate.format_sql=true
```

Estrutura dos Arquivos SQL

`schema.sql` - Criação das Tabelas

Este arquivo deve criar as tabelas **exatamente** como as entidades JPA esperam.

Regra Fundamental: Nomeação

Os nomes das tabelas e colunas no `schema.sql` devem corresponder aos nomes mapeados pelas anotações JPA (`@Table`, `@Column`) ou seguir o padrão de nomenclatura do Hibernate.

```sql
-- ================================================
-- schema.sql - Cria tabelas MANUALMENTE
-- ================================================

-- Limpa tabelas existentes (importante para testes)
DROP TABLE IF EXISTS matricula;
DROP TABLE IF EXISTS disciplina_professor;
DROP TABLE IF EXISTS professor;
DROP TABLE IF EXISTS disciplina;
DROP TABLE IF EXISTS aluno;

-- 1. Tabela ALUNO (corresponde à entidade Aluno)
CREATE TABLE aluno (
    matricula BIGINT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(255) NOT NULL,
    matricula_ativa BOOLEAN NOT NULL,
    data_inicio DATE NOT NULL
);

-- 2. Tabela DISCIPLINA (corresponde à entidade Disciplina)
CREATE TABLE disciplina (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(255) NOT NULL,
    creditos INT NOT NULL
);

-- 3. Tabela PROFESSOR (corresponde à entidade Professor)
CREATE TABLE professor (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(255) NOT NULL,
    area VARCHAR(255) NOT NULL
);

-- 4. Tabela de junção N:M SIMPLES (gerada pelo @ManyToMany)
CREATE TABLE disciplina_professor (
    disciplina_id BIGINT NOT NULL,
    professor_id BIGINT NOT NULL,
    PRIMARY KEY (disciplina_id, professor_id),
    FOREIGN KEY (disciplina_id) REFERENCES disciplina(id),
    FOREIGN KEY (professor_id) REFERENCES professor(id)
);

-- 5. Tabela MATRICULA (entidade de junção explícita)
CREATE TABLE matricula (
    aluno_id BIGINT NOT NULL,          -- Nome da coluna deve bater com @JoinColumn
    disciplina_id BIGINT NOT NULL,     -- Nome da coluna deve bater com @JoinColumn
    nota DOUBLE,
    data_matricula DATE NOT NULL,
    PRIMARY KEY (aluno_id, disciplina_id),
    FOREIGN KEY (aluno_id) REFERENCES aluno(matricula),
    FOREIGN KEY (disciplina_id) REFERENCES disciplina(id)
);
```

`data.sql` - População Inicial

Este arquivo insere os dados iniciais. Atenção especial aos **nomes das colunas** e **tipos de dados**.

```sql
-- ============================================
-- data.sql - Dados iniciais para testes
-- ============================================

-- 1. INSERIR ALUNOS
-- Nota: Não especificar o ID (deixe AUTO_INCREMENT)
INSERT INTO aluno (nome, matricula_ativa, data_inicio) VALUES
('Mariana Luz', true, '2025-01-01'),
('Joao Silva', false, '2024-08-15');

-- 2. INSERIR DISCIPLINAS
INSERT INTO disciplina (nome, creditos) VALUES
('Programacao Orientada a Objetos', 4),
('Banco de Dados', 4);

-- 3. INSERIR PROFESSORES
INSERT INTO professor (nome, area) VALUES
('Dr. Pedro Alvares', 'Programacao'),
('Dra. Maria Santos', 'Banco de Dados');

-- 4. ASSOCIAR PROFESSORES A DISCIPLINAS (N:M Simples)
-- Nota: Usamos subconsultas para obter IDs dinamicamente
INSERT INTO disciplina_professor (disciplina_id, professor_id)
SELECT d.id, p.id
FROM disciplina d, professor p
WHERE d.nome = 'Programacao Orientada a Objetos'
  AND p.nome = 'Dr. Pedro Alvares';

INSERT INTO disciplina_professor (disciplina_id, professor_id)
SELECT d.id, p.id
FROM disciplina d, professor p
WHERE d.nome = 'Banco de Dados'
  AND p.nome = 'Dra. Maria Santos';

-- 5. INSERIR MATRICULAS (N:M com atributo)
-- Importante: Usar os nomes exatos das colunas FK
INSERT INTO matricula (aluno_id, disciplina_id, nota, data_matricula)
SELECT a.matricula, d.id, 9.8, '2024-03-01'
FROM aluno a, disciplina d
WHERE a.nome = 'Mariana Luz'
  AND d.nome = 'Programacao Orientada a Objetos';

INSERT INTO matricula (aluno_id, disciplina_id, nota, data_matricula)
SELECT a.matricula, d.id, 8.5, '2024-03-01'
FROM aluno a, disciplina d
WHERE a.nome = 'Mariana Luz'
  AND d.nome = 'Banco de Dados';
```

Regras de Compatibilidade entre SQL e JPA

Regra 1: Correspondência de Nomes - O Mapa Entre Java e SQL

Esta é a regra mais importante. Se os nomes não baterem, o Hibernate não consegue fazer o mapeamento.

ATENÇÃO: Case Sensitivity no MySQL vs Java

MySQL no Windows/Mac: Não diferencia maiúsculas/minúsculas
MySQL no Linux: Diferencia maiúsculas/minúsculas
Java: Sempre diferencia maiúsculas/minúsculas
Solução: Use sempre snake_case (tudo minúsculo com underscores)

5.1.1 Como o Hibernate Converte Nomes Automaticamente

Elemento Java	Converte para	Exemplo no SQL
Nome da Classe	snake_case (minúsculo)	`Aluno → aluno`
Atributo camelCase	snake_case	`matriculaAtiva → matricula_ativa`
Atributo simples	mantém minúsculo	`nome → nome`
Relacionamento @ManyToOne	nome do campo + "_id"	`aluno → aluno_id`
Tabela @ManyToMany	tabela1_tabela2	`disciplina_professor`

```java
// JAVA: Entidade Aluno
@Entity  // Hibernate espera tabela "aluno"
public class Aluno {
    @Id
    private Long matricula;          // Espera coluna "matricula"

    private String nome;             // Espera coluna "nome"
    private Boolean matriculaAtiva;  // Espera "matricula_ativa"
    private LocalDate dataInicio;    // Espera "data_inicio"
}

// SQL CORRESPONDENTE OBRIGATÓRIO:
CREATE TABLE aluno (
    matricula BIGINT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(255) NOT NULL,
    matricula_ativa BOOLEAN NOT NULL,  -- Note o snake_case!
    data_inicio DATE NOT NULL          -- Note o snake_case!
);
```

5.1.2 Controle Manual com Anotações

Você pode forçar nomes específicos com `@Table` e `@Column`:

```java
@Entity
@Table(name = "tb_alunos")  // Força nome da tabela
public class Aluno {

    @Id
    @GeneratedValue
    @Column(name = "id_aluno")  // Força nome da coluna
    private Long matricula;

    @Column(name = "nome_completo", length = 100)
    private String nome;

    @Column(name = "ativo")  // Nome diferente do atributo
    private Boolean matriculaAtiva;

    @Column(name = "dt_inicio")  // Abreviação comum
    private LocalDate dataInicio;
}

// SQL OBRIGATÓRIO com os nomes forçados:
CREATE TABLE tb_alunos (
    id_aluno BIGINT PRIMARY KEY AUTO_INCREMENT,
    nome_completo VARCHAR(100) NOT NULL,
    ativo BOOLEAN NOT NULL,
    dt_inicio DATE NOT NULL
);
```

5.1.3 Mapeamento de Chaves Estrangeiras

Como as relações se tornam FOREIGN KEY no banco:

```java
// JAVA: Relacionamento em Matricula.java
@Entity
```

```java
public class Matricula {

    @ManyToOne
    @JoinColumn(name = "aluno_id")  // Nome da coluna FK
    private Aluno aluno;

    @ManyToOne
    @JoinColumn(name = "disciplina_id")
    private Disciplina disciplina;
}

// SQL CORRESPONDENTE OBRIGATÓRIO:
CREATE TABLE matricula (
    aluno_id BIGINT NOT NULL,        // Nome DEVE ser aluno_id
    disciplina_id BIGINT NOT NULL,   // Nome DEVE ser disciplina_id
    ...
    FOREIGN KEY (aluno_id) REFERENCES aluno(matricula),
    FOREIGN KEY (disciplina_id) REFERENCES disciplina(id)
);
```

ERRO COMUM: Nomes diferentes

Java: `@JoinColumn(name = "aluno_id")`
SQL: `id_aluno BIGINT` ← **ERRO! Nomes diferentes**
Resultado: O Hibernate não encontra a coluna!
Solução: SQL deve usar `aluno_id BIGINT`

5.1.4 Tabelas de Junção N:M

Como @ManyToMany cria tabelas intermediárias:

```java
// JAVA: Relacionamento ManyToMany em Disciplina
@Entity
public class Disciplina {

    @ManyToMany
    @JoinTable(
        name = "disciplina_professor",           // Nome da tabela
        joinColumns = @JoinColumn(name = "disc_id"),      // FK desta entidade
        inverseJoinColumns = @JoinColumn(name = "prof_id") // FK da outra
    )
    private Set<Professor> professores;
}

// SQL que o Hibernate ESPERA encontrar:
CREATE TABLE disciplina_professor (
    disc_id BIGINT NOT NULL,      // Nome DEVE ser disc_id
    prof_id BIGINT NOT NULL,      // Nome DEVE ser prof_id
    PRIMARY KEY (disc_id, prof_id),
    FOREIGN KEY (disc_id) REFERENCES disciplina(id),
    FOREIGN KEY (prof_id) REFERENCES professor(id)
);
```

Regra 2: Tipos de Dados Compatíveis - A Tradução de Tipos

Cada tipo Java precisa de um tipo SQL equivalente.

5.2.1 2.1 Mapeamento Básico de Tipos

Tipo Java	Tipo MySQL	Observações Importantes
`Long`	`BIGINT AUTO_INCREMENT`	Para chaves primárias
`long` (primitivo)	`BIGINT NOT NULL`	Não pode ser nulo
`Integer`	`INT`	Pode ser NULL
`int` (primitivo)	`INT NOT NULL`	Não pode ser nulo
`String`	`VARCHAR(255)`	Tamanho padrão 255
`String`	`VARCHAR(500)`	Com `@Column(length=500)`
`Boolean`	`BOOLEAN` ou `TINYINT(1)`	`true=1, false=0`
`boolean` (primitivo)	`BOOLEAN NOT NULL`	Sempre NOT NULL
`Double`	`DOUBLE`	Ponto flutuante
`double` (primitivo)	`DOUBLE NOT NULL`	Não pode ser nulo
`LocalDate`	`DATE`	Sem hora: '2024-03-15'
`Date`	`DATE`	Com `@Temporal(DATE)`

Regra 3: Chaves Primárias e Estrangeiras

Conceito JPA	Anotação	Equivalente SQL
Chave Primária Simples	`@Id`	`PRIMARY KEY`
Chave Primária Composta	`@EmbeddedId`	`PRIMARY KEY (campo1, campo2)`
Auto Increment	`@GeneratedValue`	`AUTO_INCREMENT`
Chave Estrangeira	`@ManyToOne @JoinColumn`	`FOREIGN KEY REFERENCES`
Chave Única	`@Column(unique=true)`	`UNIQUE KEY`

Checklist de Verificação

CHECKLIST: Verifique sempre antes de rodar

Nomes:
- Tabela no SQL = Nome da classe (em snake_case)
- Colunas no SQL = Atributos (camelCase → snake_case)
- FKs no SQL = Nomes em @JoinColumn

Tipos:
- Java Long = SQL BIGINT
- Java String = SQL VARCHAR
- Java Boolean = SQL BOOLEAN
- Java LocalDate = SQL DATE

Constraints:
- `@Id` = `PRIMARY KEY`
- `nullable=false` = `NOT NULL`
- `@ManyToOne` = `FOREIGN KEY`

Estrutura do Projeto e Ordem de Execução

Estrutura de Pastas

```
src/main/java/ifsc/eng/Lab_Ana_BCD/
 entidades/
    Aluno.java
    Disciplina.java
```

```
    Professor.java
    Matricula.java
    MatriculaId.java
 repositories/
    AlunoRepository.java
    DisciplinaRepository.java
    ProfessorRepository.java
    MatriculaRepository.java
 LabAnaBcdApplication.java
 TesteComponent.java

src/main/resources/
 application.properties    # Configurações
 schema.sql               # Executado PRIMEIRO
 data.sql                 # Executado DEPOIS
```

Ordem de Execução no Spring Boot

1. **Início:** Aplicação Spring Boot inicia
2. **Detecta:** Spring vê `spring.sql.init.mode=always`
3. **Executa:** `schema.sql` → cria todas as tabelas
4. **Executa:** `data.sql` → insere dados iniciais
5. **Valida:** Hibernate compara entidades com tabelas
6. **Roda:** `LabAnaBcdApplication.main()`
7. **Testa:** `TesteComponent.rodarTestes()`

Resumo da Integração

- `schema.sql`: Cria estrutura física do banco
- `data.sql`: Popula com dados iniciais
- **Entidades JPA**: Representam estrutura lógica
- **Spring Boot**: Orquestra execução na ordem correta
- **Compatibilidade**: Nomes e tipos devem ser equivalentes
- **Testes**: Verificam dados SQL e operações JPA

Conclusão

A integração entre SQL scripts e entidades JPA permite:

- Controle preciso da estrutura do banco
- Carga inicial de dados para testes
- Desacoplamento entre DDL e código Java
- Portabilidade entre diferentes bancos
- Manutenção simplificada do esquema

Esta abordagem é ideal para ambientes educacionais onde queremos que os alunos entendam tanto o SQL quanto o JPA.