Ronivaldo Domingues de Andrade

Fundamentos do MySQL e Projetos de Bancos de Dados

Rio de Janeiro - RJ
2026

Ronivaldo Domingues de Andrade

Fundamentos do MySQL e Projetos de Bancos de Dados

Guia prático da capacitação de MySQL e SQL, com duração de duas horas, elaborado para os membros da liga acadêmica.

Rio de Janeiro - RJ
2026

Lista de tabelas

Lista de abreviaturas e siglas

SGBD	Sistema de Gerenciamento de Banco de Dados
DBMS	*Database Management System* (o mesmo que SGBD, em inglês)
RDBMS	*Relational Database Management System*, SGBD relacional
SQL	*Structured Query Language*, linguagem de consulta estruturada
DDL	*Data Definition Language*, subconjunto do SQL que define estruturas
DML	*Data Manipulation Language*, subconjunto do SQL que manipula dados
DQL	*Data Query Language*, subconjunto do SQL que consulta dados
DCL	*Data Control Language*, subconjunto do SQL que controla permissões
TCL	*Transaction Control Language*, subconjunto do SQL que controla transações
CRUD	*Create, Read, Update, Delete*, as quatro operações básicas sobre dados
PK	*Primary Key*, chave primária
FK	*Foreign Key*, chave estrangeira
InnoDB	Motor de armazenamento padrão do MySQL, com suporte a chaves estrangeiras
LTS	*Long Term Support*, versão com suporte de longo prazo
ORM	*Object-Relational Mapping*, mapeamento objeto-relacional

Sumário

1 Introdução

Todo sistema que você vai construir na liga — um cadastro de membros, um aplicativo de projetos, um painel de pesquisa — precisa **guardar dados e recuperá-los depois**. Esta capacitação é sobre a ferramenta padrão da indústria para isso: um banco de dados relacional (MySQL) operado pela linguagem SQL.

Em duas horas não é possível formar um administrador de banco de dados. É possível, sim, sair daqui **criando uma tabela, inserindo dados, consultando com filtro e relacionando duas tabelas** — exatamente o mínimo necessário para o primeiro projeto real.

1.1 Objetivos de aprendizagem

Ao final da capacitação, o participante deve ser capaz de:

1. Explicar o que é um SGBD e o que significa "relacional".
2. Conectar-se a um servidor MySQL usando um cliente SQL.
3. Criar um banco e uma tabela com tipos, chave primária e restrições adequadas.
4. Executar as quatro operações do CRUD com `INSERT`, `SELECT`, `UPDATE` e `DELETE`.
5. Filtrar e ordenar resultados com `WHERE`, `ORDER BY` e `LIMIT`.
6. Relacionar duas tabelas com chave estrangeira e consultá-las com `JOIN`.

1.2 O que não entra nesta capacitação

Dito explicitamente para não gerar expectativa errada — cada item destes é uma capacitação inteira por si só:

- Docker e containers (usados aqui apenas como meio de subir o ambiente).
- Normalização formal (1FN, 2FN, 3FN) e modelagem entidade-relacionamento completa.
- Índices, planos de execução e otimização de desempenho.

- Transações, *views*, *procedures*, *triggers*, backup e replicação.
- Segurança, gestão de usuários e privilégios além do básico.
- Bancos não relacionais (MongoDB, Redis e afins).

1.3 Roteiro de duas horas

A Tabela 1 é o contrato de tempo da capacitação. Cada bloco tem um tempo fechado. Se um bloco estourar, o corte sai do conteúdo marcado como bônus (Seção 6.5), nunca do bloco de `JOIN`.

Tabela 1 – Distribuição dos 120 minutos da capacitação

#	Bloco	Início	Duração
0	Abertura, objetivos e escopo	00:00	5 min
1	Conceitos-base: dado, banco, SGBD, relacional, SQL	00:05	15 min
2	Ambiente e cliente SQL: todos conectados	00:20	15 min
3	DDL: banco, tipos, tabela e chave primária	00:35	20 min
–	*Pausa*	00:55	5 min
4	`INSERT` e `SELECT` com filtros	01:00	25 min
5	`UPDATE`, `DELETE` e o `WHERE` ausente	01:25	10 min
6	Chave estrangeira e `JOIN`	01:35	20 min
7	Erros comuns, próximos passos e dúvidas	01:55	5 min
Total			**120 min**

Regra prática do instrutor

Os blocos 3 a 6 são de prática. Fale no máximo um terço do tempo do bloco e deixe dois terços para os participantes digitarem. Não avance enquanto a maioria não tiver o resultado na tela.

1.4 Conceitos-base

1.4.1 Dado, informação e banco de dados

Dado é um valor bruto: `"Ana"`, `2024-03-11`, `7`. Sozinho, ele não diz nada. **Informação** é o dado dentro de um contexto: *"Ana ingressou na liga em 11/03/2024 e está no 7º período"*. Um **banco de dados** é uma coleção organizada de dados, guardada de forma persistente e estruturada para ser consultada.

1.4.2 SGBD: o software servidor

O **SGBD** (Sistema de Gerenciamento de Banco de Dados; em inglês, *DBMS*) é o software servidor que cuida desse banco. É ele quem de fato:

- grava e lê os dados em disco;
- garante a integridade, não deixando entrar dado que viole as regras definidas;
- controla o acesso concorrente, permitindo que várias pessoas escrevam ao mesmo tempo sem corromper nada;
- controla as permissões dos usuários.

Correção de um erro comum

SQL **não** é a única forma de conversar com um SGBD. A comunicação real acontece por um **protocolo de rede binário** — o MySQL escuta na porta 3306. O SQL é a linguagem que trafega dentro desse protocolo: é a interface padrão e dominante, mas não a única. O MySQL também expõe o *X DevAPI / Document Store*, uma API de CRUD sem SQL. E, quando se usa um ORM (Prisma, Eloquent, Hibernate, SQLAlchemy), escrevem-se objetos — é o ORM que traduz aquilo para SQL antes de enviar.

1.4.3 O modelo relacional

O **modelo relacional**, proposto por Edgar F. Codd em 1970 (Codd 1970), é a ideia central: os dados são organizados em **tabelas** (relações). Cada **linha** (registro ou tupla) é uma ocorrência do mundo real; cada **coluna** (campo ou atributo) é uma característica dela, com um **tipo** fixo. Tabelas se ligam a outras tabelas por meio de **chaves** — daí o nome "relacional".

```
TABELA membro
+----+--------------+--------------------+---------+
| id | nome         | email              | periodo |  <- colunas (com
   ↪ tipo)
+----+--------------+--------------------+---------+
|  1 | Ana Souza    | ana@liga.edu.br    |       7 |  <- linha (registro)
|  2 | Bruno Lima   | bruno@liga.edu.br  |       3 |
+----+--------------+--------------------+---------+
  ^
  chave primaria: identifica cada linha de forma unica
```

Listing 1.1 – Anatomia de uma tabela

SGBD relacional

Um **SGBD relacional** é um SGBD que implementa esse modelo — é o que a sigla **RDBMS** significa. MySQL, PostgreSQL, SQL Server, Oracle Database e SQLite são todos RDBMS.

2 MySQL

2.1 O que é o MySQL

O MySQL é um **SGBD relacional** de código aberto, criado em 1995 e hoje mantido pela Oracle. É o motor que armazena e gerencia os dados; a conversa com ele acontece em SQL.

É provavelmente o banco relacional mais usado em aplicações web: WordPress, boa parte dos sistemas em PHP e uma infinidade de APIs rodam sobre ele. A versão usada nesta capacitação é a **8.4 LTS**, e o motor de armazenamento padrão é o **InnoDB**.

Por que o InnoDB importa

É o InnoDB que dá suporte a **chaves estrangeiras** e a **transações**. O motor antigo, MyISAM, não suporta chave estrangeira: ele aceita a declaração e a ignora silenciosamente. Como todo o Capítulo 6 depende de chave estrangeira, as tabelas deste guia são criadas com `ENGINE=InnoDB` explícito.

2.2 MySQL e MariaDB: a confusão mais comum

Tabela 2 – Comparação entre MySQL e MariaDB

	MySQL	**MariaDB**
Origem	Original, de 1995	*Fork* de 2009, feito pelos criadores do MySQL após a compra pela Oracle
Licença da edição livre	*Community Edition*: GPLv2, gratuita	*Server*: GPLv2, gratuita
Edições pagas	Sim (*Standard* e *Enterprise*, da Oracle)	Sim (suporte e produtos da MariaDB plc)
Compatibilidade	Altíssima nos comandos básicos: tudo desta capacitação funciona igual nos dois	

uma edição livre em GPLv2 e ambos têm ofertas comerciais. A diferença que interessa agora é outra: várias distribuições Linux (Debian, Ubuntu, Fedora, RHEL) empacotam o **MariaDB** como padrão nos repositórios oficiais, e o XAMPP também entrega MariaDB. Para instalar o MySQL da Oracle especificamente, é preciso adicionar o repositório oficial do MySQL ou usar Docker.
Para o conteúdo desta capacitação, **tanto faz**: se o seu ambiente for MariaDB, siga normalmente.

2.3 Como executar o MySQL

Existem três caminhos. Escolha apenas um.

2.3.1 Opção A: pacote tudo-em-um

Instala servidor web, PHP, banco e phpMyAdmin de uma vez só. É o caminho mais fácil para quem está começando.

- **XAMPP** — Windows, Linux e macOS. É o mais popular. A sigla é **X** (*cross-platform*) + **A**pache + **M**ariaDB + **P**HP + **P**erl. Em versões antigas o "M" era MySQL; hoje é MariaDB.
- **WampServer** — apenas Windows. Aí sim: **W**indows + **A**pache + **M**ySQL + **P**HP.
- **Laragon** — apenas Windows, alternativa mais leve e moderna.

Conflito de portas

Se o MySQL não subir, quase sempre há outro serviço ocupando a porta 3306 (outra instalação de MySQL, ou o SQL Server). A solução é mudar a porta no `my.ini` ou `my.cnf` (`port=3307`) e informar essa porta no cliente.

2.3.2 Opção B: instalação nativa do servidor

- **Windows:** *MySQL Installer for Windows*, no site oficial.
- **macOS:** `brew install mysql` (Homebrew) ou o `.dmg` oficial.
- **Linux (Debian/Ubuntu):** `sudo apt install mysql-server`. Atenção: em várias distribuições esse pacote resolve para MariaDB. Para o MySQL da Oracle, adicione antes o repositório APT oficial.

Depois de instalar, execute `sudo mysql_secure_installation` para definir a senha do usuário `root`.

2.3.3 Opção C: Docker

Aviso de escopo

Docker **não** faz parte desta capacitação. Está aqui apenas para documentar como o ambiente da aula foi montado e para estimular quem quiser pesquisar depois. Os arquivos prontos estão na pasta `infrastructure/` do repositório.

O arquivo de orquestração é um `compose.yml`, que é **diferente de um** `Dockerfile`: o `Dockerfile` descreve *como construir uma imagem*; o `compose.yml` descreve *quais serviços prontos subir e como conectá-los*. Como aqui só usamos imagens prontas, só existe `compose.yml`.

```
cd infrastructure
cp .env.example .env       # preencha MYSQL_ROOT_PASSWORD
docker compose up -d       # sobe MySQL + phpMyAdmin em segundo plano
docker compose ps          # confere se estao de pe e saudaveis
```

Listing 2.1 – Subindo o ambiente

```
docker compose down        # para os containers, PRESERVA os dados
docker compose down -v     # para e APAGA o volume (perde tudo)
```

Listing 2.2 – Derrubando o ambiente

`docker compose`, sem hífen

O comando correto é `docker compose` (versão 2, um plugin do Docker). O antigo `docker-compose` com hífen é a versão 1, descontinuada desde 2023.

A área dos alunos é preparada por um script que roda automaticamente na primeira subida do container:

```
-- Banco de demonstracao do instrutor: so o root tem acesso.
CREATE DATABASE IF NOT EXISTS cap CHARACTER SET utf8mb4;

-- Area dos alunos. O GRANT abaixo e sobre um PADRAO de nome, nao sobre
-- um banco que ja exista: `_` e `%` sao curingas em GRANT, por isso o
-- underscore literal precisa do escape `\_`.
-- Resultado: 'aluno' pode criar e administrar liga_ana, liga_bruno, ...
-- e nenhum outro banco.
GRANT ALL PRIVILEGES ON `liga\_%`.* TO 'aluno'@'%';
FLUSH PRIVILEGES;
```

Listing 2.3 – infrastructure/init/01_create_databases.sql (trecho)

Scripts de init rodam uma única vez

Os arquivos em `init/` são executados apenas na primeira subida, com o volume de dados vazio. Se você editar um deles depois, é preciso zerar o volume para reaplicar: `docker compose down -v && docker compose up -d`.

3 Clientes SQL

Um **cliente SQL** é o programa pelo qual *você* conversa com o servidor: ele abre a conexão, envia os comandos SQL e exibe o resultado em tabela. O servidor (MySQL) e o cliente são programas separados — podem, inclusive, estar em máquinas diferentes.

3.1 Panorama de clientes

Tabela 3 – Clientes SQL mais comuns para MySQL

Cliente	Tipo	Plataformas	Custo
`mysql` (CLI)	Terminal	Win / Linux / macOS	Gratuito, vem com o servidor
MySQL Workbench	Desktop	Win / Linux / macOS	Gratuito (GPL)
DBeaver	Desktop	Win / Linux / macOS	*Community* gratuito; PRO pago
HeidiSQL	Desktop	Windows	Gratuito
phpMyAdmin	**Web**	navegador	Gratuito
TablePlus	Desktop	Win / Linux / macOS	Versão gratuita limitada; pago
DBVisualizer	Desktop	Win / Linux / macOS	Versão gratuita limitada; Pro pago
Navicat	Desktop	Win / Linux / macOS	**Pago** (apenas teste grátis)

Correção de um erro comum

Navicat, DBVisualizer e SQLPro Studio **não são gratuitos**: são produtos comerciais, com versão de teste ou edição gratuita limitada. Não os recomende como "gratuitos".

3.2 phpMyAdmin

O phpMyAdmin **não é um programa que se instala no sistema operacional** como os outros da lista: é uma **aplicação web escrita em PHP** que roda num servidor web e é acessada pelo **navegador**. É por isso que ele aparece junto do Apache e do PHP em pacotes como o XAMPP.

Ele foi escolhido para esta capacitação por três motivos: não exige instalação na máquina do participante (basta uma URL), mostra a estrutura do banco visualmente enquanto se aprende, e tem uma aba **SQL** onde os comandos são digitados de verdade.

3.3 Conectando

Nesta capacitação **ninguém instala nada**. O instrutor roda o MySQL e o phpMyAdmin na máquina dele e publica apenas o phpMyAdmin na internet por um túnel. O acesso é por uma URL, no navegador.

Tabela 4 – Dados de acesso da turma

Campo	Valor
URL do phpMyAdmin	Ditada pelo instrutor no início da aula
Usuário	`aluno`
Senha	`aluno`

Tela de aviso do túnel

Na primeira visita, o serviço de túnel mostra uma página de aviso antes do site. Clique em *"Visit Site"* uma vez e siga; ela não aparece de novo.

3.3.1 Cada participante tem o seu próprio banco

O servidor é um só, compartilhado pela turma inteira. Se todos trabalhassem no mesmo banco, o `CREATE TABLE` do segundo participante já falharia com `ERROR 1050: Table already exists`, e um `DELETE` de qualquer um apagaria o dado de todos.

Por isso **cada participante cria o próprio banco**, chamado `liga_` seguido do seu primeiro nome:

`liga_ana` `liga_bruno` `liga_carla` `liga_diego`

Regra do nome

Apenas letras minúsculas, sem acento e sem espaço: `liga_joao`, nunca `liga_João` nem `liga_joao silva`.
Onde este guia escrever `liga_SEUNOME`, use o seu. O login `aluno` só tem permissão em bancos que comecem com `liga_`; qualquer outro nome retorna `ERROR 1044: Access denied`. Isso é proposital: protege o banco de demonstração do instrutor (`cap`) e os dados internos do MySQL.

3.3.2 Teste de sanidade

Abra a aba **SQL** e execute:

```
SELECT VERSION(), NOW();
```

Se voltou a versão do servidor e a data e hora, está tudo certo.

4 SQL: fundamentos e definição de estruturas

SQL (*Structured Query Language*, "linguagem de consulta estruturada") é a linguagem padronizada pela ANSI e pela ISO para definir e manipular dados em bancos relacionais. Ela é **declarativa**: descreve-se *o que* se quer, não *como* buscar. O SGBD decide o caminho.

Cada SGBD tem seu "sotaque" (dialeto), mas o núcleo é o mesmo — o que se aprende aqui vale, com ajustes pequenos, para PostgreSQL, SQL Server e SQLite.

4.1 SQL não é um bloco só

Os comandos SQL são agrupados por finalidade. Conhecer esse mapa evita muita confusão.

Tabela 5 – Sublinguagens do SQL

Sigla	Nome	Para quê	Comandos
DDL	*Data Definition Language*	Define a **estrutura**	`CREATE`, `ALTER`, `DROP`
DML	*Data Manipulation Language*	Manipula os **dados**	`INSERT`, `UPDATE`, `DELETE`
DQL	*Data Query Language*	**Consulta** os dados	`SELECT`
DCL	*Data Control Language*	Controla **permissões**	`GRANT`, `REVOKE`
TCL	*Transaction Control Language*	Controla **transações**	`COMMIT`, `ROLLBACK`

Dica

Muitos autores tratam o `SELECT` como parte do DML. Os dois usos existem na literatura; o importante é entender a diferença entre **mexer na estrutura** (DDL) e **mexer nos dados** (DML e DQL). Nesta capacitação usamos DDL, DML e DQL.

4.2 A ordem de trabalho

Como o MySQL é relacional, sempre existe uma estrutura antes do dado. A ordem nunca muda:

```
1. CREATE DATABASE  ->  criar o banco          | DDL: a estrutura,
2. USE              ->  selecionar o banco     | feita uma vez
3. CREATE TABLE     ->  criar a(s) tabela(s)   |
                    v
```

```
4. INSERT          ->  inserir dados          | DML/DQL: o dia a dia,
5. SELECT          ->  consultar dados        | repetido o tempo todo
6. UPDATE          ->  alterar dados          |
7. DELETE          ->  remover dados          |
```

Listing 4.1 – Do banco vazio ao dado consultado

Ponto que costuma passar batido

Criar tabela e inserir dados não são o mesmo tipo de operação. Criar tabela é DDL, feito uma vez no projeto. `INSERT`, `UPDATE` e `DELETE` são DML, executados milhares de vezes pela aplicação.

O cenário usado no restante deste guia é um cadastro de membros da liga acadêmica, com duas tabelas: `curso` e `membro`.

4.3 Criando o banco

Atenção

Troque `SEUNOME` pelo seu primeiro nome, aqui e em todo o resto do guia. Se você é a Ana, seu banco é `liga_ana`. Veja a Seção 3.3.1.

```sql
CREATE DATABASE IF NOT EXISTS liga_SEUNOME
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_0900_ai_ci;

USE liga_SEUNOME;
```

- `IF NOT EXISTS` evita erro se o banco já existir.
- `utf8mb4` é o conjunto de caracteres que suporta **acentos e emojis** corretamente. É o padrão do MySQL 8, mas declará-lo explicitamente é boa prática: em servidores antigos o padrão era `latin1`, que quebra a acentuação.
- `USE liga_SEUNOME;` define o banco corrente. **Sem isso, os comandos seguintes não sabem onde agir** e o servidor responde `ERROR 1046: No database selected`. No phpMyAdmin, clicar no banco na barra lateral tem o mesmo efeito.

Comandos úteis de inspeção:

```sql
SHOW DATABASES;              -- lista os bancos visiveis para voce
SHOW TABLES;                 -- lista as tabelas do banco corrente
DESCRIBE membro;             -- mostra as colunas e tipos de uma tabela
SHOW CREATE TABLE membro;    -- mostra o CREATE TABLE completo
```

4.4 Tipos de dados

Cada coluna tem um **tipo**, e é ele que garante que a coluna `periodo` nunca receba `"banana"`. Escolher bem o tipo é boa parte da qualidade de uma tabela.

Tabela 6 – Tipos de dados mais usados no MySQL

Tipo	Guarda	Quando usar
`INT`	Inteiro (cerca de $\pm$2,1 bilhões)	IDs, contadores, quantidades
`TINYINT`	Inteiro pequeno	Período, idade
`BIGINT`	Inteiro muito grande	IDs de sistemas de altíssimo volume
`DECIMAL(p,s)`	Número exato com casas decimais	**Dinheiro**: `DECIMAL(10,2)`
`FLOAT` / `DOUBLE`	Número aproximado	Medidas científicas. **Nunca para dinheiro**
`VARCHAR(n)`	Texto variável, até n	Nome, e-mail, título: o caso mais comum
`CHAR(n)`	Texto de tamanho **fixo**	Siglas: `CHAR(2)` para UF
`TEXT`	Texto longo (até 64 KB)	Descrições, observações
`DATE`	Data (`AAAA-MM-DD`)	Data de nascimento, de ingresso
`DATETIME`	Data e hora	Agendamentos, registro de eventos
`TIMESTAMP`	Data e hora, sensível a fuso	`criado_em`, `atualizado_em`
`BOOLEAN`	Verdadeiro ou falso	Sinalizadores. No MySQL é apelido de `TINYINT(1)`
`ENUM(...)`	Um valor de uma lista fixa	Status: `ENUM('ativo','inativo')`

Regra de ouro

Dinheiro em `DECIMAL`, **nunca** em `FLOAT`. `FLOAT` é aproximado, e $0{,}1 + 0{,}2$ pode não dar exatamente $0{,}3$.

`NULL` não é zero nem string vazia

`NULL` significa "valor desconhecido ou não informado". Por isso `WHERE curso_id = NULL` **nunca** funciona: o correto é `WHERE curso_id IS NULL`.

4.5 Criando a tabela

```sql
CREATE TABLE curso (
  id   INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(80) NOT NULL UNIQUE
) ENGINE=InnoDB;
```

```sql
CREATE TABLE membro (
  id             INT           AUTO_INCREMENT PRIMARY KEY,
  nome           VARCHAR(120)  NOT NULL,
  email          VARCHAR(160)  NOT NULL UNIQUE,
  periodo        TINYINT       NOT NULL,
  data_ingresso  DATE          NOT NULL,
  ativo          BOOLEAN       NOT NULL DEFAULT TRUE,
  curso_id       INT           NULL,
  criado_em      TIMESTAMP     NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

As **restrições** (*constraints*) são onde mora a qualidade do dado.

Tabela 7 – Restrições de coluna

Restrição	O que faz
`PRIMARY KEY`	Identifica cada linha de forma **única**. Não aceita `NULL`, não repete. Uma por tabela
`AUTO_INCREMENT`	O MySQL gera o próximo número sozinho; esse campo não é informado no `INSERT`
`NOT NULL`	O campo é obrigatório
`UNIQUE`	O valor não pode se repetir na coluna
`DEFAULT x`	Valor assumido quando o `INSERT` não informa a coluna
`FOREIGN KEY`	Liga esta tabela a outra (Capítulo 6)

`PRIMARY KEY` e `UNIQUE`

As duas impedem repetição. A diferença: só existe **uma** chave primária por tabela e ela **não aceita** `NULL`; já `UNIQUE` pode haver várias na mesma tabela, e ela aceita `NULL`.

Alterar a estrutura depois também é DDL:

```sql
ALTER TABLE membro ADD COLUMN telefone VARCHAR(20) NULL;
ALTER TABLE membro MODIFY COLUMN nome VARCHAR(150) NOT NULL;
ALTER TABLE membro DROP COLUMN telefone;

DROP TABLE membro;     -- apaga a tabela E todos os dados, sem desfazer
```

5 CRUD: manipulando os dados

CRUD é o acrônimo das quatro operações básicas sobre dados. Todo sistema que você escrever na vida faz essas quatro coisas.

Tabela 8 – As quatro operações do CRUD

CRUD	Comando SQL	Categoria
Create	`INSERT`	DML
Read	`SELECT`	DQL
Update	`UPDATE`	DML
Delete	`DELETE`	DML

5.1 Create: `INSERT`

```sql
-- Uma linha
INSERT INTO curso (nome) VALUES ('Engenharia de Software');

-- Varias linhas de uma vez (mais eficiente)
INSERT INTO curso (nome) VALUES
  ('Ciencia da Computacao'),
  ('Sistemas de Informacao'),
  ('Engenharia de Producao');

INSERT INTO membro (nome, email, periodo, data_ingresso, curso_id) VALUES
  ('Ana Souza',   'ana@liga.edu.br',   7, '2024-03-11', 1),
  ('Bruno Lima',  'bruno@liga.edu.br', 3, '2025-02-20', 2),
  ('Carla Dias',  'carla@liga.edu.br', 5, '2024-08-05', 1),
  ('Diego Rocha', 'diego@liga.edu.br', 9, '2023-09-14', 3),
  ('Elisa Prado', 'elisa@liga.edu.br', 1, '2026-03-02', NULL);
```

Repare que `id`, `ativo` e `criado_em` não foram informados: são preenchidos pelo `AUTO_INCREMENT` e pelos `DEFAULT`.

- Datas vão sempre no formato `'AAAA-MM-DD'` e entre aspas.
- Texto entre aspas simples (`'Ana'`); números sem aspas (`7`).
- A ordem dos valores em `VALUES` deve corresponder exatamente à ordem das colunas listadas.

5.2 Read: `SELECT`

O comando mais usado da linguagem. A estrutura completa é:

```sql
SELECT   colunas
FROM     tabela
WHERE    condicao          -- filtra LINHAS
ORDER BY coluna [ASC|DESC] -- ordena
LIMIT    n;                -- limita a quantidade
```

```sql
SELECT * FROM membro;                    -- tudo (bom para explorar, ruim
    ↪ em producao)
SELECT nome, email FROM membro;          -- so as colunas que interessam
SELECT nome AS aluno, periodo AS sem FROM membro;  -- AS renomeia no
    ↪ resultado
```

5.2.1 Filtrando com `WHERE`

```sql
SELECT * FROM membro WHERE periodo > 5;
SELECT * FROM membro WHERE ativo = TRUE AND periodo >= 5;
SELECT * FROM membro WHERE periodo IN (1, 3, 5);
SELECT * FROM membro WHERE periodo BETWEEN 3 AND 7;    -- inclusivo nas 2
    ↪ pontas
SELECT * FROM membro WHERE nome LIKE 'A%';             -- comeca com A
SELECT * FROM membro WHERE email LIKE '%@liga.edu.br'; -- termina com
SELECT * FROM membro WHERE nome LIKE '%ana%';          -- contem
SELECT * FROM membro WHERE curso_id IS NULL;           -- sem curso
    ↪ informado
SELECT * FROM membro WHERE data_ingresso >= '2025-01-01';
```

Tabela 9 – Operadores de filtro

Operador	Significado
`=` `<>` (ou `!=`)	Igual / diferente
`>` `<` `>=` `<=`	Comparação
`AND` `OR` `NOT`	Combinação lógica
`IN (a, b, c)`	Está na lista
`BETWEEN a AND b`	Está no intervalo, incluindo as pontas
`LIKE`	Padrão de texto: `%` é qualquer coisa, `_` é um caractere
`IS NULL` / `IS NOT NULL`	Testa ausência de valor

5.2.2 Ordenando e limitando

```sql
SELECT nome, periodo FROM membro ORDER BY periodo DESC;
SELECT nome, periodo FROM membro ORDER BY periodo DESC, nome ASC;
SELECT nome FROM membro ORDER BY data_ingresso ASC LIMIT 3;  -- os 3 mais
    ↪ antigos
```

`ASC` é crescente (padrão, pode ser omitido) e `DESC` é decrescente.

5.3 Update: alterando

```sql
UPDATE membro
SET    periodo = 8
WHERE  id = 1;

UPDATE membro
SET    ativo = FALSE, periodo = 10
WHERE  email = 'diego@liga.edu.br';
```

5.4 Delete: removendo

Para praticar sem destruir os dados da aula, crie um registro descartável e apague apenas ele:

```sql
INSERT INTO membro (nome, email, periodo, data_ingresso)
VALUES ('Registro Teste', 'teste@liga.edu.br', 1, '2026-01-01');

SELECT * FROM membro WHERE email = 'teste@liga.edu.br';    -- confira
    ↪ ANTES
DELETE FROM membro WHERE email = 'teste@liga.edu.br';
```

5.5 O erro mais caro de quem está começando

Sem `WHERE`, o comando vale para a tabela inteira

```sql
UPDATE membro SET periodo = 1;    -- zera o periodo de TODOS os
    ↪ membros
DELETE FROM membro;               -- apaga TODOS os membros da tabela
```

E não existe "desfazer" em SQL.

O hábito obrigatório é testar com `SELECT` antes:

```sql
-- 1) veja EXATAMENTE quais linhas serao afetadas
SELECT * FROM membro WHERE id = 5;

-- 2) so entao troque o SELECT * pelo DELETE ou UPDATE,
--    mantendo exatamente o mesmo WHERE
DELETE FROM membro WHERE id = 5;
```

Rede de proteção, e a falta dela

O MySQL Workbench tem o *safe update mode*, que **bloqueia** `UPDATE` e `DELETE` sem `WHERE` em coluna-chave. O phpMyAdmin **não** protege: nele, a disciplina é sua.

`DELETE FROM tabela;` apaga as linhas uma a uma (é DML). `TRUNCATE TABLE tabela;` esvazia a tabela de forma muito mais rápida e reinicia o `AUTO_INCREMENT`, mas é DDL e não pode ser desfeito por `ROLLBACK`.

6 Relacionamentos: a parte relacional

Até aqui usamos uma tabela só — isso é uma planilha, não um banco relacional. O que muda tudo é **relacionar** tabelas.

6.1 O problema que a chave estrangeira resolve

Se guardássemos o nome do curso dentro de `membro`, teríamos "Engenharia de Software" repetido em dezenas de linhas. Aí alguém digita "Eng. de Software", outro digita "engenharia de software", e a consulta por curso quebra. Corrigir o nome exigiria alterar todas as linhas.

A solução: o curso vira uma tabela própria, e `membro` guarda apenas o `id` do curso.

```
   curso                          membro
+----+----------------------+   +----+------------+----------+
| id | nome                 |   | id | nome       | curso_id |
+----+----------------------+   +----+------------+----------+
|  1 | Eng. de Software     |<--|  1 | Ana Souza  |        1 |
|  2 | Ciencia da Comp.     |<--|  2 | Bruno Lima |        2 |
+----+----------------------+   |  3 | Carla Dias |        1 |
       ^                        +----+------------+----------+
  chave primaria (PK)                          chave estrangeira (FK)
```

Listing 6.1 – Duas tabelas ligadas por chave

PK e FK

A **chave primária (PK)** identifica a linha *dentro* da própria tabela. A **chave estrangeira (FK)** é uma coluna que *aponta* para a PK de outra tabela. Com ela declarada, o SGBD passa a **garantir** que aquele valor existe do outro lado — isso se chama **integridade referencial**.

6.2 Declarando a chave estrangeira

```
ALTER TABLE membro
  ADD CONSTRAINT fk_membro_curso
  FOREIGN KEY (curso_id) REFERENCES curso(id)
  ON DELETE SET NULL
  ON UPDATE CASCADE;
```

Com a chave estrangeira ativa, o comando abaixo passa a **falhar** — e é exatamente o que queremos:

```
INSERT INTO membro (nome, email, periodo, data_ingresso, curso_id)
VALUES ('Fake', 'fake@liga.edu.br', 1, '2026-01-01', 999);
-- ERROR 1452: Cannot add or update a child row:
--             a foreign key constraint fails
```

Tabela 10 – Ações de `ON DELETE` e `ON UPDATE`

Ação	Efeito ao apagar ou alterar o curso
`RESTRICT` (padrão)	Impede a operação enquanto houver membro apontando para ele
`SET NULL`	Os membros ficam com `curso_id = NULL`; exige coluna que aceite `NULL`
`CASCADE`	Propaga: apagar o curso apagaria os membros. **Use com muito cuidado**

6.3 Cardinalidades

- **1:N (um para muitos)** — o caso mais comum. Um curso tem vários membros; cada membro tem um curso. **A chave estrangeira fica no lado "muitos"**, ou seja, em `membro`. É o nosso caso.
- **1:1** — chave estrangeira com `UNIQUE`. Raro.
- **N:N (muitos para muitos)** — por exemplo, um membro participa de vários projetos e um projeto tem vários membros. Resolve-se com uma **tabela intermediária** (`membro_projeto`) que guarda duas chaves estrangeiras. Conceito citado, não praticado nesta capacitação.

6.4 `JOIN`: consultando as duas tabelas juntas

`JOIN` combina linhas de tabelas diferentes usando a condição de ligação declarada no `ON`.

```
-- INNER JOIN: so os membros QUE TEM curso
SELECT m.nome AS membro,
       c.nome AS curso,
       m.periodo
FROM   membro m
INNER  JOIN curso c ON m.curso_id = c.id
ORDER  BY c.nome, m.nome;
```

```
-- LEFT JOIN: TODOS os membros, tenham curso ou nao.
-- Elisa aparece com curso = NULL.
SELECT m.nome AS membro,
       c.nome AS curso
```

```
FROM   membro m
LEFT   JOIN curso c ON m.curso_id = c.id;
```

Tabela 11 – Tipos de `JOIN`

Tipo	Retorna
`INNER JOIN`	Só as linhas com correspondência **nos dois lados**
`LEFT JOIN`	**Todas** as da tabela da esquerda, mais as correspondentes da direita (`NULL` se não houver)
`RIGHT JOIN`	O espelho do `LEFT`; raro, basta inverter a ordem das tabelas

Apelidos e a armadilha do `ON`

`membro m` e `curso c` são **apelidos** (*aliases*). Com duas tabelas em jogo, `m.nome` e `c.nome` deixam claro de qual tabela é cada coluna; sem isso o MySQL responde `ERROR 1052: Column 'nome' in field list is ambiguous`. A condição do `ON` é sempre **FK = PK**. Esquecer o `ON` produz um produto cartesiano: cada membro cruzado com cada curso.

6.5 Bônus: contar e agrupar

Dica

Este conteúdo entra somente se o cronograma permitir (Seção 1.3). Caso contrário, fica como material de estudo.

Funções de agregação resumem várias linhas em um valor: `COUNT`, `SUM`, `AVG`, `MIN` e `MAX`.

```
SELECT COUNT(*) FROM membro;       -- quantos membros existem
SELECT AVG(periodo) FROM membro;   -- periodo medio
SELECT MIN(data_ingresso), MAX(data_ingresso) FROM membro;
```

`GROUP BY` aplica a agregação **por grupo**:

```
SELECT c.nome         AS curso,
       COUNT(m.id)    AS total_membros
FROM   curso c
LEFT   JOIN membro m ON m.curso_id = c.id
GROUP  BY c.id, c.nome
ORDER  BY total_membros DESC;
```

`WHERE` e `HAVING`

`WHERE` filtra **linhas antes** de agrupar; `HAVING` filtra **grupos depois** de agregar. Por exemplo: `... GROUP BY c.id, c.nome HAVING COUNT(m.id) >= 2;`

7 Erros comuns

A tabela a seguir reúne as mensagens que mais aparecem para quem está começando, com a causa e a correção de cada uma. Vale a pena consultá-la antes de pedir ajuda: quase todo erro de aula está aqui.

Tabela 12 – Erros frequentes, causas e correções

Erro ou mensagem	Causa	Correção
`ERROR 1046: No database selected`	Faltou escolher o banco	`USE liga_SEUNOME;` ou clicar no banco na lateral
`ERROR 1044: Access denied to database`	Nome fora do padrão `liga_*`, ou tentativa de abrir o banco `cap`	Use `liga_SEUNOME`: underscore, minúsculas, sem acento
`ERROR 1050: Table already exists`	O `CREATE TABLE` foi executado duas vezes	`DROP TABLE membro;` antes, ou rode o script do Apêndice B inteiro
`ERROR 1062: Duplicate entry`	Violou `PRIMARY KEY` ou `UNIQUE`	O valor já existe; use outro ou faça `UPDATE`
`ERROR 1452: foreign key constraint fails`	Chave estrangeira apontando para um `id` inexistente	Insira primeiro na tabela-pai (`curso`)
`ERROR 1452` ao apagar da tabela-pai	Há filhos referenciando aquela linha	Trate os filhos antes, ou defina `ON DELETE`
`ERROR 1052: Column is ambiguous`	Duas tabelas com coluna de mesmo nome	Qualifique: `m.nome`, `c.nome`
`UPDATE` ou `DELETE` afetou a tabela toda	`WHERE` esquecido	Sempre testar com `SELECT` antes (Seção 5.5)
`WHERE campo = NULL` não retorna nada	`NULL` não é comparável com `=`	`WHERE campo IS NULL`
Acentuação vira caractere estranho	Conjunto de caracteres errado (`latin1`)	Crie o banco com `utf8mb4`
Datas não filtram direito	Formato ou tipo errado	Use `DATE`/`DATETIME` e `'AAAA-MM-DD'`
Valores monetários erram centavos	Coluna declarada como `FLOAT`	Use `DECIMAL(10,2)`

Tabela 12 — continuação

Erro ou mensagem	Causa	Correção
O comando não executa	Falta o ponto e vírgula final	Termine todo comando com ;

8 Conclusão e próximos passos

Em duas horas percorremos o caminho completo de um banco relacional: **entender o modelo**, **subir o servidor**, **conectar por um cliente**, **definir a estrutura** (DDL), **manipular os dados** (CRUD) e **relacionar tabelas** (chave estrangeira e `JOIN`). Isso é suficiente para modelar e operar o banco de um primeiro projeto da liga.

8.1 O que estudar depois

Nesta ordem:

1. **Normalização (1FN, 2FN, 3FN)** — como decidir quantas tabelas o seu projeto precisa (Elmasri e Navathe 2016).
2. **Modelagem entidade-relacionamento** — desenhar o modelo antes de escrever SQL.
3. **Relacionamento N:N** com tabela intermediária, e `JOIN` de três ou mais tabelas.
4. **Agregação avançada**: `GROUP BY`, `HAVING` e subconsultas.
5. **Índices** — por que uma consulta fica lenta e como acelerá-la (`EXPLAIN`).
6. **Transações** — `START TRANSACTION`, `COMMIT`, `ROLLBACK` e o conceito ACID.
7. **Integração com aplicação** — conectar via linguagem (Python, PHP, Node) e conhecer os ORMs.
8. **Backup e restauração** — `mysqldump`.

8.2 Onde praticar sem instalar nada

- SQLite Online — vários bancos direto no navegador
- DB Fiddle — MySQL e PostgreSQL no navegador
- OneCompiler MySQL
- SQLZoo e SQLBolt — exercícios interativos guiados
- HackerRank SQL — desafios com correção automática

A documentação oficial do MySQL 8.4 (Oracle Corporation 2024) é a referência definitiva para qualquer dúvida de sintaxe.

8.3 Materiais desta capacitação

- `docs/` — este guia e a apresentação
- `materials/liga.sql` — o script completo da aula, pronto para colar no phpMyAdmin
- `infrastructure/` — `compose.yml`, `.env.example` e os scripts de inicialização do banco

Referências

Codd 1970 CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, v. 13, n. 6, p. 377–387, 1970.

Elmasri e Navathe 2016 ELMASRI, R.; NAVATHE, S. B. *Sistemas de Banco de Dados*. 7. ed. São Paulo: Pearson, 2016.

Oracle Corporation 2024 Oracle Corporation. *MySQL 8.4 Reference Manual*. 2024. <https://dev.mysql.com/doc/refman/8.4/en/>.

Apêndices

APÊNDICE A – Colinha de sintaxe

```
CREATE DATABASE nome CHARACTER SET utf8mb4;
USE nome;
CREATE TABLE t (id INT AUTO_INCREMENT PRIMARY KEY, campo VARCHAR(100) NOT
    ↪  NULL);
ALTER TABLE t ADD COLUMN novo VARCHAR(50);
ALTER TABLE t MODIFY COLUMN campo VARCHAR(200) NOT NULL;
ALTER TABLE t DROP COLUMN novo;
DROP TABLE t;
```

Listing A.1 – Estrutura (DDL)

```
SHOW DATABASES;   SHOW TABLES;   DESCRIBE t;   SHOW CREATE TABLE t;
```

Listing A.2 – Inspeção

```
INSERT INTO t (c1, c2) VALUES (v1, v2), (v3, v4);
SELECT c1, c2 FROM t WHERE cond ORDER BY c1 DESC LIMIT 10;
UPDATE t SET c1 = v WHERE id = 1;       -- SEMPRE com WHERE
DELETE FROM t WHERE id = 1;             -- SEMPRE com WHERE
```

Listing A.3 – Dados (DML e DQL)

```
WHERE a = 1 AND b <> 2
WHERE a IN (1,2,3)          WHERE a BETWEEN 1 AND 10
WHERE nome LIKE 'A%'        WHERE campo IS NULL
```

Listing A.4 – Filtros

```
FOREIGN KEY (curso_id) REFERENCES curso(id) ON DELETE SET NULL;
SELECT m.nome, c.nome FROM membro m INNER JOIN curso c ON m.curso_id = c.
    ↪ id;
SELECT m.nome, c.nome FROM membro m LEFT  JOIN curso c ON m.curso_id = c.
    ↪ id;
```

Listing A.5 – Relacionamento

```
SELECT c.nome, COUNT(*) FROM curso c JOIN membro m ON m.curso_id = c.id
GROUP BY c.id, c.nome HAVING COUNT(*) >= 2;
```

Listing A.6 – Agregação

APÊNDICE B – Script completo da aula

Execute do início ao fim para reconstruir todo o cenário da capacitação. O arquivo também está disponível no repositório, em `materials/liga.sql`.

Antes de colar

Substitua as três ocorrências de `liga_SEUNOME` pelo seu banco — `liga_ana`, `liga_joao` e assim por diante. Se você colar sem trocar, vai criar um banco literalmente chamado `liga_seunome` e disputá-lo com quem fizer o mesmo.

```
-- ============================================================
-- Capacitação de MySQL - Liga Acadêmica
-- Script completo da aula (Anexo B do guia)
-- ============================================================
-- COMO USAR
--   1. Troque `liga_SEUNOME` pelo seu banco nas linhas 22, 23 e 24.
--      Ex.: liga_ana, liga_joao, liga_maria.
--      Só letras minúsculas, sem acento e sem espaço.
--   2. Cole no phpMyAdmin, aba SQL, e execute.
--
-- O login da turma só tem permissão em bancos que começam com
-- "liga_". Qualquer outro nome retorna ERROR 1044.
-- ============================================================

-- ============================================================
-- Capacitação MySQL - Liga Acadêmica
-- Cenário: cadastro de membros e cursos
-- ============================================================

-- 1. BANCO ----------------------------------------------------
-- O DROP zera SÓ o seu banco, para o script poder ser rodado de novo.
DROP DATABASE IF EXISTS liga_SEUNOME;
CREATE DATABASE liga_SEUNOME CHARACTER SET utf8mb4 COLLATE
    ↪ utf8mb4_0900_ai_ci;
USE liga_SEUNOME;

-- 2. ESTRUTURA (DDL) ------------------------------------------
CREATE TABLE curso (
  id    INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(80) NOT NULL UNIQUE
) ENGINE=InnoDB;

CREATE TABLE membro (
```

```sql
  id              INT           AUTO_INCREMENT PRIMARY KEY,
  nome            VARCHAR(120)  NOT NULL,
  email           VARCHAR(160)  NOT NULL UNIQUE,
  periodo         TINYINT       NOT NULL,
  data_ingresso DATE            NOT NULL,
  ativo           BOOLEAN       NOT NULL DEFAULT TRUE,
  curso_id        INT           NULL,
  criado_em       TIMESTAMP     NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_membro_curso
    FOREIGN KEY (curso_id) REFERENCES curso(id)
    ON DELETE SET NULL
    ON UPDATE CASCADE
) ENGINE=InnoDB;

-- 3. DADOS (INSERT) ----------------------------------------
INSERT INTO curso (nome) VALUES
  ('Engenharia de Software'),
  ('Ciência da Computação'),
  ('Sistemas de Informação'),
  ('Engenharia de Produção');

INSERT INTO membro (nome, email, periodo, data_ingresso, curso_id) VALUES
  ('Ana Souza',   'ana@liga.edu.br',   7, '2024-03-11', 1),
  ('Bruno Lima',  'bruno@liga.edu.br', 3, '2025-02-20', 2),
  ('Carla Dias',  'carla@liga.edu.br', 5, '2024-08-05', 1),
  ('Diego Rocha', 'diego@liga.edu.br', 9, '2023-09-14', 3),
  ('Elisa Prado', 'elisa@liga.edu.br', 1, '2026-03-02', NULL);

-- 4. CONSULTAS (SELECT) ------------------------------------
SELECT * FROM membro;
SELECT nome, email FROM membro WHERE periodo > 5;
SELECT * FROM membro WHERE nome LIKE 'A%';
SELECT * FROM membro WHERE curso_id IS NULL;
SELECT nome, periodo FROM membro ORDER BY periodo DESC;
SELECT nome FROM membro ORDER BY data_ingresso ASC LIMIT 3;

-- 5. ALTERAÇÃO E REMOÇÃO -----------------------------------
SELECT * FROM membro WHERE id = 1;          -- confira ANTES
UPDATE membro SET periodo = 8 WHERE id = 1;

-- registro descartável, criado só para ser apagado
INSERT INTO membro (nome, email, periodo, data_ingresso)
VALUES ('Registro Teste', 'teste@liga.edu.br', 1, '2026-01-01');

SELECT * FROM membro WHERE email = 'teste@liga.edu.br';   -- confira
   ↪ ANTES
DELETE FROM membro WHERE email = 'teste@liga.edu.br';
```

```sql
-- 6. RELACIONAMENTO (JOIN) ------------------------------------
SELECT m.nome AS membro, c.nome AS curso, m.periodo
FROM   membro m
INNER  JOIN curso c ON m.curso_id = c.id
ORDER  BY c.nome, m.nome;

SELECT m.nome AS membro, c.nome AS curso
FROM   membro m
LEFT   JOIN curso c ON m.curso_id = c.id;

-- 7. BÔNUS: AGREGAÇÃO -----------------------------------------
SELECT c.nome AS curso, COUNT(m.id) AS total_membros
FROM   curso c
LEFT   JOIN membro m ON m.curso_id = c.id
GROUP  BY c.id, c.nome
ORDER  BY total_membros DESC;
```

APÊNDICE C – Exercícios

Resolva no seu banco `liga_SEUNOME`, já construído pelo Apêndice B.

1. Liste o nome e o e-mail de todos os membros **ativos**.
2. Mostre os membros do 1º ao 5º período, do mais novo para o mais antigo na liga.
3. Quantos membros ingressaram a partir de 2025?
4. Cadastre um novo curso, `'Engenharia Elétrica'`, e um membro nele.
5. Corrija o e-mail da Carla para `carla.dias@liga.edu.br`.
6. Liste membro e nome do curso, incluindo quem está sem curso.
7. Desative (`ativo = FALSE`) todo membro que ingressou antes de `2024-01-01`.
8. Tente inserir um membro com `curso_id = 999`. Explique a mensagem de erro.

C.1 Respostas

```sql
-- 1
SELECT nome, email FROM membro WHERE ativo = TRUE;

-- 2
SELECT * FROM membro
WHERE periodo BETWEEN 1 AND 5
ORDER BY data_ingresso DESC;

-- 3
SELECT COUNT(*) FROM membro WHERE data_ingresso >= '2025-01-01';
```

Listing C.1 – Exercícios 1 a 3

```sql
-- 4  (o curso PRIMEIRO: a chave estrangeira exige que o pai exista)
INSERT INTO curso (nome) VALUES ('Engenharia Eletrica');
INSERT INTO membro (nome, email, periodo, data_ingresso, curso_id)
VALUES ('Felipe Nunes', 'felipe@liga.edu.br', 2, '2026-03-10',
        (SELECT id FROM curso WHERE nome = 'Engenharia Eletrica'));

-- 5
```

```sql
SELECT * FROM membro WHERE nome = 'Carla Dias';          -- confira
    ↪ antes
UPDATE membro SET email = 'carla.dias@liga.edu.br' WHERE nome = 'Carla
    ↪ Dias';
```

Listing C.2 – Exercícios 4 e 5

```sql
-- 6  (LEFT JOIN, porque INNER JOIN esconderia quem nao tem curso)
SELECT m.nome AS membro, c.nome AS curso
FROM   membro m LEFT JOIN curso c ON m.curso_id = c.id;

-- 7
SELECT * FROM membro WHERE data_ingresso < '2024-01-01';  -- confira
    ↪ antes
UPDATE membro SET ativo = FALSE WHERE data_ingresso < '2024-01-01';

-- 8
INSERT INTO membro (nome, email, periodo, data_ingresso, curso_id)
VALUES ('Teste', 'teste@liga.edu.br', 1, '2026-01-01', 999);
-- ERROR 1452: a chave estrangeira exige que exista um curso com id =
    ↪ 999.
-- Como nao existe, o SGBD REJEITA a insercao. E a integridade
    ↪ referencial
-- protegendo o banco contra dado orfao.
```

Listing C.3 – Exercícios 6 a 8