

# Bases de Dados

PL06 & PL07 – Modelação Física

**Docente:** Diana Ferreira

**Email:** [diana.ferreira@algoritmi.uminho.pt](mailto:diana.ferreira@algoritmi.uminho.pt)

**Horário de Atendimento:**

4ª feira 10h-11h | DI 1.15



# Sumário

**1** Revisão do Modelo Lógico

**2** Instalação do MySQL Server

**3** Instruções SQL de DDL

**4** Instruções SQL de DML

## **Bibliografia:**

- Connolly, T., Begg, C., Database Systems, A Practical Approach to Design, Implementation, and Management , Addison-Wesley, 4a Edição, 2004. **(Chapter 18)**
- Belo, O., "Bases de Dados Relacionais: Implementação com MySQL", FCA – Editora de Informática, 376p, Set 2021. ISBN: 978-972-722-921-5. **(Capítulo 2)**

# Modelação Lógica – MySQL

Quando estamos a construir o modelo lógico de dados no MySQL, é importante ter em consideração os seguintes aspetos:

- Tipo de relacionamento:



## Relacionamentos identificadores (linha cheia)

Quando a chave primária da entidade pai é incluída na chave primária da entidade filho.



- Chave estrangeira e chave primária.



## Relacionamentos não identificadores (linha tracejada)

Quando a chave primária da entidade pai é incluída na entidade filho, mas não como parte da sua chave primária.



- Chave estrangeira NOT NULL.



- Chave estrangeira.

**Nota:** Usamos a participação para definir se a chave estrangeira pode ser nula ou não.

- Direcção do relacionamento:

Os relacionamentos devem começar na relação/tabela que deve alocar a chave estrangeira.

# Modelação Lógica – MySQL

- Valores padrão/por defeito: Devem ser usados caso se queira considerar um valor por *default*.



The screenshot shows the MySQL Workbench interface for defining a column. The column name is 'estado\_civil', the data type is 'CHAR(1)', and the default value is 'S'. The 'Default/Expression' field contains the SQL definition: ``estado_civil` CHAR(1) NULL DEFAULT 'S',`. Below the main field, there are input boxes for 'Data Type: CHAR(1)' and 'Default: 'S''.

- SQL constraints: PK (Primary Key), NN (Not Null), UQ (Unique Index), B (Binary), UN (Unsigned), ZF (Zero Fill), AI (auto increment), G (generated)
  - PK – deve ser usado para atributos que são chave primária;
  - NN – deve ser usado em todos os atributos de chave primária e todos os atributos que não possam ser NULL;
  - UQ – deve ser aplicado sempre que há chaves candidatas, faz com que não hajam valores duplicados na tabela;
  - UN – define que não podem ser inseridos valores negativos nessa coluna.
  - ZF – preenche o valor definido para o campo com zeros até a largura de exibição especificada na definição da coluna. Se a coluna for definida como INT(5) e for introduzido o valor 1 → 00001.

# Modelação Lógica – MySQL

- SQL constraints: PK (Primary Key), NN (Not Null), UQ (Unique Index), B (Binary), UN (Unsigned), ZF (Zero Fill), AI (auto increment), G (generated)
  - AI – deve ser usado para gerar um número único automaticamente quando um novo registo é inserido na tabela.
  - G – deve ser usado para gerar atributos a partir de outros usando uma expressão.

NOTAÇÃO: <column\_name> <data\_type> GENERATED ALWAYS AS (<expression>)

```
CREATE TABLE contacts (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  first_name VARCHAR(50) NOT NULL,  
  last_name VARCHAR(50) NOT NULL,  
  email VARCHAR(100) NOT NULL  
);
```

```
SELECT  
  id,  
  CONCAT(first_name, ' ', last_name),  
  email  
FROM  
  contacts;
```

```
CREATE TABLE contacts (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  first_name VARCHAR(50) NOT NULL,  
  last_name VARCHAR(50) NOT NULL,  
  fullname varchar(101) GENERATED ALWAYS AS (CONCAT(first_name, ' ',last_name)),  
  email VARCHAR(100) NOT NULL  
);
```

- Tipos de dados

# Tipos de Dados no MySQL

## → Dados Numéricos

- Integer Types (Exact Value) – INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

Data Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2 <sup>63</sup>	0	2 <sup>63</sup> -1	2 <sup>64</sup> -1

\*INT é sinónimo de INTEGER

# Tipos de Dados no MySQL

## → Dados Numéricos

- Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC

Os tipos DECIMAL e NUMERIC armazenam valores de dados numéricos exatos. Estes tipos de dados são usados quando é importante preservar a precisão exata, por exemplo, com dados monetários.

DECIMAL(n,m)

n – precisão – representa o número de dígitos significativos que são armazenados.

m – escala – representa o número de dígitos que podem ser armazenados após o ponto decimal.

**Exemplo:** 105,98€ -> DECIMAL (5,2)

\* DEC e FIXED são sinónimos de DECIMAL

# Tipos de Dados no MySQL

## → Dados Numéricos

- Floating-Point Types (Approximate Value) - FLOAT, DOUBLE

Os tipos FLOAT e DOUBLE representam valores de dados numéricos aproximados. O MySQL usa quatro bytes para valores de precisão simples e oito bytes para valores de precisão dupla.

Types	Description
FLOAT	A precision from 0 to 23 results in a four-byte single-precision FLOAT column.
DOUBLE	A precision from 24 to 53 results in an eight-byte double-precision DOUBLE column.

- Bit Value Type - BIT

O tipo de dados BIT é usado para armazenar valores de bits. Um tipo de BIT(M) permite o armazenamento de valores de M-bit. M pode variar de 1 a 64.

# Tipos de Dados no MySQL

## → Dados Alfanuméricos

**VARCHAR** (strings de tamanho variável) vs. **CHAR** (strings de tamanho fixo)

- Os tipos CHAR e VARCHAR são declarados com um comprimento que indica o número máximo de caracteres que o utilizador deseja armazenar;
- Os dados do tipo CHAR são preenchidos à direita com espaços em branco para o comprimento especificado.

Valor	CHAR(4)		VARCHAR(4)	
"	'____'	4 bytes	"	1 byte
'AB'	'AB__'	4 bytes	'AB'	3 bytes
'ABC'	'ABC_'	4 bytes	'ABC'	4 bytes
'ABCD'	'ABCD'	4 bytes	'ABCD'	5 bytes

O VARCHAR usa 1 ou 2 bytes de memória adicionais para tamanho ou para marcar o fim dos dados. 

\* NCHAR e NVARCHAR são semelhantes a CHAR e VARCHAR a diferença é que armazenam dados no formato Unicode.

# Tipos de Dados no MySQL

## → Dados Alfanuméricos

- Os tipos BINARY e VARBINARY são semelhantes a CHAR e VARCHAR, exceto que eles armazenam strings binárias em vez de strings não binárias. Ou seja, eles armazenam cadeias de bytes em vez de cadeias de caracteres.
- Para armazenar textos mais longos:
  - TEXT
  - TINYTEXT
  - MEDIUMTEXT
  - LONGTEXT

# Tipos de Dados no MySQL

## → Dados Alfanuméricos

- O tipo **ENUM** é um objeto de string cujo valor é seleccionado a partir de um conjunto de valores permitidos que são definidos explicitamente no momento de criação da coluna.

### EXEMPLO:

```
CREATE TABLE urgencia_pulseiras (  
id INT AUTO_INCREMENT,  
cor VARCHAR(45) NOT NULL,  
prioridade ENUM('Não Urgente', 'Pouco Urgente', 'Urgente', 'Muito Urgente', 'Emergente') NOT NULL );
```

A coluna prioridade aceitará apenas a inserção de um dos cinco valores definidos. O MySQL mapeia cada membro de enumeração para um índice numérico. Neste caso, 'Não Urgente', 'Pouco Urgente', 'Urgente', 'Muito Urgente' e 'Emergente' são mapeados para 1, 2, 3, 4 e 5 respectivamente.

# Tipos de Dados no MySQL

## → Dados Alfanuméricos

- O tipo **SET** é um objeto string que pode ter zero ou mais valores, cada um dos quais deve ser escolhido a partir de um conjunto de valores especificados quando a tabela é criada.

### EXEMPLO:

```
CREATE TABLE categorias(  
id INT AUTO_INCREMENT,  
nome VARCHAR(45) NOT NULL,  
tipo SET('A', 'B') NOT NULL );
```

A coluna tipo aceitará a inserção de "", 'A', 'B' ou 'A,B'. O MySQL armazena valores SET numericamente, com o bit de ordem inferior do valor armazenado correspondendo ao primeiro membro do conjunto.

Os tipos de dados SET e ENUM funcionam de maneira semelhante, exceto que o tipo de dados ENUM pode conter apenas um único membro da lista predefinida de valores, enquanto o tipo de dados SET permite armazenar zero ou qualquer número de valores juntos.

# Tipos de Dados no MySQL

## → Dados de Data/Hora

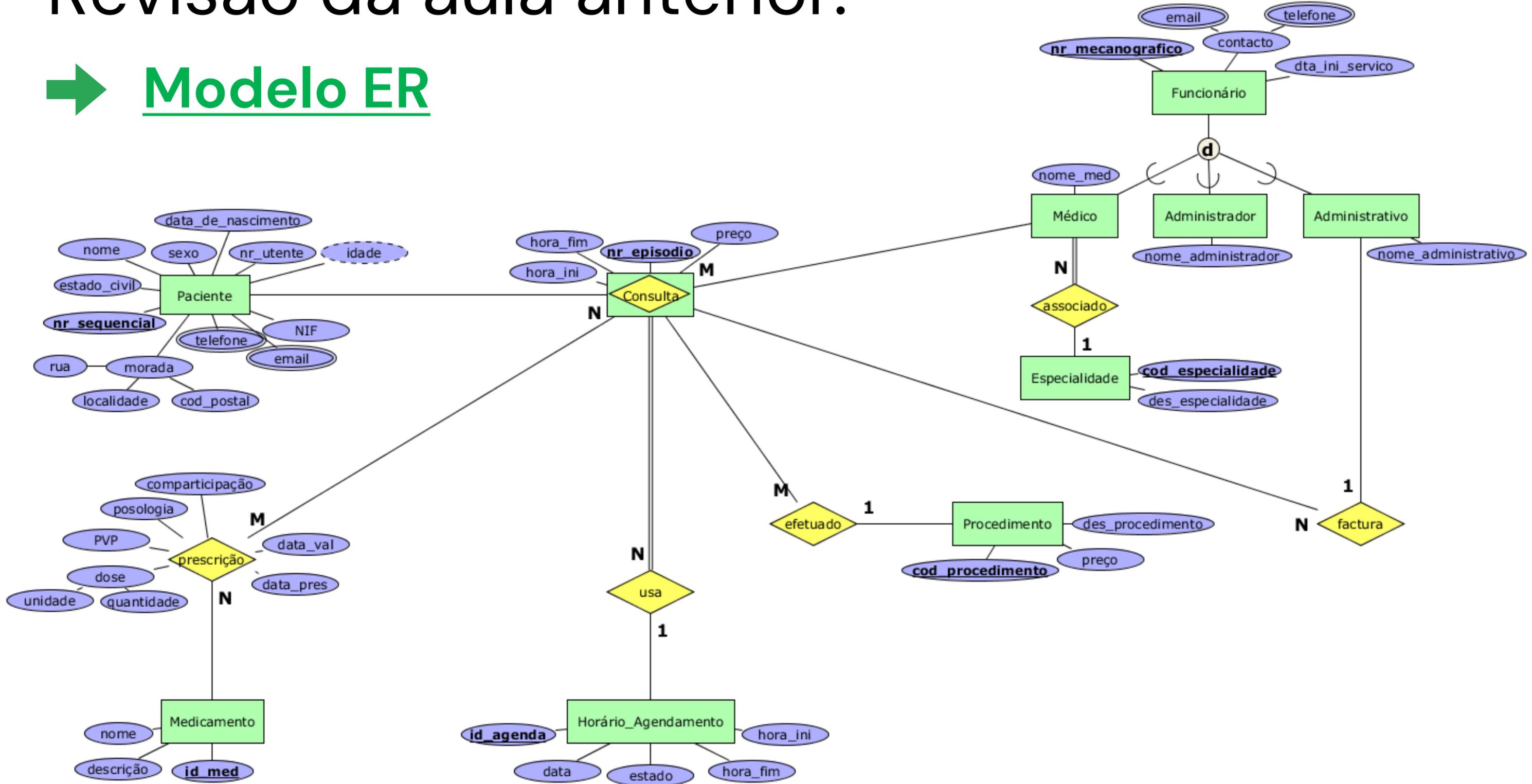
Tipo de Dados	Notação
<u>DATE</u>	YYYY-MM-DD
<u>TIME</u>	hh:mm:ss
<u>DATETIME</u> *	YYYY-MM-DD hh:mm:ss
<u>TIMESTAMP</u> **	YYYY-MM-DD hh:mm:ss
<u>YEAR</u>	YYYY

\* O intervalo suportado varia de '1000-01-01 00:00:00' a '9999-12-31 23:59:59'.

\*\* O intervalo suportado varia de '1970-01-01 00:00:01' a '2038-01-19 03:14:07'.

# Revisão da aula anterior:

## ➔ Modelo ER

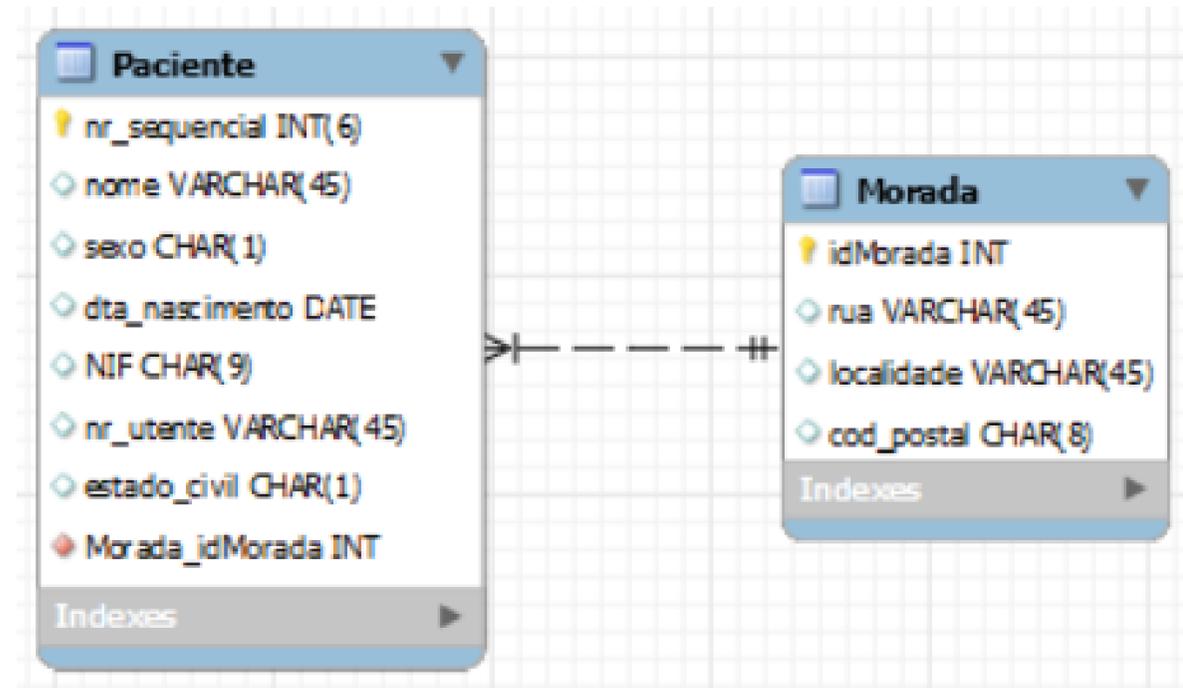


# Revisão da aula anterior:

## → Modelo Relacional



ou



# Revisão da aula anterior:

## → Modelo Relacional

Para lidarmos com as restrições de domínio no caso do sexo e do estado civil temos 3 opções possíveis:

### **A) Definir a coluna com o tipo ENUM;**

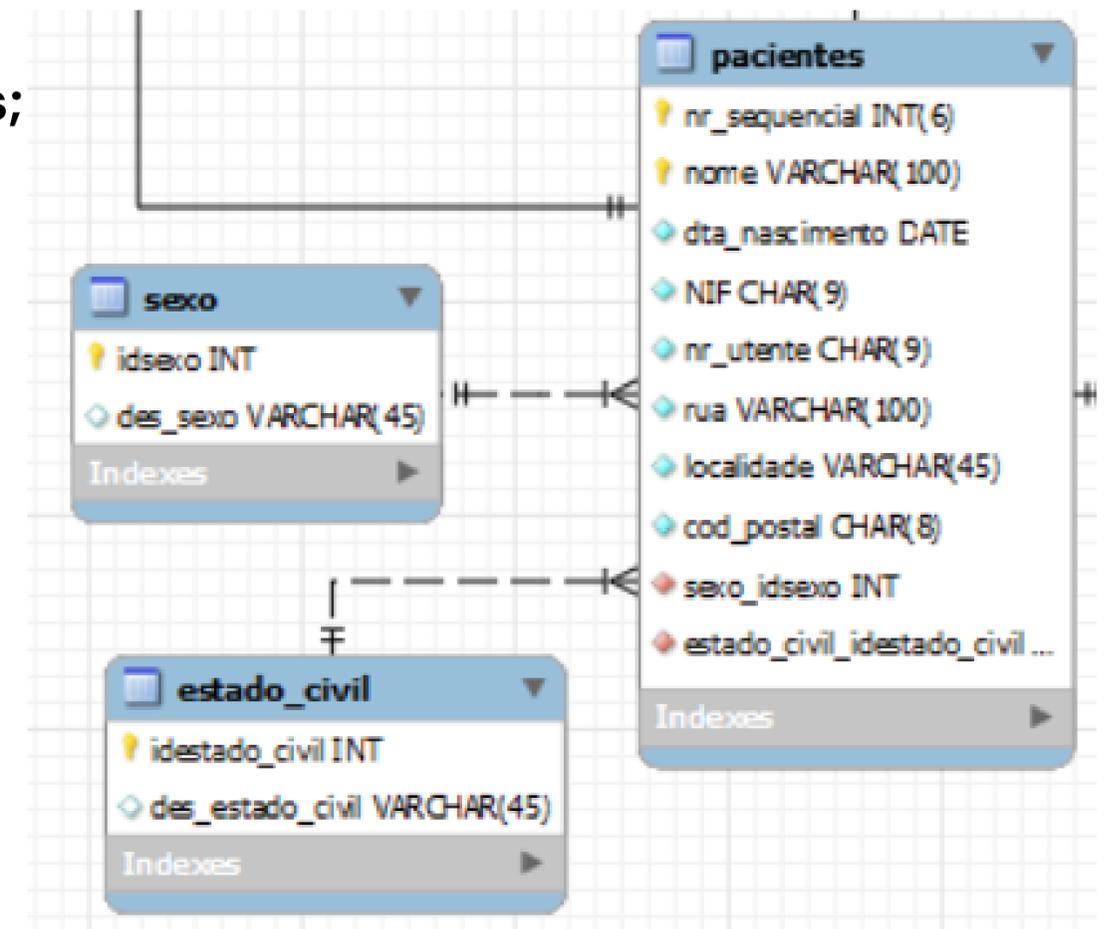
```
CREATE TABLE pacientes (  
  nr_sequencial INT NOT NULL AUTO_INCREMENT,  
  ...  
  sexo ENUM('F', 'M', 'I') NOT NULL,  
  estado_civil ENUM('S', 'C', 'D', 'V') NOT NULL,  
  ...  
);
```

# Revisão da aula anterior:

## ➔ Modelo Relacional

Para lidarmos com as restrições de domínio no caso do sexo e do estado civil temos 3 opções possíveis:

**B) Criar uma tabela à parte para definir as opções possíveis;**



# Revisão da aula anterior:

## → Modelo Relacional

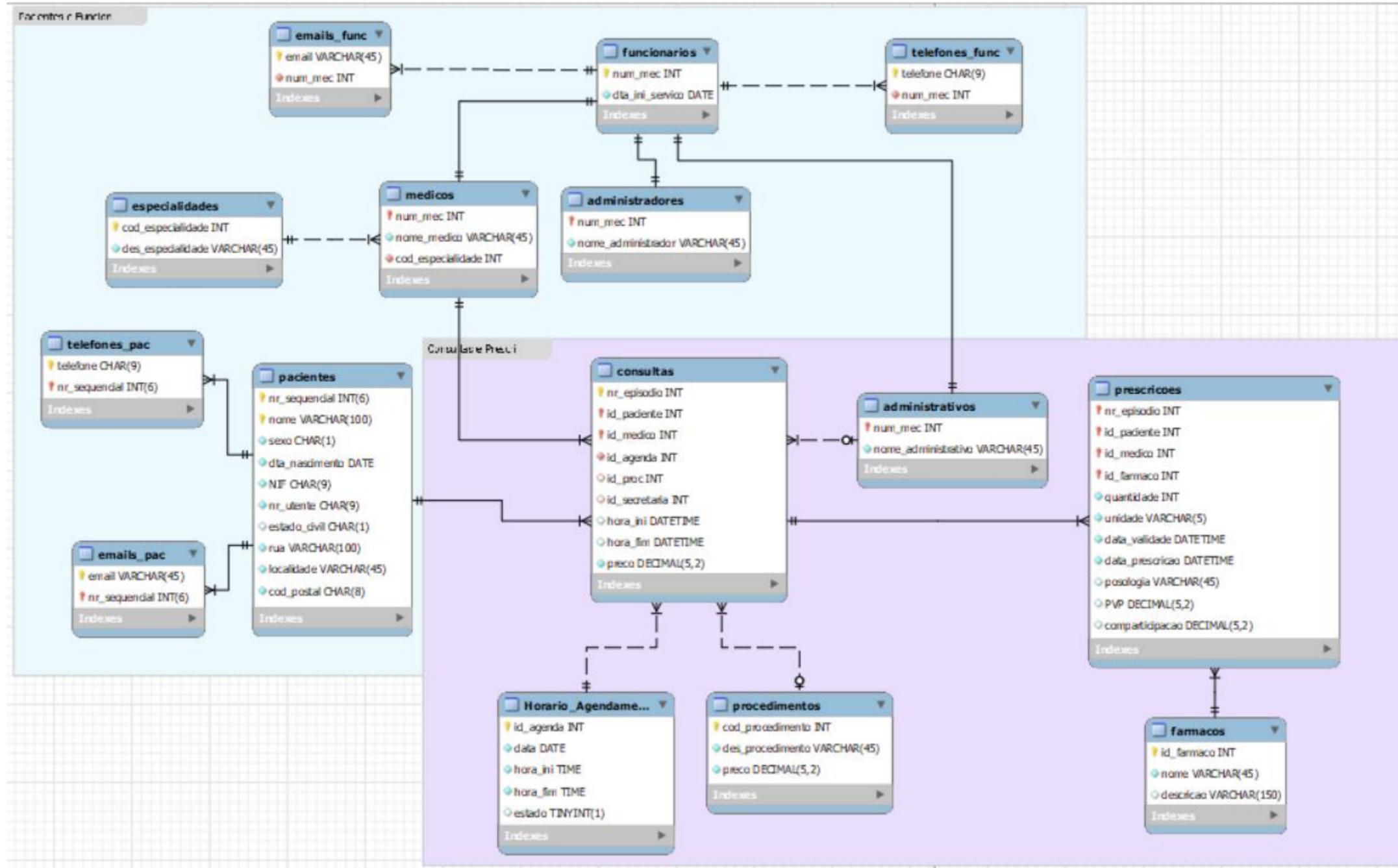
Para lidarmos com as restrições de domínio no caso do sexo e do estado civil temos 3 opções possíveis:

### **C) Definir a coluna com o tipo CHAR(1) e aplicar *check constraints*;**

```
CREATE TABLE pacientes (  
  nr_sequencial INT NOT NULL AUTO_INCREMENT,  
  ...  
  sexo CHAR(1) NOT NULL,  
  estado_civil CHAR(1) NULL,  
  CONSTRAINT chk_sexo  
    CHECK(sexo = 'F' OR sexo = 'M' OR sexo = 'I')  
  CONSTRAINT chk_estado_civil  
    CHECK(estado_civil IN ('S', 'C', 'D', 'V'))  
);
```

# Revisão da aula anterior:

## ➔ Modelo Relacional



# Material p/ a aula

## MySQL Community Server

### Windows

- <https://dev.mysql.com/downloads/installer/>
- <https://dev.mysql.com/doc/mysql-installation-excerpt/5.7/en/windows-installation.html>

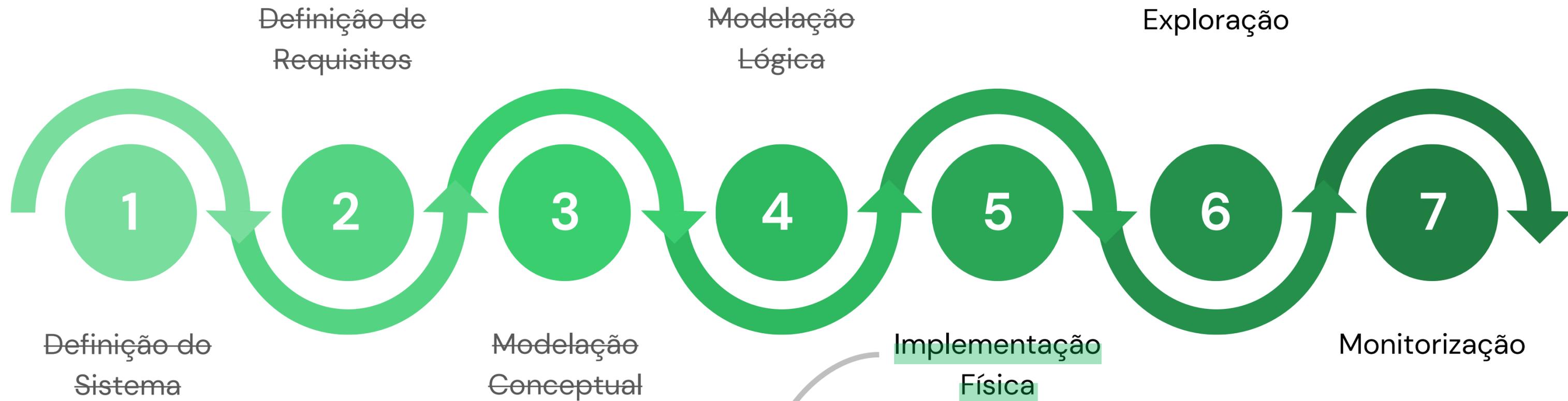
### Linux

- <https://dev.mysql.com/doc/refman/8.0/en/linux-installation.html>

### MacOS

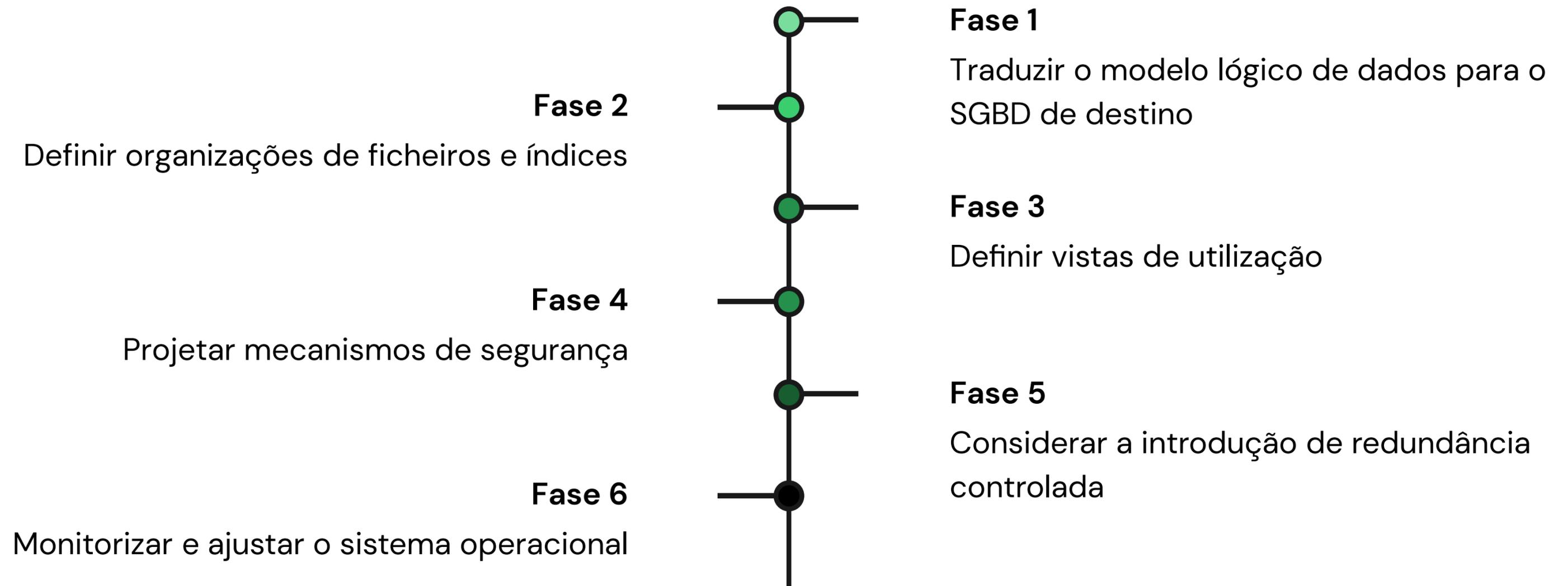
- <https://dev.mysql.com/downloads/mysql/>
- <https://dev.mysql.com/doc/refman/8.0/en/macos-installation.html>

# Ciclo de vida de um SBD



Decidir como traduzir o projeto de base de dados lógico (ou seja, as entidades, atributos, relacionamentos e restrições) num projeto de base de dados físico que pode ser implementado usando o SGBD de destino.

# Ciclo de vida de um SBD: Modelação Lógica



# FASE 5: Modelação Física

## → Traduzir o modelo lógico para o SGBD de destino

A primeira fase do projeto de BD físico envolve a **tradução** das relações no modelo de dados lógico num formato que possa ser implementado no SGBD relacional de destino.

Esta fase divide-se em:

Fase 1.1 – Representar relações de base

Fase 1.2 – Representar os dados derivados

Fase 1.3 – Representar restrições gerais

# FASE 5: Modelação Física

## ➔ Traduzir o modelo lógico para o SGBD de destino

### Fase 1.1 – Representar relações de base

Para cada relação identificada no modelo de dados lógico, deve criar uma definição que consiste em:

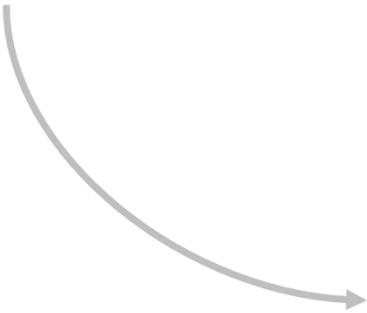
- o nome da relação;
- uma lista de atributos simples entre parêntesis;
- a chave primária e, quando apropriado, chaves candidatas e chaves estrangeiras;
- restrições de integridade referencial para quaisquer chaves estrangeiras identificadas;
- o domínio de cada atributo, consistindo num tipo de dados, comprimento e quaisquer restrições no domínio;
- um valor padrão opcional para o atributo;
- se o atributo pode conter nulos;
- se o atributo é derivado e, em caso afirmativo, como deve ser calculado.

# FASE 5: Modelação Física

## → Data Definition Language (DDL)

**DDL (Data Definition Language)** – Linguagem usada para especificar a informação acerca de cada relação, incluindo:

- O esquema de cada relação;
- O domínio de valores associados a cada atributo;
- As restrições de integridade;
- O conjunto de índices a manter para cada relação;
- As informações de segurança e autorização para cada relação;
- As estruturas de armazenamento físico em disco de cada relação.



CREATE  
ALTER  
DROP  
TRUNCATE  
COMMENT  
RENAME

# FASE 5: Modelação Física

## ➔ Data Definition Language (DDL)

- Instruções para criar/apagar uma base de dados:

→ cria uma base de dados física

```
CREATE DATABASE/SCHEMA [IF NOT EXISTS] <nome_BD>
```

```
[CHARACTER SET charset_name]
```

```
[COLLATE collation_name];
```

\* Se não incluirmos as cláusulas CHARACTER SET e COLLATE, o MySQL usará os valores default/padrão.

→ para consultar os valores suportados podemos executar a instrução:

```
SHOW CHARACTER SET;
```

→ listar as base de dados

```
SHOW DATABASES;
```

→ identificação da área de trabalho

```
USE <nome_BD>;
```

# FASE 5: Modelação Física

## ➔ Data Definition Language (DDL)

- Instruções para criar e apagar uma base de dados:

→ altera a base de dados com o nome especificado

```
ALTER {DATABASE | SCHEMA} <nome_BD>
```

```
alter_option: {
```

```
    [DEFAULT] CHARACTER SET [=] <charset_name>
```

```
    | [DEFAULT] COLLATE [=] <collation_name>
```

```
    | [DEFAULT] ENCRYPTION [=] {'Y' | 'N'}
```

```
    | READ ONLY [=] {DEFAULT | 0 | 1}
```

```
}
```

→ apaga a base de dados com o nome especificado

```
DROP {DATABASE | SCHEMA} [IF EXISTS] <nome_BD>
```

# FASE 5: Modelação Física

## ➔ Data Definition Language (DDL)

- Instruções para criar e apagar uma tabela:

→ cria uma tabela com o nome escolhido e com as colunas especificadas

```
CREATE TABLE [IF NOT EXISTS ] <nome_tabela> (
<nome_coluna> <tipo_coluna[tamanho]> [NOT NULL | NULL] [DEFAULT <value>] [AUTO_INCREMENT][UNIQUE],
...,
PRIMARY KEY (<nome_coluna_PK>,...)
[CONSTRAINT <constraint_name>] UNIQUE {KEY | INDEX} (<nome_coluna>,...)
[CONSTRAINT <constraint_name>] FOREIGN KEY (<nome_coluna_FK>) REFERENCES <nome_tabela_FK> (<nome_coluna_FK>)
[ON UPDATE <referential_integrity_constraint>] [ON DELETE <referential_integrity_constraint>]
) [ENGINE=<storage_engine>];
```

Se o ENGINE não for declarado, o MySQL usará o InnoDB por padrão.

referential\_integrity\_constraint = {NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT } Se não especificar a cláusula ON DELETE e ON UPDATE, o MySQL usará RESTRICTED por padrão.

# FASE 5: Modelação Física

## → Data Definition Language (DDL)

### Exemplo:

```
CREATE TABLE IF NOT EXISTS `Paciente` (  
  `nr_sequencial` INT(6) NOT NULL AUTO_INCREMENT,  
  `nome` VARCHAR(100) NOT NULL,  
  `sexo` CHAR(1) NOT NULL,  
  `dta_nascimento` DATE NOT NULL,  
  `NIF` CHAR(9) NOT NULL UNIQUE,  
  `nr_utente` CHAR(9) NOT NULL UNIQUE,  
  `estado_civil` CHAR(1) NULL,  
  `rua` VARCHAR(100) NULL,  
  `localidade` VARCHAR(45) NULL,  
  `cod_postal` CHAR(8) NULL,  
  PRIMARY KEY (`nr_sequencial`)  
);
```

```
CREATE TABLE IF NOT EXISTS `Paciente` (  
  `nr_sequencial` INT(6) NOT NULL AUTO_INCREMENT,  
  `nome` VARCHAR(100) NOT NULL,  
  `sexo` CHAR(1) NOT NULL,  
  `dta_nascimento` DATE NOT NULL,  
  `NIF` CHAR(9) NOT NULL,  
  `nr_utente` CHAR(9) NOT NULL,  
  `estado_civil` CHAR(1) NULL,  
  `rua` VARCHAR(100) NULL,  
  `localidade` VARCHAR(45) NULL,  
  `cod_postal` CHAR(8) NULL,  
  PRIMARY KEY (`nr_sequencial`),  
  UNIQUE KEY (`NIF`),  
  UNIQUE KEY (`nr_utente`)  
);
```

# FASE 5: Modelação Física

## ➔ Data Definition Language (DDL)

- Instruções para criar e apagar uma tabela:

→ mostra informação sobre os elementos do esquema criado (metadados)

**{DESC | DESCRIBE} <nome\_tabela>;**

→ mostra a definição da tabela com o nome especificado

**SHOW COLUMNS FROM <nome\_tabela>;**

→ mostra a instrução de criação da tabela com o nome especificado

**SHOW CREATE TABLE <nome\_tabela>;**

→ mostra a definição das chaves e dos índices de uma tabela

**SHOW KEYS FROM <nome\_tabela>;**

→ apaga a tabela com o nome especificado

**DROP TABLE <nome\_tabela> [RESTRICT | CASCADE];**

# FASE 5: Modelação Física

## ➔ Data Definition Language (DDL)

- Instruções para modificar uma tabela:

**ALTER TABLE <nome\_tabela\_antigo> RENAME TO <nome\_tabela\_novo>;** → altera o nome da tabela.

**ALTER TABLE <nome\_tabela> ADD <nome\_campo> <domínio\_campo>;** → cria um novo atributo na tabela com o nome e domínio especificados. Todos os tuplos existentes ficam com NULL no novo atributo.

**ALTER TABLE <nome\_tabela> DROP <nome\_campo>;** → apaga o atributo com o nome especificado da tabela.

**ALTER TABLE <nome\_tabela> MODIFY <nome\_campo> <domínio\_campo>;** → modifica o atributo com o nome especificado da tabela.

**ALTER TABLE <nome\_tabela> ALTER < nome\_campo> SET DEFAULT <value>;** → modifica uma coluna de uma tabela para lhe atribuir valores padrão.

**ALTER TABLE <nome\_tabela> ALTER < nome\_campo> DROP DEFAULT;** → remove os valores padrão de uma coluna de uma tabela.

**ALTER TABLE <nome\_tabela> ADD CONSTRAINT <nome\_constraint> UNIQUE KEY(column\_1,column\_2,...);** → modifica uma tabela para lhe atribuir uma indexação única .

# FASE 5: Modelação Física

## → Traduzir o modelo lógico para o SGBD de destino

### Fase 1.2 – Representar os dados derivados

Frequentemente, os atributos derivados não aparecem no modelo de dados lógico, mas são documentados no dicionário de dados. A decisão entre armazenar um atributo derivado na BD ou calculá-lo sempre que for necessário, deve ter em consideração:

- o custo adicional para armazenar os dados derivados e mantê-los consistentes com os dados operacionais dos quais são derivados;
- o custo para calculá-lo cada vez que for necessário.

No caso de estudo do Hospital Portucalense, apenas existe a **idade** como atributo derivado. Se decidíssemos armazenar o atributo na BD, este precisaria de ser atualizado sempre que um paciente do hospital fizesse anos. Por outro lado, se o atributo não for armazenado diretamente na relação Paciente, deve ser calculado cada vez que for necessário. Para isso podemos desenvolver uma função.

# FASE 5: Modelação Física

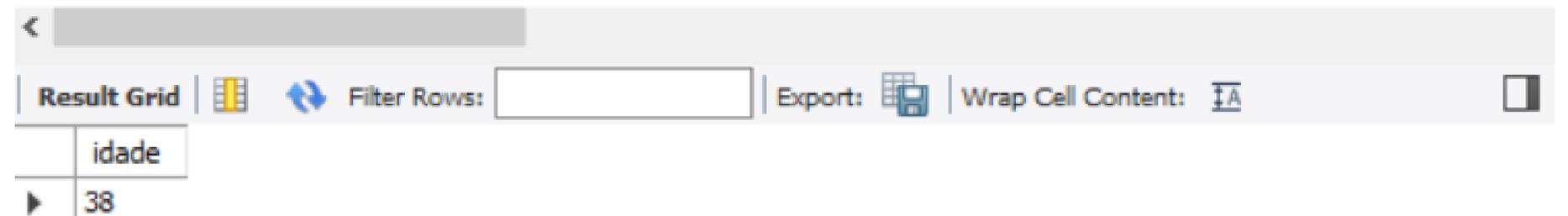
## ➔ Data Definition Language (DDL)

### Fase 1.2 – Representar os dados derivados

- Instrução para criar uma função na tabela:

```
CREATE FUNCTION Idade (dta date)
RETURNS INT
DETERMINISTIC
BEGIN
    RETURN TIMESTAMPDIFF(YEAR, dta, CURDATE());
END $$
DELIMITER ;
```

```
1 SELECT idade('1983-10-12') as idade;
```



The screenshot shows a database interface with a toolbar and a result grid. The toolbar includes a back arrow, a 'Result Grid' button, a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' button. The result grid displays a single row with the column name 'idade' and the value '38'.

idade
38

# FASE 5: Modelação Física

## ➔ Traduzir o modelo lógico para o SGBD de destino

### Fase 1.3 – Representar restrições gerais

As atualizações das relações podem ser limitadas por restrições de integridade que governam as transações do “mundo real”. Na Etapa 1.1, projetamos várias restrições de integridade: dados necessários, restrições de domínio e integridade de entidade e referencial. Nesta etapa, é necessário considerar as restrições gerais restantes.

**Exemplo:** Uma receita não pode conter mais do que 5 fármacos.

- Instruções para criar regras de negócio/restrições gerais:

**CONSTRAINT MaximoMeds**

```
CHECK (NOT EXISTS (SELECT nr_episodio, id_paciente, id_medico, id_farmaco
FROM prescicoes GROUP BY nr_episodio, id_paciente, id_medico, id_farmaco
HAVING COUNT(*) > 5))
```

# FASE 5: Modelação Física

## ➔ Definir organizações de ficheiros e índices

- Um índice é uma estrutura de dados que melhora a velocidade de recuperação dos dados de uma tabela. Os índices podem ser usados para localizar dados rapidamente sem precisar de “varrer” cada linha de uma tabela para uma determinada consulta.
- Quando se cria uma tabela com uma chave primária ou chave candidata (Unique Constraint), o MySQL cria automaticamente um índice chamado PRIMARY.
- Por padrão, o MySQL cria o índice B-Tree se este não for especificado. Os tipos de índices permitidos variam com base no mecanismo de armazenamento da tabela:

Storage Engine	Allowed Index Types
InnoDB	BTREE
MyISAM	BTREE
MEMORY/HEAP	HASH, BTREE

# FASE 5: Modelação Física

## ➔ Data Definition Language (DDL)

- Instruções para criar/apagar índices:

→ cria índice

```
CREATE [UNIQUE] INDEX <nome_índice>  
ON <nome_tabela> (<nome_campo> [ASC | DESC],...);
```

→ mostra os índices de uma tabela

```
SHOW INDEXES FROM <nome_tabela>;
```

→ mostra **todos os índices criados sobre as tabelas de uma base de dados**

```
INFORMATION_SCHEMA.STATISTICS  
SELECT DISTINCT TABLE_NAME, INDEX_NAME  
FROM INFORMATION_SCHEMA.STATISTICS  
WHERE TABLE_SCHEMA = <nome_tabela>;
```

→ apaga índice

```
DROP INDEX <nome_índice> ON <nome_tabela>
```

# FASE 5: Modelação Física

## ➔ MySQL Command Line Client

O MySQL fornece um shell interativa para criar tabelas, inserir dados, etc.

- **show databases/tables** – mostra as bases de dados/tabelas disponíveis;
- **use/connect <nome\_bd>** – estabelece a ligação a uma base de dados;
- **select database()** – mostra a base de dados atualmente selecionada;
- **describe <nome\_tabela>** – mostra a estrutura interna de uma tabela;
- **create database <nome\_bd>** – cria uma nova base de dados;
- **source <nome\_ficheiro>** – carrega um ficheiro e tenta executar os comandos SQL nele contidos;
- **\c** – cancela um comando
- **quit/exit** – sai do interpretador

# FASE 5: Modelação Física

## ➔ Data Manipulation Language (DML)

Existem 4 instruções básicas para a manipulação de dados:

- INSERT → para inserir dados na BD;

```
INSERT INTO <nome_tabela> (<c1>,<c2>,...) VALUES (<v1>,<v2>,...);
```

```
INSERT INTO <nome_tabela> (<c1>,<c2>,...)
```

```
VALUES
```

```
  (<v11>,<v12>,...),
```

```
  ...
```

```
  (<vnn>,<vn2>,...);
```

- DELETE → para remover dados da BD;

```
DELETE FROM <nome_tabela> WHERE <condição>;
```

- SELECT → para consultar dados da BD;

```
SELECT [DISTINCT] {* | <nome_c1>, ...}
```

```
FROM <nome_tabela>,...
```

```
[WHERE <condição>]
```

```
[ORDER BY <c1> [ASC | DESC], ...];
```

- UPDATE → para atualizar dados da BD;

```
UPDATE <nome_tabela>
```

```
SET
```

```
  <c1> = <v1>,
```

```
  <c2> = <v2>,
```

```
  ...
```

```
[WHERE <condição>];
```

# FASE 5: Modelação Física

## ➔ Data Manipulation Language (DML)

### OPERADORES:

AND – para condições conjuntas.

OR – para condições disjuntas.

IN – para determinar se um valor especificado corresponde a qualquer valor de uma lista de valores.

BETWEEN – para determinar se um valor está contido num intervalo de valores.

LIKE – para consultar dados com base num padrão especificado.

LIMIT – para limitar o número de instâncias retornadas.

# Próxima aula: Modelação Conceptual

