EVOLUÇÃO DE BANCOS DE DADOS

Sergio Mergen

Sumário

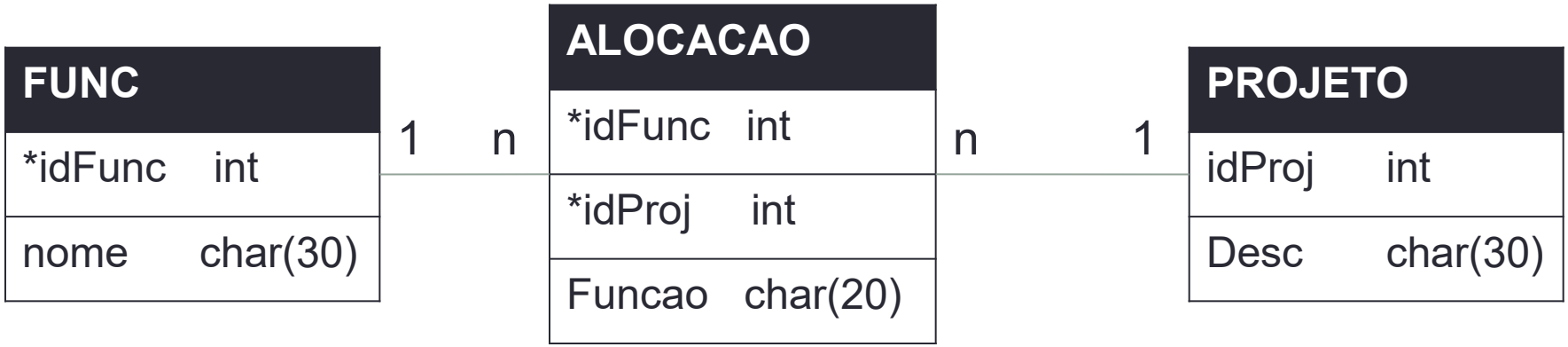
- Evolução de bancos de dados
- Modelagem Defensiva
- Controle da evolução
  - Boas práticas gerais
  - Encapsular acesso aos dados
  - Uso de ORMs

Evolução de bancos de dados

- Um modelo de banco de dados pode evoluir ao longo do tempo
  - Normalmente devido a presença de novos requisitos funcionais que usam dados ainda não presentes no modelo

Evolução de bancos de dados

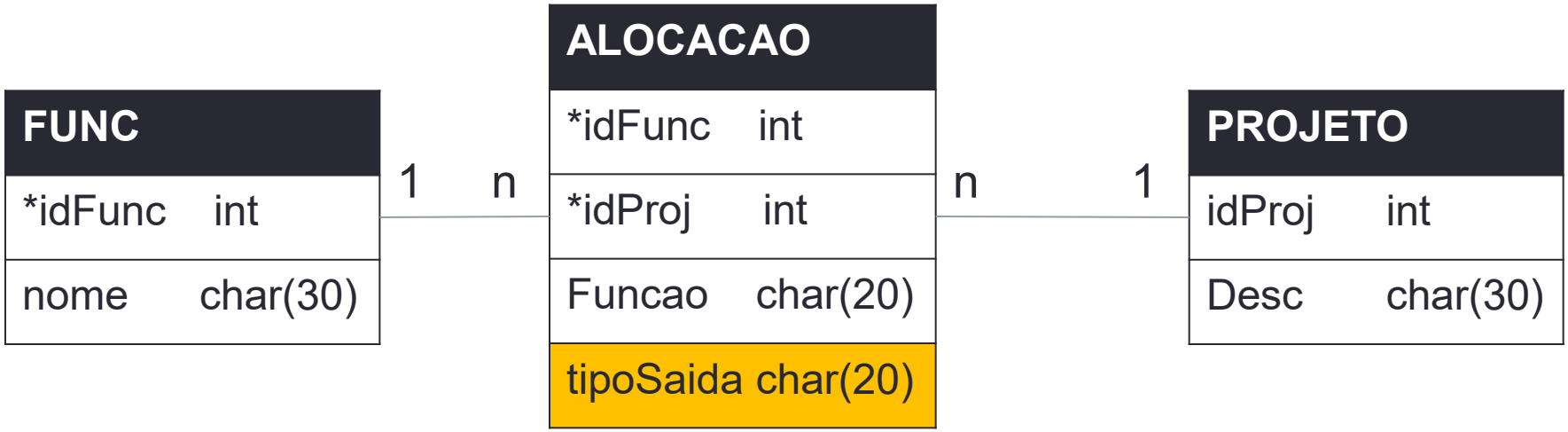
- Ex. funcionários que saíram de mais do que 3 projetos por motivo de improbidade devem ser exonerados
- Esse requisito não pode ser implementado no modelo atual
  - O motivo da saída não foi modelado



Evolução de bancos de dados

- Ex. funcionários que saíram de mais do que 3 projetos por motivo de improbidade devem ser exonerados

- Com a adição de um atributo o requisito pode ser implementado

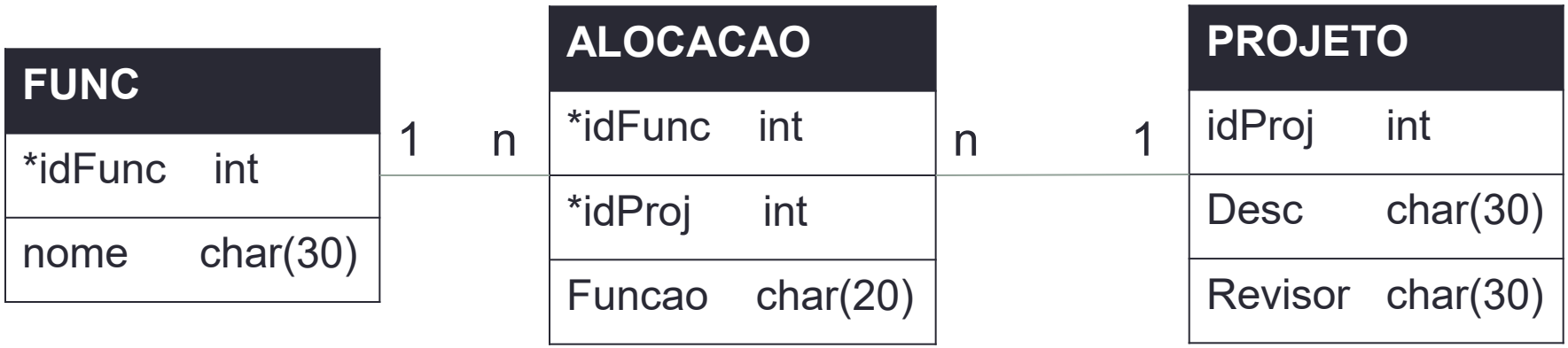


Evolução de bancos de dados

- No exemplo anterior, a mudança no esquema é fácil de ser realizada
  - Um atributo é adicionado
  - Registros já existentes podem receber valor nulo
    - Ou um valor padrão. Ex. “desconhecido”
- No entanto, em alguns casos a mudança leva a alterações mais profundas no esquema
  - Levando a um processo mais complexo de atualização dos dados já existentes

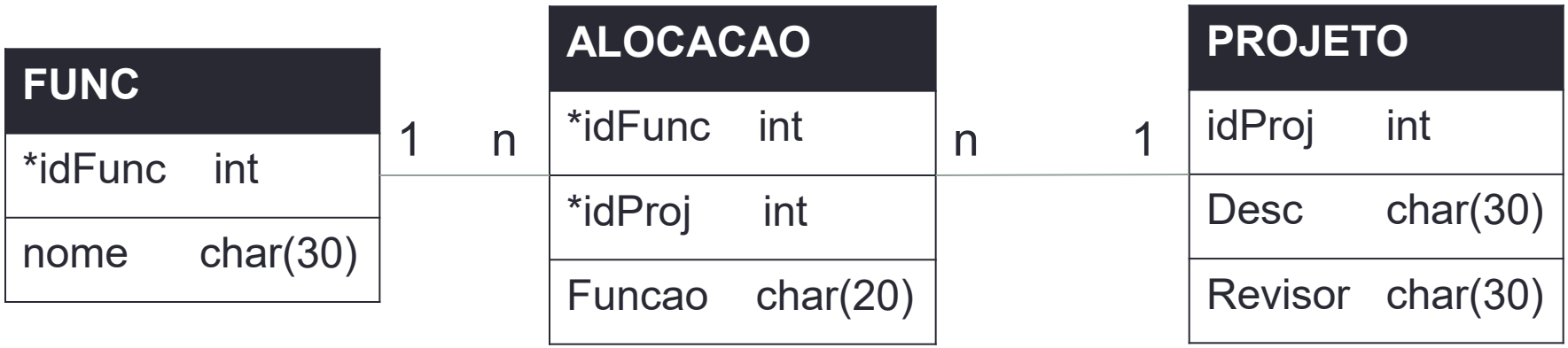
Evolução de bancos de dados

- Ex.
- Como era antes
  - Todo projeto possui apenas um revisor
  - O revisor podia ser qualquer pessoa, mesmo uma que não possuísse vínculo algum com a empresa
  - Na prática, apenas funcionários da empresa eram eventuais revisores



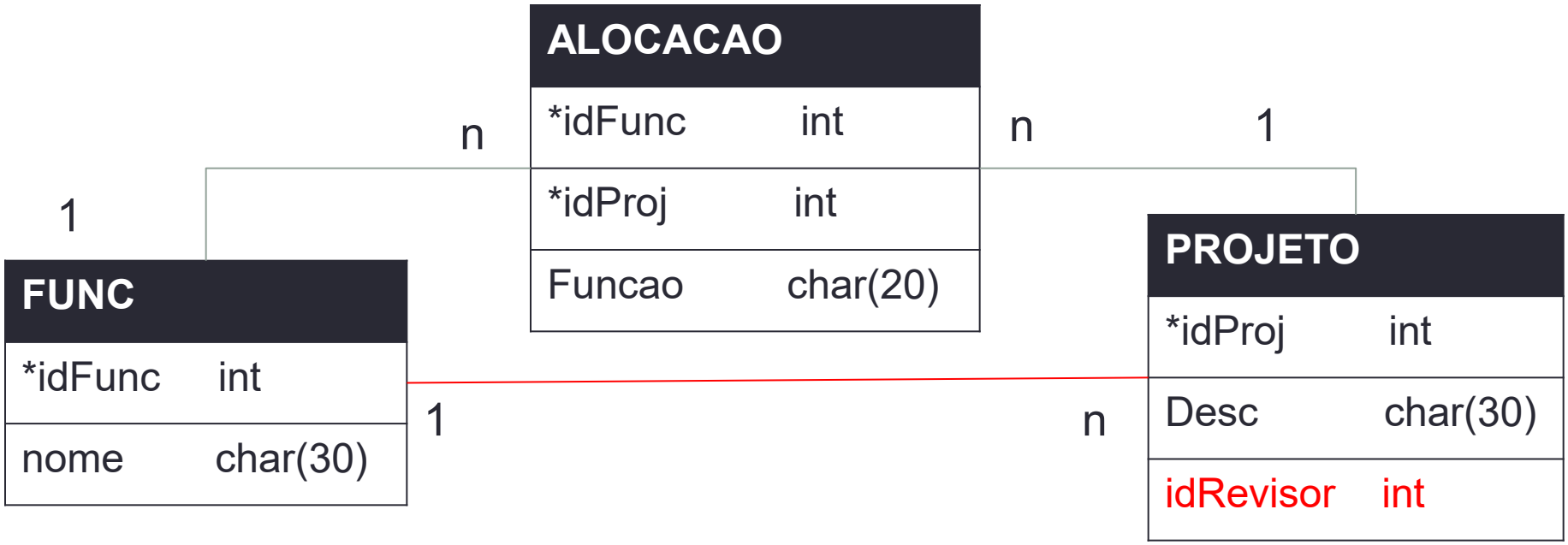
Evolução de bancos de dados

- Ex.
- Como passou a ser
  - Todo projeto possui apenas um revisor
  - O revisor deve ser um funcionário da empresa



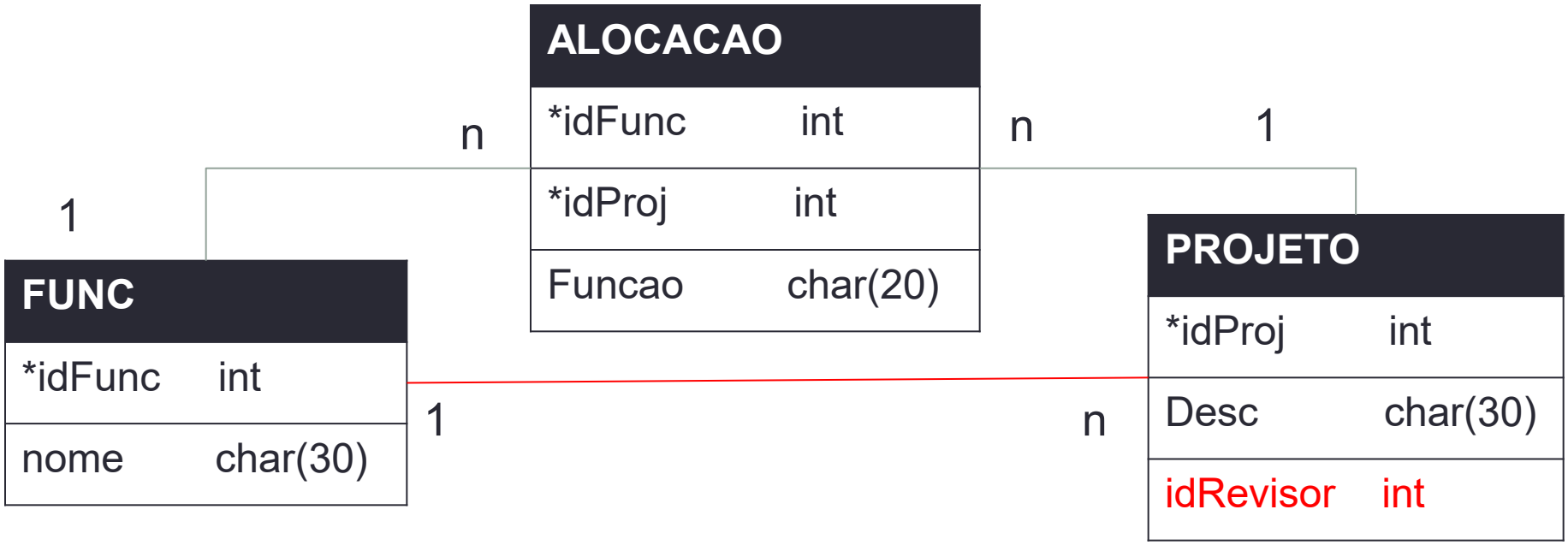
Evolução de bancos de dados

- Ex.
- Como passou a ser:
  - Todo projeto possui apenas um revisor
  - O revisor deve ser um funcionário da empresa



Evolução de bancos de dados

- Agora a alteração do esquema se tornou mais complexa
  - O que fazer com os registros já existentes?
  - Como migrar valores antigos do atributo legado “revisor”?



Evolução de bancos de dados

- Existem catálogos de processos relacionados a alteração de esquemas de bancos de dados já em produção
  - Veremos em outra aula

- Em alguns casos, vale a pena minimizar a necessidade de recorrer a um processo de alteração de esquema.

- Como?
  - Por meio de uma modelagem defensiva, que tem por objetivo deixar o modelo mais preparado para possíveis evoluções

Sumário

- Evolução de bancos de dados
- Modelagem Defensiva
- Controle da evolução
  - Boas práticas gerais
  - Encapsular acesso aos dados
  - Uso de ORMs

Modelagem Defensiva

- A modelagem defensiva depende muito do conhecimento dos analistas
  - Eles têm mais condições de prever as partes do modelo que podem sofrer atualizações no futuro
- De modo geral, uma dica bastante válida é analisar os atributos do modelo.
- Em muitos casos, o modelo fica mais estendível se os atributos forem transformados em tabelas

Modelagem Defensiva

- Situações que favorecem o uso de tabelas em vez de atributos
  - Quando o valor do atributo pode ser quebrado em valores menores
  - Quando outras informações podem complementar o sentido do valor do atributo

Modelagem Defensiva

- Quando o valor do atributo pode ser quebrado em valores menores

FUNC	
*idFunc	int
nome	char(50)
endereco	char(100)

antes

FUNC	
*idFunc	int
nome	char(50)

1

1

Endereco	
*idFunc	int
Rua	char(50)
Num	int
Cidade	char(40)

depois

Modelagem Defensiva

- Quando outras informações podem complementar o sentido do valor do atributo

FUNC	
*idFunc	int
nome	char(50)
dataContrato	date

antes

FUNC	
*idFunc	int
nome	char(50)

1

1

contrato	
*idFunc	int
dataContrato	date
tipoContrato	char(30)

depois

Modelagem Defensiva

- As duas possibilidades podem transformar uma tabela em duas tabelas com relacionamento 1-1
  - Nem sempre isso é desejável
  - Em alguns casos é melhor manter o atributo na tabela de origem
    - E adicionar novos atributos na mesma tabela
  - Falaremos sobre isso mais adiante

FUNC	
*idFunc	int
nome	char(50)
Rua	char(50)
Num	int
Cidade	char(50)

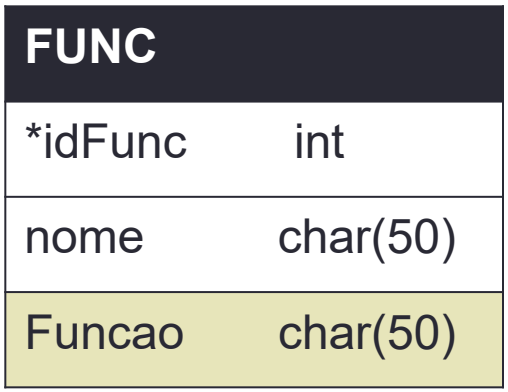
FUNC	
*idFunc	int
Nome	char(50)
dataContrato	date
tipoContrato	char(30)

Modelagem Defensiva

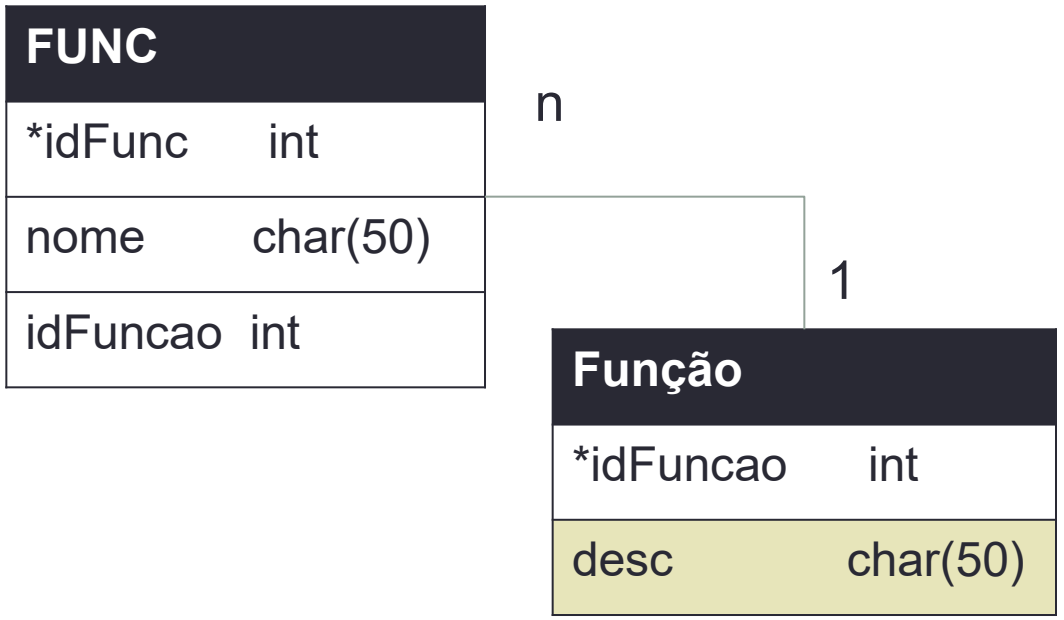
- Outras situações que favorecem o uso de tabelas em vez de atributos
  - Quando é desejável que se possa modificar o conjunto de valores permitido do atributo ao longo do tempo (domínio variável)
  - Quando um atributo pode ser multivalorado
  - Quando um valor de atributo pode estar associado a múltiplos registros
  - Quando é desejável guardar o histórico de mudanças do valor do atributo

Modelagem Defensiva

- Quando é desejável que se possa modificar o conjunto de valores permitido do atributo ao longo do tempo (domínio variável)



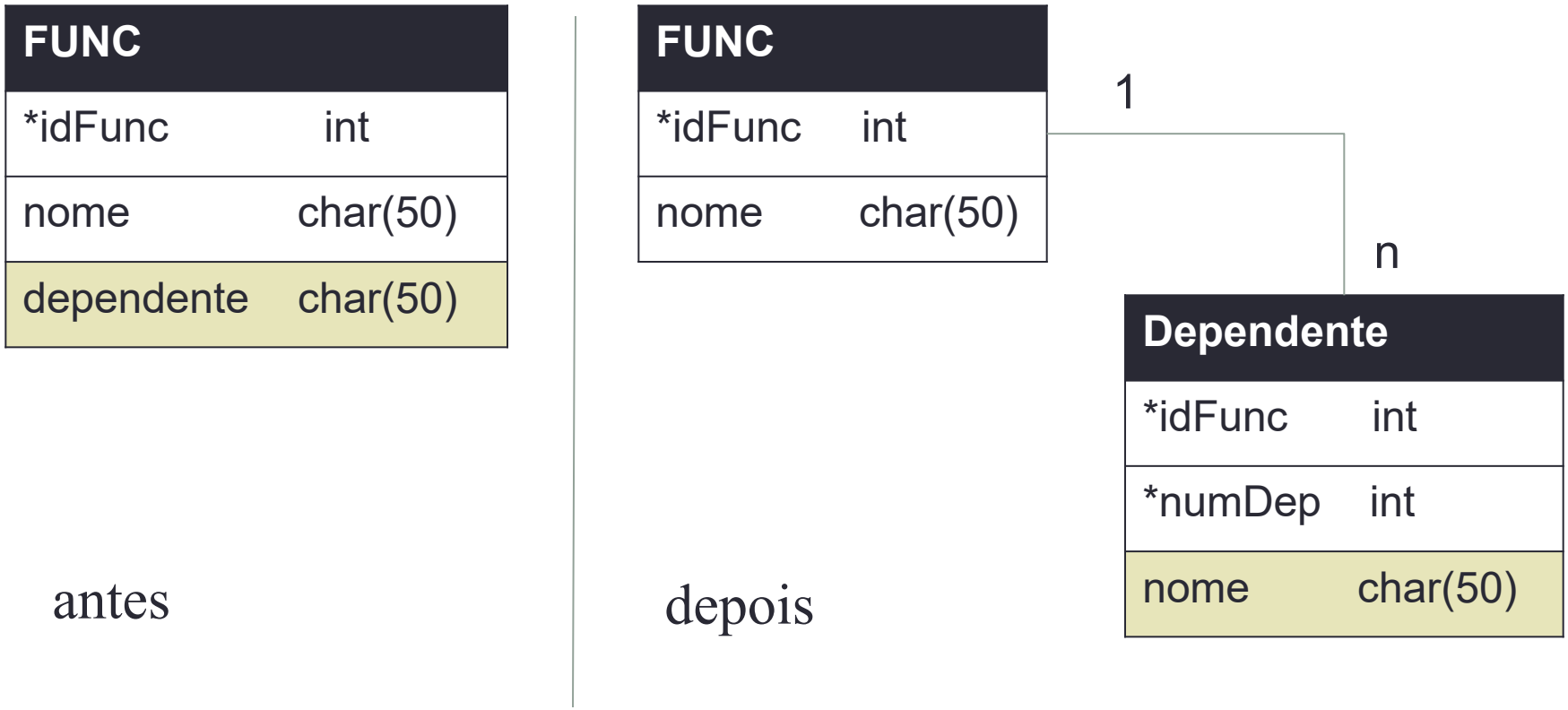
antes



depois

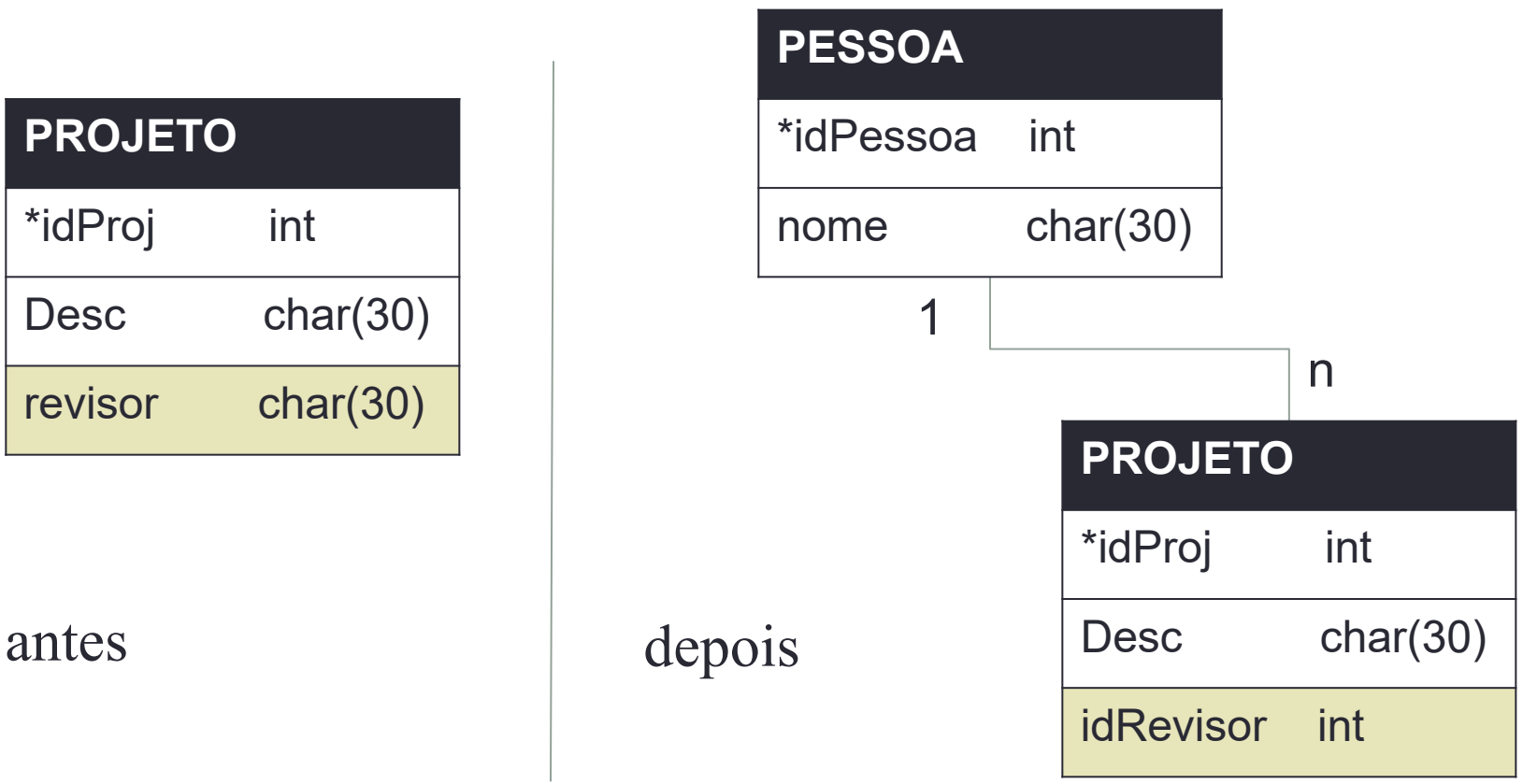
Modelagem Defensiva

- Quando um atributo pode ser multivalorado



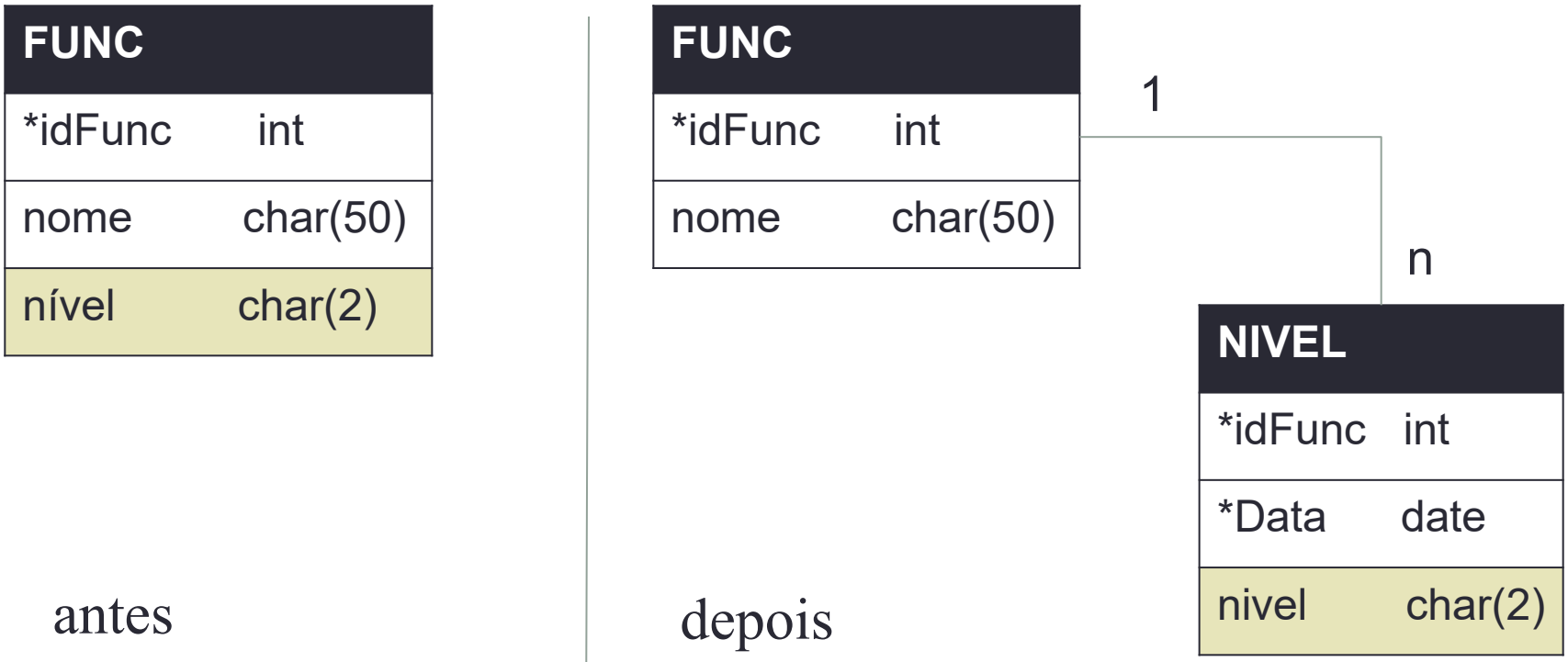
Modelagem Defensiva

- Quando um valor de atributo pode estar associado a múltiplos registros



Modelagem Defensiva

- Quando é desejável guardar o histórico de mudanças do valor do atributo



Modelagem Defensiva

- E se os valores de nível forem de domínio variável?

FUNC	
*idFunc	int
nome	char(50)
nível	char(2)

antes

FUNC	
*idFunc	int
nome	char(50)

1

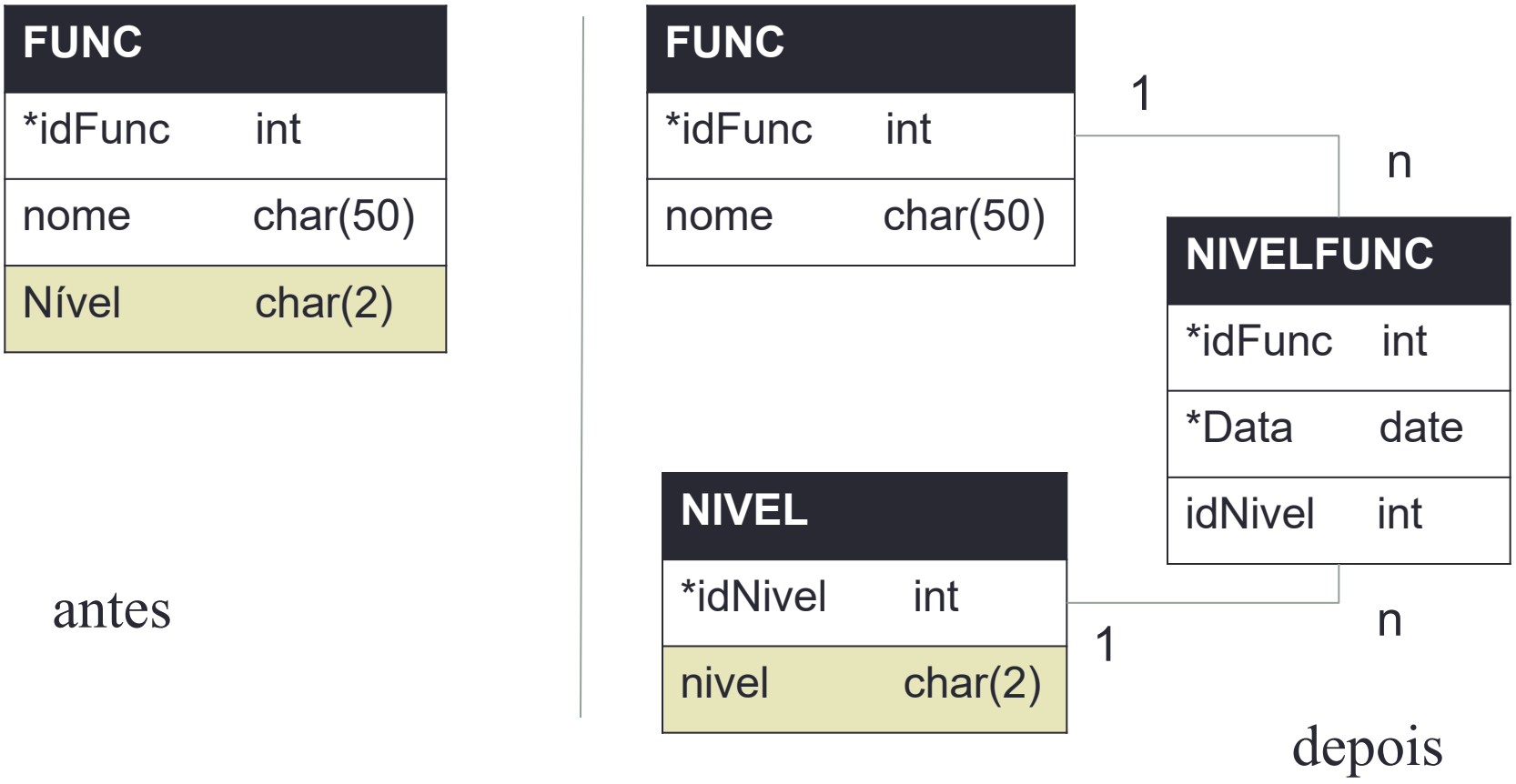
n

NIVEL	
*idFunc	int
*Data	date
nivel	char(2)

depois

Modelagem Defensiva

- E se os valores de nível forem de domínio variável?



Modelagem Defensiva

- De modo geral
  - Quanto mais tabelas
    - mais preparado o modelo fica para situações futuras
  - No entanto, isso não significa que todo atributo deva virar uma tabela
    - Isso encareceria bastante o custo de acesso aos dados.

Sumário

- Evolução de bancos de dados
- Modelagem Defensiva
- Controle da evolução
  - Boas práticas gerais
  - Encapsular acesso aos dados
  - Uso de ORMs

Controle da evolução

- Existem situações em que, mesmo com a modelagem defensiva, o esquema do banco precisa ser alterado

- Nesse caso, é recomendável seguir algumas boas práticas para que o processo de evolução ocorra sem transtornos

Sumário

- Evolução de bancos de dados
- Modelagem Defensiva
- Controle da evolução
  - Boas práticas gerais
  - Encapsular acesso aos dados
  - Uso de ORMs

Boas práticas gerais

- Separar ambiente de Desenvolvimento e Produção
  - Isso garante que testes e mudanças no código possam ser realizados com segurança antes de serem aplicados em produção.
- Containers Docker são bastante úteis nesse sentido
  - Os ambientes criados são isolados entre si
  - É fácil configurar um ambiente e recriá-lo em qualquer máquina
  - Uma vez definido o ambiente, é fácil subir um banco

Boas práticas gerais

- Versionar Scripts de Migração
  - Isso permite aplicar mudanças de forma incremental, rastreável e reversível.
- Existem ferramentas específicas para versionamento de SQL
  - Ex. Liquibase, Flyway
- Caso seja necessário desfazer alguma mudança no esquema, essas ferramentas executam scripts responsáveis por ajustar o esquema
  - A migração dos dados exige uma atenção especial

Sumário

- Evolução de bancos de dados
- Modelagem Defensiva
- Controle da evolução
  - Boas práticas gerais
  - Encapsular acesso aos dados
  - Uso de ORMs

Encapsular acesso aos dados

- Outra sugestão é encapsular o acesso ao banco de dados
  - Diversas tecnologias são úteis para essa finalidade
  - Ex.
    - Uso de Visões/Procedures
    - Uso de DAOs

Encapsular acesso aos dados

- O exemplo abaixo mostra uma procedure que devolve dados de projeto
  - As aplicações usam a procedure para obter dados de projeto em vez de uma consulta SQL
  - Caso o esquema da tabela proj mude, apenas o código da procedure precisa mudar

```
CREATE PROCEDURE listar_projetos ()
BEGIN
  SELECT
    p.nome AS nome,
    p.custo AS custo
  FROM proj p;
END;
```

Encapsular acesso aos dados

- Visões também podem ser usadas para a mesma finalidade, porém, são mais limitadas do que procedures
  - Não aceitam parâmetros
  - Não aceitam transações
  - Não aceitam comandos de atualização
- Por isso, procedures são mais indicadas como forma de abstrair o acesso aos dados

```
CREATE VIEW listar_projetos as (
  SELECT
    p.nome AS nome,
    p.custo AS custo
  FROM proj p
);
```

Encapsular acesso aos dados

- Um padrão de projeto que realiza essa encapsulamento é chamado de DAO (Data Acess Object)
  - Com DAO, caso o esquema do banco mude, apenas o DAO precisa ser atualizado
    - Desde que a mudança seja apenas uma refatoração, e não o acréscimo de informações

- Os próximos dois slides demonstram parte de um DAO referente a Projeto, definido com a linguagem Java

Encapsular acesso aos dados

```
public class ProjetoDAO {
    private Connection conexao;

    public ProjetoDAO(Connection conexao) {
        this.conexao = conexao;
    }

    public Projeto buscarPorId(int id) throws SQLException {
        String sql = "SELECT idProj, nome, custo FROM proj WHERE idProj = ?";
        try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
            stmt.setInt(1, id);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                Projeto p = new Projeto();
                p.setId(rs.getInt("idProj"));
                p.setNome(rs.getString("nome"));
                p.setCidade(rs.getInt(“custo"));
                return p;
            }
            return null;
        }
    }
```

Encapsular acesso aos dados

```
  .
  .
  .


public void salvar(Projeto p) throws SQLException {
        String sql = "INSERT INTO proj (nome, custo) VALUES (?, ?)";
        try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
            stmt.setString(1, p.getNome());
            stmt.setInt(2, p.getCusto());
            stmt.executeUpdate();
        }
    }

    // Outros métodos como deletar, atualizar etc.
}
```

Encapsular acesso aos dados

- Outros padrões de projeto que podem ser combinados com DAO
  - Repository: Permite acesso aos dados de modo a satisfazer necessidades do sistema modelado
    - Pode usar DAOs para acessar os dados, e organizá-los de modo a satisfazer algum requisito funcional
  - MVC: Separa a aplicação em três camadas (Model, View e Controller)
    - Usado para que a visualização possa ser adaptada sem que as demais partes do código sejam afetadas
    - Em uma arquitetura MVC, o Repository, ou o DAO, fazem parte da camada de Modelo

Encapsular acesso aos dados

- Uma das principais diferenças entre o uso de procedures e DAOs é onde o código é implementado

- Com procedures
  - a lógica fica integrada ao banco de dados
    - Isso pode conferir maior eficiência no acesso aos dados, e menor tráfego de dados

- Com DAOs
  - a lógica fica na aplicação
    - Isso confere mais portabilidade, por reduzir a dependência de recursos específicos de um fornecedor de banco de dados

Sumário

- Evolução de bancos de dados
- Modelagem Defensiva
- Controle da evolução
  - Boas práticas gerais
  - Encapsular acesso aos dados
  - Uso de ORMs

Uso de ORMs

- É bastante comum usar frameworks de persistência, chamados ORMs (*Object-Relational Mapping*), para realizar o mapeamento entre classes e tabelas de um banco de dados relacional.

- Um ORM fornece uma **camada de abstração** que oculta os detalhes de acesso aos dados, permitindo que a aplicação interaja com o banco.

Uso de ORMs

- Um ORM agiliza bastante o desenvolvimento.
- Mas e para o problema de evolução de bancos de dados, ORMs podem ajudar?
- Vamos analisar dois tipos de evolução
  - Mudança do fornecedor de banco de dados
  - Mudança no esquema do banco

Uso de ORMs

- Mudança do fornecedor de banco de dados
  - ORMs são voltados a **bancos relacionais** e já abstraem diferenças entre fornecedores (PostgreSQL, MySQL, SQL Server, etc.).
  - Contudo, se a mudança for para **outro paradigma de persistência** (ex.: um banco **NoSQL**), o esforço de adaptação será muito grande, pois todo o modelo de dados e consultas mudará.
  - Nesse caso, para máxima flexibilidade de troca de paradigma, é preferível **controlar diretamente a persistência**, usando padrões como **DAO**.

Uso de ORMs

- Mudança no esquema do banco
  - Muitos ORMs oferecem **migração automática de esquema** (criação/alteração de tabelas, colunas, índices, chaves, etc.).
  - Porém, **em produção**, isso é arriscado:
    - Falta controle explícito sobre o tratamento dos dados existentes.
    - Pode levar a **perda de dados** ou interrupções inesperadas.
  - Assim, para bancos já em produção
    - o mais seguro é planejar e executar manualmente as alterações, analisando seu impacto antes da aplicação.

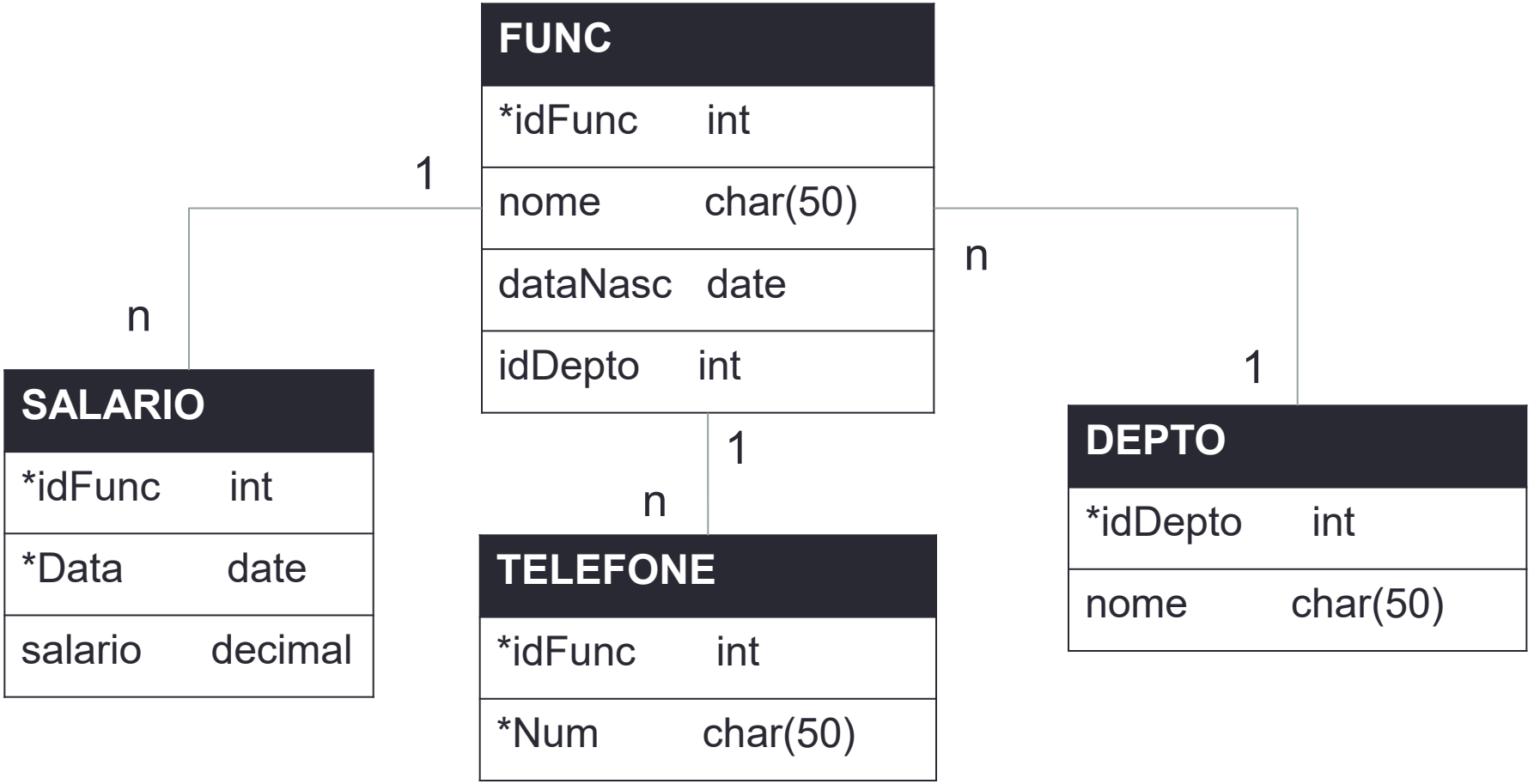
Exercício

- Com base na tabela abaixo, gere um esquema relacional mais propenso para evolução?

FUNC	
idFunc	int
nome	varchar(50)
salário	decimal
dataNasc	date
Telefone	varchar(50)
depto	varchar(50)

Exercício

- Resposta



Atividade Individual

- Normalize a tabela abaixo
- Em seguida, faça uma modelagem defensiva visando deixar o modelo mais propensão à evolução

Filme	Diretor	Prêmio	Ano Evento	Evento	Host	Idade
Gladiador	Ridley S.	Melhor Diretor	2000	Oscar	Billy Crystal	39
Gladiador	Ridley S.	Melhor Música	2000	Oscar	Billy Crystal	39
Infiltrados	Scorcese	Melhor Filme	2000	Oscar	Billy Crystal	39
Aviador	Scorcese	Melhor Filme	2007	Oscar	James Franco	35
Tropa Elite	Padilha	Melhor Filme	2007	Kikito	Billy Crystal	39
Tropa Elite	Padilha	Melhor Diretor	2007	Kikito	Billy Crystal	39