Por: Alice Dantas

MODELAGEM DE DADOS

1. Modelo Conceitual vs Modelo Lógico

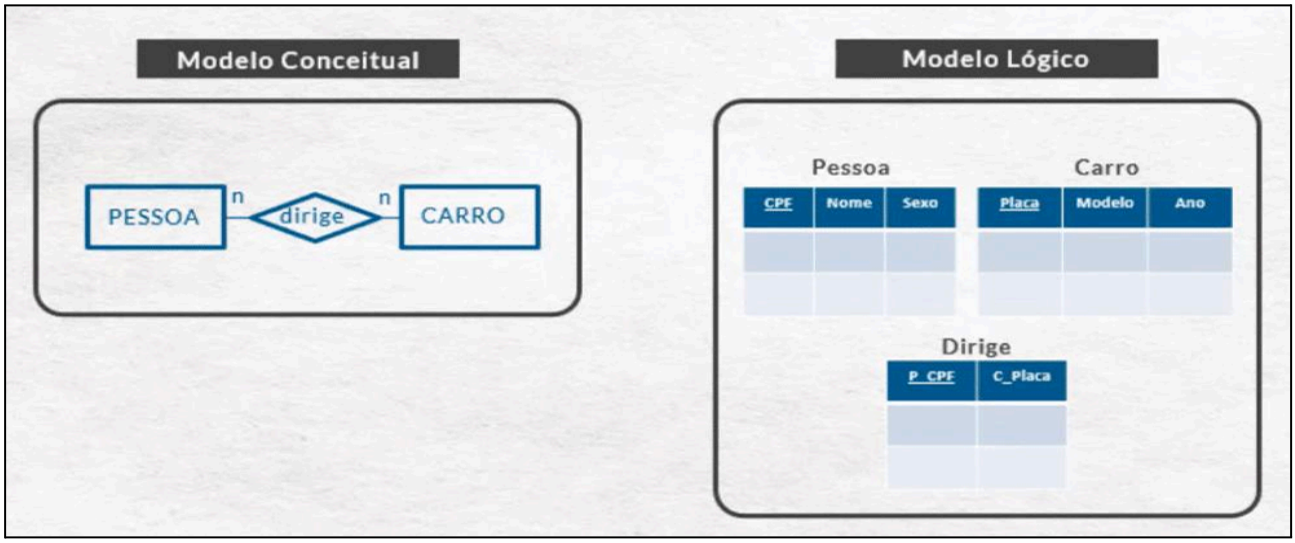
2. Modelo Físico
2.1 Principais tipos de dados
2.2 Constraints
2.3 Comando SELECT
2.4 Views

3. Modelagem de Dados com UML

4. Dados como apoio a tomadas de decisões

Por: Alice Dantas

1. Modelo Conceitual vs Modelo Lógico



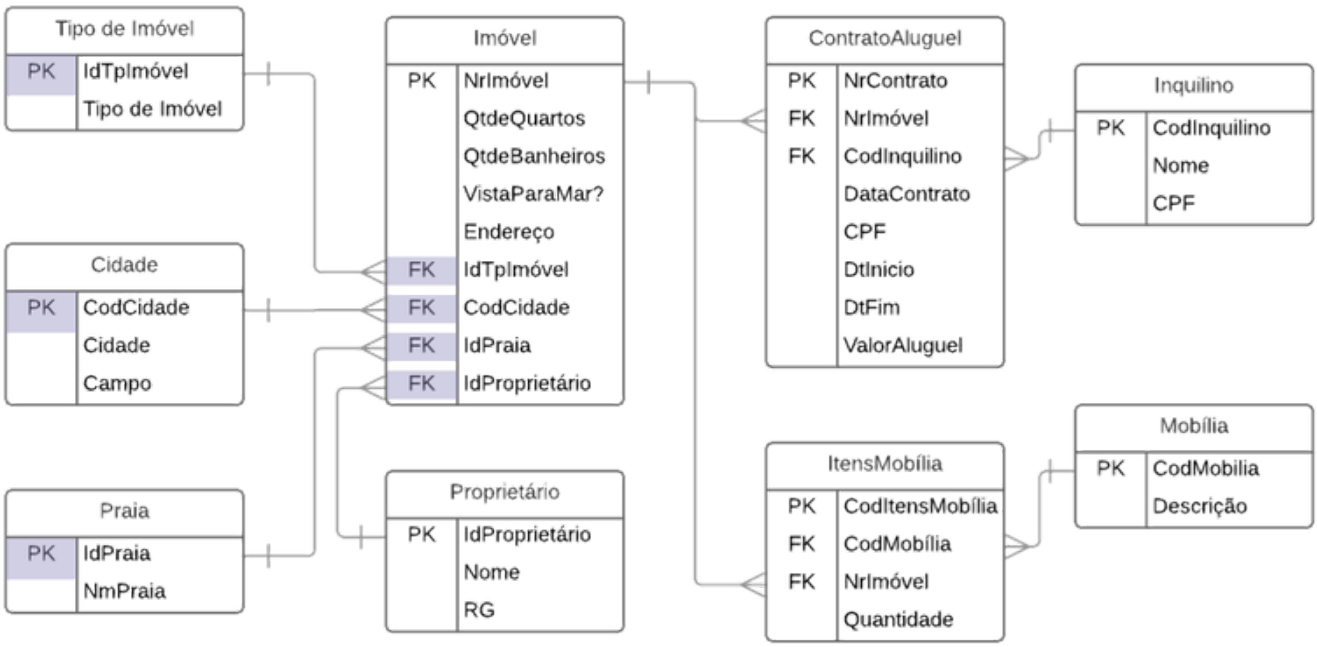
- O modelo conceitual descreve a estrutura de dados sem se preocupar com detalhes de implementação. Componentes:

- **Entidade:** é um objeto ou coisa sobre a qual coletamos informações em um banco de dados, como um cliente, um produto ou um funcionário.
- **Atributo:** são as características que descrevem uma entidade, como nome, endereço ou número de telefone.
- **Relacionamento:** representam a forma como as entidades interagem, ou a ação que uma exerce sobre a outra.

- O modelo lógico detalha a estrutura dos dados de forma mais técnica. Componentes:

- **Tabela**: representa uma entidade ou relacionamento do modelo conceitual. Cada tabela armazena registros com as mesmas características.
- **Coluna**: equivale a um atributo da entidade, representando uma característica específica dos dados armazenados. Por exemplo, uma tabela de clientes pode ter colunas como "Nome", "CPF" e "Data de Nascimento".
- **Tipo de dado**: Define o formato dos dados que podem ser armazenados em cada coluna, como `INTEGER`, `VARCHAR`, `DATE`, entre outros. Garante que as informações sejam armazenadas corretamente e com integridade.

- **Chave primária (Primary Key)**: É uma coluna (ou conjunto de colunas) que identifica unicamente cada registro em uma tabela. Nenhum valor na chave primária pode se repetir ou ser nulo.
- **Chave estrangeira (Foreign Key)**: É uma coluna que estabelece uma ligação entre duas tabelas, apontando para a chave primária de outra tabela. Serve para manter a integridade referencial entre os dados.



Como saber em qual tabela colocar a chave estrangeira?

1:N (um para muitos): A chave estrangeira vai na tabela que representa o lado "muitos".

N:M (muitos para muitos): Você cria uma **tabela intermediária** e coloca as chaves estrangeiras de ambas as tabelas.

1:1 (um para um): Você pode colocar a chave estrangeira em qualquer uma das duas tabelas, já que o relacionamento é exclusivo entre elas. Porém, geralmente escolhemos colocar a chave estrangeira na tabela que é "dependente" ou que tem mais dados relacionados ao outro.

Por: Alice Dantas

2. Modelo Físico

- O modelo físico traduz o modelo lógico em uma implementação real, com foco em como os dados vão estar organizados no disco.

Data Definition Language (DDL): comandos DDL são usados para definir e gerenciar a estrutura dos objetos de banco de dados, como tabelas, índices e esquemas.

Comando	Função
`CREATE`	Cria objetos no banco, como tabelas, índices, visões, esquemas etc.
`ALTER`	Altera a estrutura de um objeto existente (ex: adicionar colunas com ADD ou modificar tipo de uma coluna com MODIFY).
`DROP`	Exclui objetos existentes no banco, como tabelas ou visões.
`TRUNCATE`	Remove **todos os dados** de uma tabela, mas mantém sua estrutura.
`RENAME`	Renomeia um objeto de banco de dados, como uma tabela.
`COMMENT`	Adiciona comentários de descrição a objetos do banco.

```sql
-- Criar tabela
CREATE TABLE Aluno (
    ID INT PRIMARY KEY,
    Nome VARCHAR(100)
);

-- Alterar tabela
ALTER TABLE Aluno ADD Email VARCHAR(150);

-- Apagar tabela
DROP TABLE Aluno;

-- Apagar todos os dados da tabela (sem deletar a estrutura)
TRUNCATE TABLE Aluno;

-- Renomear uma tabela
ALTER TABLE Aluno RENAME TO Estudante;
```

Por: Alice Dantas

Data Manipulation Language (DML): comandos DML são usados para manipular os dados dentro dos objetos definidos pelo DDL, ou seja, para inserir, atualizar, excluir e consultar dados.

Comando	Função
SELECT	Consulta dados das tabelas.
INSERT	Insere novos registros nas tabelas.
UPDATE	Atualiza dados existentes nas tabelas.
DELETE	Exclui registros de uma tabela.

```sql
-- Inserir dados
INSERT INTO Aluno (ID, Nome) VALUES (1, 'Carlos Silva');

-- Consultar dados
SELECT * FROM Aluno WHERE Nome = 'Carlos Silva';

-- Atualizar dados
UPDATE Aluno SET Nome = 'Carlos S.' WHERE ID = 1;

-- Deletar dados
DELETE FROM Aluno WHERE ID = 1;
```

Por: Alice Dantas

2.1 Principais tipos de dados

Tipos Numéricos	Descrição	Exemplo
`INT`	Inteiro padrão (32 bits)	`122`
`DECIMAL(p,s)`	Precisão exata – útil para dinheiro	`10.99`
`FLOAT`	Ponto flutuante (menos preciso)	`3.14`
`DOUBLE`	Ponto flutuante (dupla precisão)	`3.1415926`

Tipo de Texto	Descrição	Exemplo
`CHAR(n)`	Tamanho fixo de n caracteres	`'ABC '`
`VARCHAR(n)`	Tamanho variável até n caracteres	`'Nome completo'`
`TEXT`	Texto longo (sem limite fixo)	`'Lorem ipsum...'`

Tipo de Data e Hora	Descrição	Exemplo
`DATE`	Data (ano-mês-dia)	`'2025-05-10'`
`TIME`	Hora (hora:minuto:segundo)	`'14:30:00'`
`DATETIME`	Data e hora (sem fuso horário)	`'2025-05-10 14:30:00'`
`YEAR`	Apenas o ano	`2025`

Por: Alice Dantas

2.2 Constraints

NOT NULL: A coluna não pode aceitar valores nulos (vazios).
```
nome VARCHAR(100) NOT NULL
```

AUTO_INCREMENT: O valor da coluna será gerado automaticamente, geralmente incrementando 1 a cada nova linha.
```
id INT AUTO_INCREMENT PRIMARY KEY
```

PRIMARY KEY: Define a coluna (ou conjunto de colunas) que identifica unicamente cada registro na tabela. Só pode haver uma por tabela e sempre é NOT NULL automaticamente.
```
id INT PRIMARY KEY
```

FOREIGN KEY: A coluna faz referência a outra tabela. Usado para criar relacionamentos entre tabelas.
```
cliente_id INT,
FOREIGN KEY (cliente_id) REFERENCES clientes(id)
```

DEFAULT: Define um valor padrão para a coluna caso nenhum valor seja informado.
```
status VARCHAR(20) DEFAULT 'pendente'
```

UNIQUE: Garante que os valores da coluna sejam únicos (não se repitam).
```
email VARCHAR(100) UNIQUE
```

CHECK: Impõe uma condição lógica que os valores da coluna devem obedecer.
```
idade INT CHECK (idade >= 18)
```

`ON DELETE` / `ON UPDATE`: Usado com `FOREIGN KEY` para definir o que acontece quando um registro da tabela referenciada for deletado ou atualizado. Exemplo:

```
FOREIGN KEY (cliente_id) REFERENCES clientes(id) ON DELETE CASCADE
```

- `CASCADE`: deleta também os registros relacionados.
- `SET NULL`: define como `NULL`.
- `RESTRICT`: impede a exclusão/alteração se houver dependentes.

Por: Alice Dantas

2.3 Comando SELECT

Sintaxe Básica:

```
SELECT *
FROM <nome_tabela>;
```

Filtrando Registros com a Cláusula WHERE

```
SELECT column1, column2
FROM <nome_tabela>
WHERE <condição>;
```

WHERE com LIKE. Exemplos: selecionar clientes cujo nome começa com "A“ e tenha “Ana” como parte do nome, respectivamente.

```
SELECT nome
FROM Clientes
WHERE nome LIKE 'A%’;
```

```
SELECT nome
FROM Clientes
WHERE nome LIKE ‘%Ana%';
```

WHERE com BETWEEN. Exemplo: selecionar clientes entre 20 e 30 anos.

```
SELECT nome, idade
FROM Clientes
WHERE idade BETWEEN 20 AND 30;
```

Ordenando Resultados com ORDER BY

```
SELECT column1, column2
FROM <nome_tabela>
ORDER BY column1 ASC/DESC;
```

Limitando Resultados com LIMIT

```
SELECT column1, column2
FROM <nome_tabela>
LIMIT 10;
```

Eliminando Resultados Duplicados com a Cláusula DISTINCT

```
SELECT DISTINCT column1
FROM <nome_tabela>;
```

Por: Alice Dantas

Agrupando Registros com a Cláusula GROUP BY

```
SELECT column1, f_agregada(cln2)
FROM <nome_tabela>
GROUP BY column1;
```

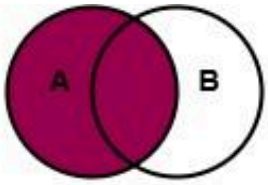
* As funções agregadas realizam cálculos em um conjunto de valores. COUNT(): Conta o número de registros. SUM(): Soma valores. AVG(): Calcula a média. MIN(), MAX(): Retorna o menor e o maior valor.

Filtrando Grupos com a C. HAVING

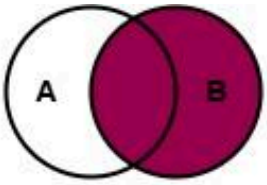
```
SELECT column1, f_agregada(cln2)
FROM <nome_tabela>
GROUP BY column1
HAVING f_agregada(cln2) > <valor>;
```

* Filtra os **grupos** gerados pelo GROUP BY com base em condições aplicadas às funções agregadas.

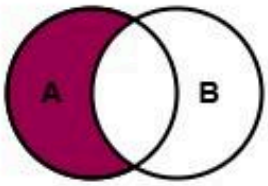
JOIN combina registros de duas ou mais tabelas com base em uma condição relacionada:



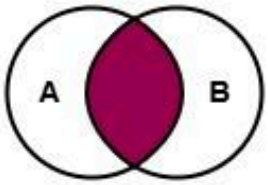

```
SELECT <coluna>
FROM tabelaA A
LEFT JOIN tabelaB B
ON A.key = B.key
```



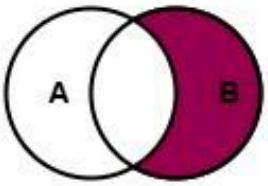

```
SELECT <coluna>
FROM tabelaA A
RIGHT JOIN tabelaB B
ON A.key = B.key
```



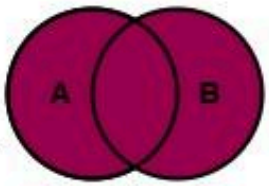

```
SELECT <coluna>
FROM tabelaA A
LEFT JOIN tabelaB B
ON A.key = B.key
WHERE B.key IS NULL
```



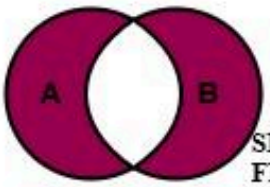

```
SELECT <coluna>
FROM tabelaA A
INNER JOIN tabelaB B
ON A.key = B.key
```




```
SELECT <coluna>
FROM tabelaA A
RIGHT JOIN tabelaB B
ON A.key = B.key
WHERE A.key IS NULL
```




```
SELECT <coluna>
FROM tabelaA A
FULL OUTER JOIN tabelaB B
ON A.key = B.key
```




```
SELECT <coluna>
FROM tabelaA A
FULL OUTER JOIN tabelaB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

Por: Alice Dantas

2.4 Views

Uma *VIEW* é como uma **consulta salva**. Você usa uma *VIEW* quando quer **reutilizar** um `SELECT` complexo, organizar melhor o banco de dados ou dar acesso controlado aos dados.

Sintaxe:

```
CREATE VIEW nome_da_view AS
SELECT colunas
FROM tabela
WHERE condição;
```

Observe a estrutura abaixo:

```
CREATE TABLE clientes (
    id INT PRIMARY KEY,
    nome VARCHAR(50),
    cidade VARCHAR(50),
    saldo DECIMAL(10, 2)
);
```

Criando a view:

```
CREATE VIEW clientes_ricos AS
SELECT nome, cidade, saldo
FROM clientes
WHERE saldo > 1000;
```

Usando uma view

```
SELECT * FROM clientes_ricos;
```

Por: Alice Dantas

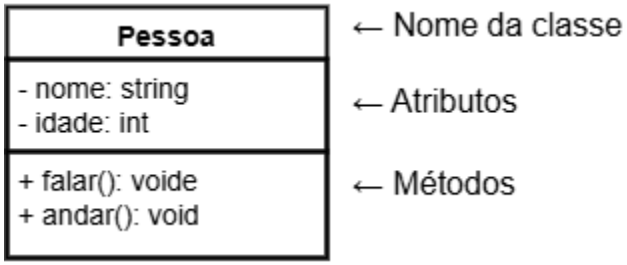
3. Modelagem de Dados com UML

A UML é uma linguagem padrão para modelar sistemas orientados a objetos, representando graficamente a estrutura e o comportamento de sistemas. Ela se assemelha ao Diagrama Entidade-Relacionamento, mas é **mais indicada quando estamos lidando com orientação a objetos**, especialmente em **sistemas que envolvem classes com métodos (funções/comportamentos).**

Visibilidade:

+ público: visível para qualquer classe.
- privado: visível somente para classe.
 # protegido: visível somente para classes derivadas.

Estrutura de uma Classe:



Por: Alice Dantas

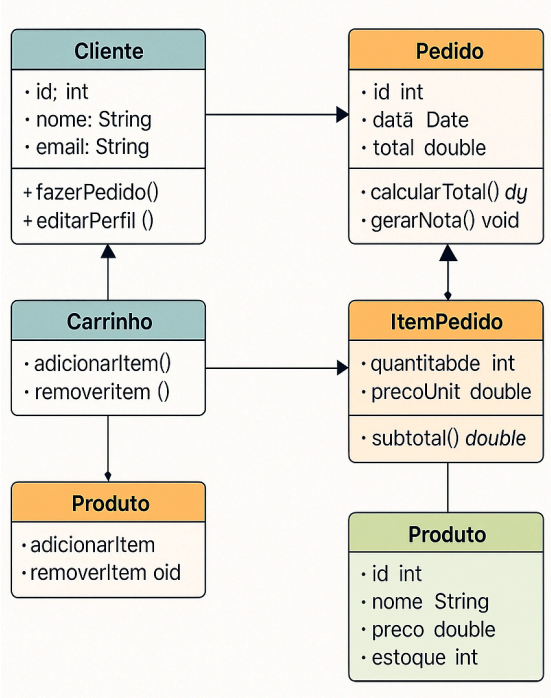
O que fazer com os métodos da UML ao converter para SQL:

Você **não traduz diretamente** métodos em comandos SQL de criação de tabelas (CREATE TABLE). Você implementa esses comportamentos depois, em:

- **Código da aplicação (Java, Python, C#, etc.)**
- **Views com lógica de consulta**

Exemplo:

Diagrama UML:



Código, em Python, para o método `subtotal()` da classe `ItemPedido`:

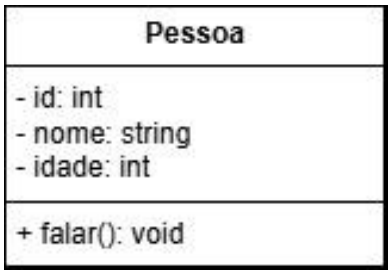
```python
class ItemPedido:
    def __init__(self, quantidade, preco_unit):
        self.quantidade = quantidade
        self.preco_unit = preco_unit

    def subtotal(self):
        return self.quantidade * self.preco_unit
```

`subtotal()` como uma VIEW em SQL:

```sql
CREATE VIEW vw_item_subtotal AS
SELECT
    id_pedido,
    id_produto,
    quantidade,
    preco_unit,
    quantidade * preco_unit AS subtotal
FROM item_pedido;
```

Exemplo 2:

Classe em UML:



Código em Python (POO):

```python
class Pessoa:
    def __init__(self, id, nome, idade):
        self.id = id
        self.nome = nome
        self.idade = idade

    def falar(self):
        print(f"{self.nome} está falando
```

Por: Alice Dantas

4. Dados como apoio a tomadas de decisões

DATA WAREHOUSE

Data Warehouse ou Depósito de Dados é um tipo especial de banco de dados. É um arquivo ou repositório de informações obtidas de várias origens (de vários bancos de dados) e armazenadas em um único local (preserva o banco de dados original da empresa). É orientado por assunto, integrado e não volátil, permitindo consultas para ajudar na tomada de decisão.

DATA MINING

Data Mining ou mineração de dados refere-se à descoberta de novas informações em função de regras ou padrões em grandes quantidades de dados e pode ser aplicado em pesquisas científicas ou em empresas com o objetivo de aumentar significativamente a lucratividade.

BI

Business Intelligence (Inteligência de Negócios) é o processo de coleta, análise, monitoria e compartilhamento de informações para a gestão de negócios. O BI analisa dados brutos operacionais para encontrar informação útil e auxiliar na tomada de decisão.