Pós-Graduação em Inteligência Artificial Generativa e LLMs

PUC-Rio

Transformando Dados em Percepção

Notas de Aula

Vítor Wilher

TDP · 2025.2

Versão 1.0

Resumo. Bancos de dados e SQL: modelagem relacional, DDL, DML e DQL, dados estruturados e não estruturados, arquiteturas de dados.

Palavras-chave: SQL; bancos de dados; modelagem; ETL

Notas de estudo elaboradas a partir do material das aulas.

Índice

1 Visão Geral da Disciplina

A disciplina **Transformando Dados em Percepção (TDP)** parte de uma premissa simples e poderosa, enunciada logo na apresentação do curso: **dados são o combustível da Inteligência Artificial**. Antes de treinar qualquer modelo, é preciso saber onde os dados estão, como eles se organizam, como garantir sua qualidade e como recuperá-los de forma eficiente. É exatamente esse o percurso proposto: sair da modelagem conceitual de um domínio de negócio, descer até a implementação física em um banco relacional, dominar a linguagem SQL em todas as suas famílias de comandos e, por fim, abrir o escopo para o universo dos dados que não cabem em tabelas.

A primeira metade do curso é dedicada aos **bancos de dados relacionais**. O ponto de partida é a modelagem conceitual com o modelo Entidade-Relacionamento de Peter Chen, apoiada pela ferramenta BR Modelo. Em seguida vem a tradução desse modelo para o **modelo lógico** (tabelas, chaves primárias e estrangeiras) e para o **modelo físico**, já em SQL. A partir daí a disciplina percorre sistematicamente as famílias de comandos da linguagem: **DDL** para criar estruturas, **DML** para manipular dados, **DQL** para consultar, **joins** para combinar tabelas e, finalmente, **funções de agregação, subconsultas e visões** para construir consultas analíticas sofisticadas. Todo esse trajeto é feito sobre um mesmo estudo de caso incremental — um sistema de gestão de projetos de uma empresa, com funcionários, departamentos, cargos, projetos e alocações — de modo que cada aula acrescenta uma camada sobre a anterior.

A segunda metade muda de terreno. Se aproximadamente 20% dos dados corporativos são estruturados, os outros 80% são **dados não estruturados**: e-mails, relatórios técnicos, pareceres, artigos, posts em redes sociais. Essas aulas apresentam a taxonomia estruturado / semiestruturado / não estruturado, os desafios linguísticos do texto (ambiguidade, ruído, vocabulário especializado, morfologia rica do português), o pipeline clássico de **pré-processamento** (limpeza, tokenização, remoção de stopwords, lematização e stemming), as formas de **representação numérica** do texto (Bag of Words, TF-IDF e word embeddings) e as operações que essas representações viabilizam: similaridade e classificação.

O fio condutor entre as duas metades é a ideia de **percepção**: SQL organiza e estrutura os dados, a IA os interpreta e gera recomendações, e juntos transformam dados em ações inteligentes. A pirâmide dado-informação-conhecimento-sabedoria, apresentada logo na primeira aula, é a metáfora que atravessa toda a disciplina.

2 Trabalhos

A avaliação da disciplina está ancorada em um **caso único e integrador**, apresentado de forma recorrente ao longo das aulas: o *Case Negociações na Bolsa de Valores (Versão Estendida).* O enunciado descreve o minimundo de uma corretora que deseja gerenciar Investidores, Ações e suas Negociações.

2.1 O minimundo do caso avaliativo

Os requisitos apresentados nos slides são os seguintes:

- Cada **Investidor** deve ser identificado por CPF ou CNPJ, além de possuir nome completo, tipo de investidor (pessoa física ou jurídica), e-mail e telefone.
- Cada **Ação** pertence a uma Empresa listada na bolsa, identificada por seu *ticker* (código de negociação), além de nome da empresa, setor de atuação e valor de mercado.
- Os investidores realizam diversas **Negociações** (compra ou venda). Para cada negociação registram-se data e hora da transação, tipo de operação, quantidade de ações e valor unitário no momento da negociação. Como um mesmo investidor pode negociar várias ações e uma ação pode ser negociada por vários investidores, esse relacionamento precisa ser bem estruturado — trata-se de um relacionamento muitos-para-muitos com atributos próprios.
- A corretora deseja manter o **Histórico de Cotações** das Ações, armazenando data, hora e valor da cotação para cada ação em determinado momento, permitindo análises de séries temporais.
- Por fim, deseja-se acompanhar o **Saldo de Carteira** de cada investidor, isto é, a posição atual em quantidade de ações mantidas, atualizada a partir das negociações realizadas.

2.2 Entregas exigidas

A atividade avaliativa pede a implementação dos três níveis de modelagem:

- **Modelo conceitual** no BR Modelo, com entrega do arquivo e do print da tela;
- **Modelo lógico**, textual ou gráfico, também com arquivo e print;
- **Modelo físico**, em arquivo `.SQL` contendo as DDLs, DMLs e DQLs.

A entrega é feita via Classroom, podendo ser realizada em equipes de até dez alunos. A sugestão dos slides é montar um documento com os resultados do conceitual e do lógico, anexando o arquivo `.SQL` à parte. A data-limite indicada é 01/12, com abertura para negociação de prazo mediante contato com o professor.

A escolha desse caso não é casual: ele exercita simultaneamente relacionamentos M:N com atributos, entidades fracas ou dependentes (o histórico de cotações), atributos identificadores alternativos (CPF ou CNPJ) e a distinção entre dados transacionais e dados derivados (o saldo de carteira, calculável a partir das negociações). É, portanto, uma síntese prática de tudo o que a primeira metade da disciplina ensina.

3 Modelagem de Bancos de Dados Relacionais

Esta aula estabelece o vocabulário fundamental da disciplina. Suas quatro máximas de abertura resumem a motivação: *quem entende a estrutura dos dados, entende o negócio*; *modelagem é o mapa antes da jornada da IA*; *conceituar primeiro é economizar tempo depois*; *IA poderosa nasce de bases de dados bem estruturadas.*

3.1 Conceitos fundamentais

Dados são fatos conhecidos que podem ser armazenados de alguma maneira e que possuem significado implícito. Um **Banco de Dados (BD)** é uma coleção *logicamente coerente* de dados relacionados — uma reunião aleatória de informações não pode ser propriamente chamada de banco de

dados. Um BD representa algum aspecto do mundo real, denominado **mini-mundo** ou **Universo do Discurso**; mudanças nesse mini-mundo devem se refletir no banco.

Três características complementares definem um banco de dados: ele é *projetado, construído e povoado* para um propósito específico; possui um grupo específico de usuários e aplicações preestabelecidas; e é mantido em dispositivos de armazenamento secundário.

3.2 A pirâmide do conhecimento

A disciplina distingue quatro níveis, que dão nome ao próprio curso:

- **Dado** — elementos não interpretados, observações, fatos ou características. Exemplos: "hoje é sexta", "12 graus Celsius", "85 km/h", "90% de umidade relativa do ar".
- **Informação** — dados que possuem significado em determinado contexto, isto é, dados após processamento. Exemplo: "95% de umidade relativa do ar em Duque de Caxias"; "85 km/h, velocidade das rajadas de vento em Copacabana".
- **Conhecimento** — interpretação formal das relações entre dados e informação; informação organizada, a chamada *expertise*. Exemplo: "se a umidade relativa do ar está em 95%, vai chover".
- **Sabedoria** — integração e evolução de múltiplos domínios de conhecimento ao longo do tempo, permitindo prever tendências e desenvolver novas teorias, antecipando-se proativamente a problemas.

3.3 Metadados, esquema, modelo e instância

Quatro noções técnicas precisam ser distinguidas com cuidado:

- **Metadados**: o SGBD mantém não somente os dados, mas a descrição completa de como eles são armazenados. Essas informações ficam no **catálogo** do SGBD e incluem a estrutura de cada arquivo, o tipo e formato de armazenamento de cada dado e as restrições.
- **Esquema de BD**: é a *descrição* do banco, especificada durante o projeto. É importante distinguir a descrição do banco do banco em si. Poucas mudanças costumam ocorrer no esquema ao longo da vida do sistema.
- **Modelo de dados**: conjunto de conceitos usados para descrever a estrutura **conceitual**, **lógica** e **física** de um banco. Por "estrutura" entendem-se os tipos de dados, os relacionamentos e as restrições. O modelo de dados fornece níveis de abstração que ocultam do usuário final os detalhes de armazenamento.
- **Instância**: os dados armazenados em um determinado instante do tempo. A instância se altera a cada modificação; cabe ao SGBD garantir que toda instância satisfaça o esquema, respeitando estrutura e restrições.

3.4 Sistemas de Gerenciamento de Bancos de Dados

Um **SGBD** é uma coleção de programas que possibilita ao usuário criar e manter um banco de dados. Trata-se de software de uso geral, não direcionado a nenhuma aplicação específica, e permite três operações centrais:

- **Definir** um banco: especificar tipos de dados, estruturas e restrições;
- **Construir**: armazenar os dados em algum meio manipulado pelo SGBD;
- **Manipular**: buscar informações, modificar dados existentes e gerar relatórios.

As características desejáveis em um SGBD listadas na aula são: propriedades **ACID** (Atomicidade, Consistência, Isolamento e Durabilidade), persistência de objetos, controle de redundância, manutenção de restrições de integridade, representação de relacionamentos complexos, restrições de acesso, múltiplas interfaces de usuário, realização de inferências com regras de dedução e rotinas de backup e recuperação.

3.5 O modelo Entidade-Relacionamento

O modelo conceitual adotado é o **Modelo Entidade-Relacionamento (E-R)**, proposto por **Peter Chen** e baseado no princípio: *"O mundo está cheio de coisas que possuem características próprias e que se relacionam umas com as outras."*

Cada trecho dessa frase corresponde a um construto do modelo:

- **Entidades** ("o mundo está cheio de coisas"): conjuntos de objetos com os mesmos tipos de características. Exemplos: a entidade Máquina, representando todas as máquinas de uma fábrica; a entidade Funcionário. Um objeto pertencente ao conjunto é uma **instância**.
- **Atributos** ("que possuem características próprias"): tipos de características comuns a todas as instâncias de uma entidade. A identificação de atributos comuns é justamente o que permite agrupar objetos em entidades.
- **Relacionamentos** ("e que se relacionam umas com as outras"): associações possíveis entre instâncias de entidades diferentes ou da mesma entidade.

Para reconhecer entidades, a aula sugere observar cinco grandes grupos de elementos (referência a Shlaer e Mellor): coisas tangíveis (meio de transporte, animal, equipamento), funções exercidas por elementos (especialista, cliente, atendente), eventos ou ocorrências (voo comercial, acidente de trânsito), interações (compra de imóvel, venda realizada por um fornecedor) e especificações (modelo de refrigerador). O nível de abstração adotado pode ser maior ou menor conforme o problema.

3.6 Tipologia dos atributos

Os atributos classificam-se segundo quatro eixos independentes:

1. **Simples (atômicos) x Compostos** — atributos simples não podem ser divididos; compostos podem ser divididos em subpartes. Exemplo clássico: `Endereco` composto por rua, bairro e cidade.
2. **Monovalorados x Multivalorados** — monovalorados possuem um único valor por instância; multivalorados podem ter vários. Exemplo: `Telefone` de uma pessoa.
3. **Armazenados x Derivados** — armazenados têm valores próprios; derivados são obtidos a partir de outros atributos. Exemplo: `Idade` derivada de `Data_de_Nascimento`.
4. **Chave (identificador) x Não-chave** — um atributo chave possui valor distinto para cada instância. Exemplo: `CPF` na entidade Pessoa.

O exemplo canônico da aula é a entidade Pessoa, com `CPF` como chave, `Endereco` composto (rua, bairro, cidade), `Telefone` multivalorado, `Tipo_Sanguineo` monovalorado e simples, e `Idade` derivada de `Data_de_Nascimento`.

3.7 Multiplicidade: participação e cardinalidade

A **multiplicidade** de um relacionamento é a composição de **participação** e **cardinalidade**, representada no formato `(min, max)`:

- A **cardinalidade** indica o **número máximo** de relacionamentos daquele tipo dos quais cada instância de uma entidade pode participar. Representa-se do lado oposto ao da entidade a que se refere. Os tipos são **1:1**, **1:N** e **M:N**. As letras M e N são usadas quando o máximo é indefinido; havendo um limite conhecido, ele pode ser escrito no lugar (por exemplo, "cada empregado supervisiona no máximo cinco projetos" gera cardinalidade 5).
- A **participação** indica o **número mínimo**, ou seja, se o relacionamento é obrigatório. Participação **total ou obrigatória** (representada por 1) significa que todas as instâncias devem participar; participação **parcial ou opcional** (representada por 0) admite instâncias sem nenhum relacionamento daquele tipo.

O exemplo trabalhado é: "Uma pessoa pode não possuir automóveis. Todo automóvel pertence a uma pessoa", que produz `Pessoa (0,N) Possui (1,1) Automovel`.

3.8 Nomes de papéis e atributos de relacionamento

Nomes de papéis especificam a função realizada por uma entidade em um relacionamento e são usados quando há risco de ambiguidade. Exemplo: no relacionamento entre Departamento e Funcionário, o papel "trabalhador" deixa claro que é o funcionário quem trabalha para o departamento. São especialmente úteis em autorrelacionamentos, como Empregado *supervisiona* Empregado, com os papéis "supervisor" e "supervisionado".

Atributos de relacionamento aparecem quando uma informação não pode ser alocada em nenhuma das entidades envolvidas, porque varia de acordo com a combinação das instâncias. O exemplo dado é definitivo: "Um empregado pode trabalhar em vários projetos. Um projeto tem vários empregados. O número de horas trabalhadas por cada empregado em cada projeto varia de empregado para empregado e de projeto para projeto." Logo, `Numero_de_Horas` é atributo do relacionamento *Trabalha Em*, e não de Empregado nem de Projeto.

3.9 Especialização e generalização

Algumas vezes identificam-se características comuns a apenas *algumas* instâncias de uma entidade, formando subconjuntos. Esses subconjuntos são as **subclasses** ou **especializações**; a entidade de origem é a **superclasse** ou **generalização**. As instâncias de uma subclasse possuem todas as características da superclasse acrescidas das características pertinentes apenas à subclasse.

Os dois processos são inversos:

- **Especialização**: processo *top-down* de identificação de subclasses a partir de uma entidade genérica. Exemplo: de Cliente para Pessoa Física e Pessoa Jurídica.

- **Generalização**: processo *bottom-up* pelo qual se define uma entidade mais genérica a partir de entidades especializadas.

O exemplo desenvolvido é o de Cliente: todos os clientes possuem número único de identificação e telefones; apenas os do tipo pessoa física possuem CPF único e nome; apenas os do tipo pessoa jurídica possuem CNPJ e razão social. Outro exemplo é Veículo Automotor como superclasse de Carro, Caminhão e Trator.

Duas restrições qualificam a hierarquia:

- **Totalidade** — a especialização é **total** quando todas as instâncias da superclasse pertencem a alguma subclasse (representada por linha dupla ligando a superclasse ao círculo Gen-Espec, marcada com T); é **parcial** quando pode haver instâncias que não pertencem a nenhuma subclasse (linha simples, marcada com P). Exemplo de total: todo Colaborador é CLT ou PJ. Exemplo de parcial: alguns Alunos não são bolsistas nem monitores.
- **Exclusividade** — a restrição **exclusiva** implica que uma instância não pode pertencer a mais de uma subclasse simultaneamente (Carro não pode ser Caminhão). A restrição **não-exclusiva** admite pertencimento múltiplo (um Aluno pode ser bolsista e monitor ao mesmo tempo).

3.10 Ferramentas e casos práticos

As ferramentas indicadas são o SGBD **PostgreSQL**, o modelador **BR Modelo** (versões 2.0 em Delphi, 3.x em Java e a versão Web) e o site **sqliteonline.com** para prática online. O caso motivador introdutório é o de um sistema de recomendação de filmes a partir dos filmes assistidos por um usuário de streaming.

Os exercícios propostos incluem o **Estudo de Caso Empresa** (funcionários com nome, CPF, endereço, telefones, cargo e salário; departamentos com nome, localização composta por bloco, andar e sala, e sigla única de 3 caracteres; projetos com nome, código, descrição e gerente, com data de início e quantidade de horas registradas por associação) e o **Gerenciamento de Atendimento** (clientes internos e externos, contratos e SLAs). Há ainda seis micro casos de fixação: Clínica de Exames, Biblioteca Digital, Curso Online de IA, Loja Virtual, Funcionários de uma Empresa de Tecnologia e Veículos em uma Locadora — os três últimos exercitando especificamente especialização.

4 Modelagem Lógica e Física de Bancos Relacionais

O **modelo lógico** é o produto da conversão de um modelo conceitual para um determinado tipo de banco de dados. É a descrição do banco no nível de abstração visto pelo usuário do SGBD. Nessa fase o projetista já deve conhecer o tipo de banco em que o projeto será implementado — relacional, hierárquico, objeto-relacional ou não-relacional. O **modelo relacional** é o modelo lógico mais utilizado. É também aqui que se introduzem as **chaves**: primárias, estrangeiras e candidatas.

4.1 Mapeamento de entidades e atributos

Normalmente, **entidades viram tabelas**. Em uma entidade simples, a entidade Filme torna-se a tabela Filme, representada textualmente como:

```
Filme (codFilme, nomeFilme, anoProducao)
```

A representação pode incluir os tipos de dados:

```
Filme (codFilme:integer, nomeFilme:varchar(100), anoProducao:integer)
```

O modelo lógico também pode ser representado graficamente, e é assim que a maioria das ferramentas o faz — BR Modelo, MySQL Workbench, ERWIN, Power Architect e DBeaver.

Para atributos não atômicos, valem duas regras:

- **Atributo composto**: cria-se uma coluna para cada uma de suas partes.
- **Atributo multivalorado**: cria-se colunas na mesma tabela ou, preferencialmente, cria-se uma nova tabela.

O exemplo de mapeamento apresentado, com as chaves estrangeiras destacadas, é:

```
Distribuidora (codDistribuidora, log, comp, num, bairro,
               cidade, uf, cep, email0br, email0p)
Telefone (codTel, numero, codDistribuidora)
```

Note que o endereço composto foi decomposto em colunas na própria tabela Distribuidora, enquanto os telefones multivalorados geraram uma tabela própria com chave estrangeira apontando para a distribuidora.

4.2 Mapeamento de relacionamentos

As regras variam conforme a multiplicidade:

Relacionamentos 1:N — o lado obrigatório cede uma **chave estrangeira** para o lado "N". Havendo atributo de relacionamento, este acompanha a chave estrangeira. Resultado:

```
Departamento (codDepto, nome)
Empregado (codEmp, nome, codDepto)
```

Relacionamentos 1:1 — ocorre a **fusão entre as tabelas**, escolhendo-se uma das entidades para nomear a tabela resultante:

```
Empregado (codemp, nomeemp, armario)
```

Relacionamentos M:N — cria-se uma **nova tabela**, cuja chave primária é composta pelas duas chaves primárias participantes. Os atributos de relacionamento são mapeados para essa nova tabela:

```
Empregado (codEmp, nomeEmp)
Projeto (codProj, tituloProj)
Atuacao (codEmp, codProj, dataalocacao)
```

Esse último padrão é o mais importante da aula: é ele que resolve o relacionamento Investidor–Ação do caso avaliativo, e é ele que aparece na tabela `alocacao` usada durante todo o restante da disciplina.

4.3 Mapeamento de especialização e generalização

Há três estratégias possíveis:

1) Uma relação para toda a hierarquia. A relação recebe o nome da entidade genérica, a chave primária é o atributo identificador desta, e todos os atributos das entidades especializadas são mapeados para a mesma tabela. *Vantagem*: melhor desempenho, pois todos os dados ficam juntos. *Desvantagens*: os relacionamentos específicos são perdidos, a manutenção é prejudicada, há desperdício de espaço e — na formulação enfática dos slides — o modelo é ignorado.

2) Uma relação para cada entidade especializada. Somente as entidades especializadas viram tabelas; os atributos da genérica são repetidos em cada uma; a entidade genérica deixa de existir; cria-se um atributo especial para tipar a classe. *Vantagens*: fácil mudança de categoria e melhor desempenho. *Desvantagem*: redundância.

3) Uma relação por entidade. Esta é a estratégia adotada na disciplina. Cada entidade do modelo ganha uma tabela própria; cria-se uma chave primária para a entidade especializada; as entidades especializadas recebem uma chave estrangeira a partir da entidade genérica. *Vantagem*: é a que melhor representa o modelo. *Desvantagem*: desempenho em consultas prejudicado pelas junções necessárias.

4.4 Autorrelacionamento

Ocorre quando uma entidade se relaciona com ela mesma, o que evita duplicar a entidade no modelo. O mapeamento gera uma chave estrangeira que aponta para a própria tabela:

```
Aluno (matr, nome, lider)
```

Aqui `lider` é uma chave estrangeira que referencia `matr` da própria tabela Aluno.

4.5 Notação de James Martin

Além da notação de Peter Chen, a aula apresenta a **notação Engenharia de Informações**, criada na década de 1980 por **James Martin** e popularmente conhecida como notação **"pés-de-galinha"** (*crow's foot*). Os slides comparam lado a lado as duas notações para as mesmas situações de modelagem, mostrando que se trata de representações alternativas dos mesmos conceitos de cardinalidade e participação.

4.6 Exercícios da aula

Os exercícios pedem o mapeamento para o modelo lógico dos modelos conceituais construídos na aula anterior: o micro caso Clínica de Exames, o minimundo Gerenciamento de Atendimento e, como exercício extra, o Case de Negociações na Bolsa de Valores.

5 SQL Para Criação de Estruturas (DDL)

A **SQL (Structured Query Language)** é a linguagem padrão de consulta a bancos de dados relacionais. Seus comandos agrupam-se em cinco categorias:

- **DDL** — Linguagem de Definição de Dados;
- **DML** — Linguagem de Manipulação de Dados;
- **DCL** — Linguagem de Controle de Dados;
- **DTL** — Linguagem de Transação de Dados;
- **DQL** — Linguagem de Consultas de Dados.

Esta aula concentra-se na DDL, cujos três comandos centrais são `CREATE`, `ALTER` e `DROP`.

5.1 O comando CREATE

Usado para criar objetos no banco: tabelas, índices, visões, esquemas. A sintaxe básica para tabelas é:

```
CREATE TABLE nomedatabela (
  coluna_1 tipo primary key,
  coluna_2 tipo [restricao],
  coluna_n tipo [restricao]
);
```

É possível — e recomendável — **nomear as restrições** de chave primária e estrangeira. Caso não se faça isso, o próprio SGBD se encarrega de nomeá-las, geralmente com nomes pouco legíveis:

```
CREATE TABLE nomedatabela (
  coluna_1 tipo,
  coluna_2 tipo,
  coluna_n tipo,
  CONSTRAINT pk_nomedatabela PRIMARY KEY (coluna_1),
  CONSTRAINT fk_nomedatabela_coluna2 FOREIGN KEY (coluna_2)
    REFERENCES outratabela (colunaX)
);
```

5.2 Tipos de dados

Os principais tipos apresentados, no contexto do PostgreSQL, são:

- Inteiros: `int` ou `integer`, `bigint`, `serial`, `smallint`;
- Decimais: `real`, `float`, `money`, `decimal(x,y)`;

- Texto de tamanho variável: `varchar(n)`, onde `n` é o número máximo de caracteres;
- Texto de tamanho fixo: `char(n)`;
- Datas: `date`;
- Carimbo de data e hora: `timestamp`.

O tipo `serial` merece destaque: é ele que provê a numeração automática exigida no Desafio 1 da aula.

5.3 Chaves estrangeiras

As chaves estrangeiras viabilizam o relacionamento entre tabelas, minimizando a repetição de dados. Podem ser declaradas de forma anônima ou nomeada:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo primary key,
  coluna2 tipo,
  coluna_n tipo,
  foreign key (coluna_n)
    references tabela_referenciada (chave_primaria)
);
```

5.4 A restrição CHECK

A restrição `CHECK` ajuda a manter a consistência do banco, validando os valores admissíveis em uma coluna. Pode ser nomeada ou não:

```
-- Nomeando a constraint
CREATE TABLE pessoa (
  id INT PRIMARY KEY,
  nome VARCHAR(100) NOT NULL,
  estcivil CHAR(1),
  CONSTRAINT chk_estcivil CHECK (estcivil IN ('S', 'V', 'C', 'D'))
);

-- Sem nomear a constraint
CREATE TABLE pessoa (
  id INT PRIMARY KEY,
  nome VARCHAR(100) NOT NULL,
  estcivil CHAR(1) CHECK (estcivil IN ('S', 'V', 'C', 'D'))
);
```

5.5 O comando ALTER

Usado sempre que for preciso modificar a estrutura de uma tabela já existente. As dez formas apresentadas cobrem praticamente todas as necessidades cotidianas:

```
-- 1) Criar uma nova coluna
ALTER TABLE nome_da_tabela ADD COLUMN nome_da_coluna TIPO_COLUNA;

-- 2) Alterar o tipo de uma coluna
```

```sql
ALTER TABLE nome_da_tabela ALTER COLUMN nome_da_coluna TYPE novo_tipo;

-- 3) Apagar uma coluna
ALTER TABLE nome_da_tabela DROP COLUMN nome_da_coluna;

-- 4) Definir um valor default
ALTER TABLE nome_da_tabela
  ALTER COLUMN nome_coluna SET DEFAULT valor_default;

-- 5) Renomear uma coluna
ALTER TABLE nome_da_tabela RENAME COLUMN nome_antigo TO nome_novo;

-- 6) Renomear uma tabela
ALTER TABLE nome_da_tabela RENAME TO nome_novo;

-- 9) Adicionar uma chave primária
ALTER TABLE nome_da_tabela ADD PRIMARY KEY (nome_da_coluna);

-- 10) Adicionar uma chave estrangeira
ALTER TABLE nome_da_tabela ADD FOREIGN KEY (nome_da_coluna)
  REFERENCES nome_da_tabela_referenciada (nome_da_coluna_referenciada);
```

Também é possível definir ou remover restrições de coluna, como `NOT NULL`, usando `ALTER COLUMN ... SET` e `ALTER COLUMN ... DROP`.

5.6 O comando DROP

Elimina objetos do banco:

```sql
-- Apagar uma tabela
DROP TABLE nome_da_tabela;

-- Apagar uma tabela que tenha dependência com outra
DROP TABLE nome_da_tabela CASCADE;

-- Apagar mais de uma tabela
DROP TABLE tabela1, tabela2;
```

Um exercício da aula explora didaticamente esse ponto: pede-se que se tente apagar a tabela `funcionario` sem `CASCADE`. A operação falha justamente porque existe uma tabela `telefone` com chave estrangeira apontando para ela — demonstração prática da **integridade referencial** em ação.

5.7 Exercícios e o Desafio 1

Os exercícios constroem, tabela a tabela, o banco `empresa` que será usado no restante da disciplina: `projeto`, `departamento`, `funcionario`, `participa` (posteriormente renomeada para `alocacao`), `cargo` e `telefone`. Entre as manipulações pedidas: renomear o campo `descricao` da tabela `projeto` para `descproj`, criar a coluna `estcivil` como `char(1)`, alterar `nomefunc` para `varchar(100)`, apagar a coluna CBO da tabela `cargo`, definir `'Interno'` como default da coluna `tipo` em `projeto`, tornar `nomefunc` obrigatório e criar `idcargo` em `funcionario` como chave estrangeira.

O **Desafio 1** integra tudo: implementar uma tabela de clientes em que `codcliente` seja chave primária com numeração automática, `sexo` aceite `'M'` ou `'F'` podendo ser vazio, `estcivil` aceite apenas `'Solteiro'`, `'Casado'`, `'Separado'`, `'Divorciado'` ou `'Viúvo'` e não possa ser vazio, `UF` tenha default `'RJ'`, e `datacadastro` grave automaticamente data e hora do cadastro. Além disso, cinco clientes devem ser inseridos informando apenas nome, sexo e estado civil — todos os demais campos preenchidos automaticamente pelo SGBD. O desafio exige, portanto, combinar `serial`, `CHECK`, `DEFAULT`, `NOT NULL` e default temporal em um único `CREATE TABLE`.

6 SQL Para Manipulação de Dados (DML)

A **DML** é a parte da SQL responsável por manusear os dados nas tabelas. Além dos comandos de recuperação baseados em `SELECT` (que a disciplina trata separadamente como DQL), a DML tem três comandos: **INSERT**, **UPDATE** e **DELETE**.

6.1 INSERT

Insere dados em uma tabela:

```
INSERT INTO nome_da_tabela (coluna1, coluna2, coluna_n)
VALUES (valor1, valor2, valor_n);
```

Quando todas as colunas são preenchidas, sem pular nenhuma, é possível omitir o nome dos campos — desde que a ordem dos valores respeite exatamente a ordem das colunas na definição da tabela:

```
INSERT INTO nome_da_tabela VALUES (valor1, valor2, valor_n);
```

Também é possível inserir vários registros de uma só vez, o que é bem mais eficiente do que executar múltiplos comandos:

```
INSERT INTO nome_da_tabela (coluna1, coluna2, coluna_n) VALUES
  (valor1, valor2, valor_n),
  (valor1, valor2, valor_n);
```

Dois cuidados são explicitamente ressaltados nos slides: **textos devem ser inseridos com aspas simples** e **datas no formato americano** (ano-mês-dia).

6.2 Carga de dados dos exercícios

Os exercícios populam o banco `empresa` com dados que serão reutilizados em todas as aulas seguintes. Os projetos incluem Governança de Dados, Segurança da Informação, Análise Preditiva, Integração de Sistemas e Qualidade de Dados, com orçamentos entre 30.000 e 120.000 e classificação em Interno ou Externo. Os departamentos são TI e ADM. Os cargos são Analista de Sistemas, Cientista de Dados e Analista de Dados, com salários de 9.500,00, 15.000,00 e 10.000,00. Os funcionários cadastrados são Ana Flor, André Maia, Andreia Silva, Bruna Martins e Bruno Moreira, cada um com gênero, data de nascimento, cargo e departamento.

Em seguida vêm as **alocações**, que materializam o relacionamento M:N entre funcionários e projetos, com quantidade de horas e data. Um exemplo de leitura: "Ana Flor trabalhou 30 horas no projeto Qualidade de Dados, em 01-02-2025" traduz-se em uma linha da tabela `alocacao` com o código do funcionário, o código do projeto, as horas e a data.

6.3 UPDATE

Permite alterar dados já inseridos. O ponto pedagógico central da aula é a advertência sobre a cláusula `WHERE`: sem ela, **todos** os registros da tabela são modificados.

```
UPDATE nome_da_tabela
SET nome_do_campo = alteracao_desejada
WHERE condicao_para_alteracao;
```

O exemplo dado é intencionalmente problemático:

```
UPDATE funcionario SET cidade = 'Rio de Janeiro'
WHERE nomefunc = 'Paulo';
```

Os slides perguntam: "Qual o problema aqui?". O problema é que `nomefunc` não é chave: pode haver mais de um Paulo, e a atualização atingiria todos eles. Filtros de atualização devem, sempre que possível, usar a chave primária.

Os exercícios de UPDATE exploram atualizações de negócio realistas: aumentar o salário dos Analistas de Sistemas para 10.000,00, transferir uma funcionária para outro setor, acrescentar 10 horas a todas as alocações de uma pessoa, dar 5% de aumento a todos os funcionários, aplicar 10% de desconto no orçamento de um projeto e padronizar nomes de departamentos ("ADM" para "Administração", "TI" para "Tecnologia da Informação"). Note que os dois últimos tipos exigem expressões aritméticas no `SET`, como `orcamento = orcamento * 0.9`.

6.4 DELETE

Elimina linhas de uma tabela. A aula insiste na distinção crucial: **DELETE não é DROP**. O `DELETE` mantém a estrutura da tabela, eliminando apenas os dados; o `DROP` apaga toda a estrutura.

```
DELETE FROM nome_da_tabela;

DELETE FROM nome_da_tabela
WHERE condicao;
```

Assim como no `UPDATE`, a ausência do `WHERE` é destrutiva: apaga todas as linhas.

7 SQL Para Consultas de Dados (DQL)

A **DQL** é frequentemente vista como parte da DML, mas distingue-se por ser usada somente para **consultar** dados já armazenados, sem modificá-los. É composta essencialmente pelo comando `SELECT` e suas cláusulas.

7.1 As cláusulas associadas ao SELECT

- **FROM**: define de onde os dados vêm — tabelas ou views;
- **WHERE**: filtra linhas específicas;
- **GROUP BY / HAVING**: agrupam registros e permitem aplicar funções de agregação;
- **ORDER BY**: ordena os resultados;
- **JOINs**: combinam dados de várias tabelas.

7.2 Sintaxe básica

```
SELECT coluna1, coluna2, coluna_n
FROM nome_da_tabela;

-- Todas as colunas
SELECT *
FROM nome_da_tabela;
```

7.3 ORDER BY

Altera a forma como os resultados são exibidos, em ordem ascendente ou descendente:

```
SELECT coluna1
FROM tabela
ORDER BY coluna_ordenada ASC;

SELECT coluna1
FROM tabela
ORDER BY coluna_ordenada DESC;
```

O `ASC` é opcional quando a ordem desejada é crescente, já que este é o comportamento padrão.

7.4 DISTINCT

Frequentemente os resultados se repetem desnecessariamente. Quando interessa apenas o conjunto de valores distintos, usa-se `DISTINCT`:

```
SELECT DISTINCT coluna1
FROM tabela1;
```

7.5 WHERE e operadores de comparação

Filtros limitam os resultados, fazendo com que o banco retorne apenas as linhas que satisfazem determinada condição:

```
SELECT coluna1, coluna2, coluna_n
FROM tabela1
WHERE condicao_desejada;
```

Os operadores de comparação disponíveis são: maior, menor, maior ou igual, menor ou igual e as duas formas de "diferente" aceitas pela SQL. Múltiplas condições podem ser combinadas com `OR` e `AND`:

```
SELECT coluna1
FROM tabela1
WHERE condicao1 OR condicao2
  AND condicao3;
```

Vale a advertência implícita nesse exemplo: `AND` tem precedência sobre `OR`, de modo que expressões mistas devem ser parentetizadas para evitar resultados inesperados.

7.6 LIKE e o curinga percentual

A cláusula `LIKE` permite filtrar por parte do texto. O símbolo `%` representa qualquer sequência de caracteres, e sua posição determina o comportamento da busca:

```
-- Contém "MARIA" em qualquer posição
SELECT nome FROM cliente WHERE nome LIKE '%MARIA%';
-- Retorna: ANA MARIA, MARIA SANDRA, MARIANA SILVA, MARIAH DO NASCIMENTO

-- Termina com "MARIA"
SELECT nome FROM cliente WHERE nome LIKE '%MARIA';
-- Retorna: Ana Maria, Luana Silva Maria

-- Começa com "MARIA"
SELECT nome FROM cliente WHERE nome LIKE 'MARIA%';
-- Retorna: Maria Sandra, Mariana Silva
```

7.7 Padrões de consulta exercitados

Os exercícios da aula combinam sistematicamente essas cláusulas: recuperar nome e estado civil dos funcionários; listar nomes em ordem alfabética; obter datas de alocação da mais recente para a mais antiga, com e sem repetição; filtrar por gênero; recuperar funcionários nascidos antes de 1990 ordenados por data de nascimento; excluir um cargo específico usando o operador de diferença; listar projetos com orçamento maior ou igual a determinado valor; recuperar códigos de projeto sem repetição para alocações acima de certo número de horas; filtrar alocações de um mês específico; e usar `LIKE` para encontrar nomes que iniciem com uma letra, sobrenomes específicos ou projetos cujo nome termine com determinada palavra. O último exercício combina `LIKE` com filtro por departamento, exigindo múltiplas condições simultâneas.

8 SQL Avançado - Joins e Combinações Complexas

Uma das principais características do modelo relacional é permitir a **junção entre tabelas** através de chaves primárias e estrangeiras. É justamente esse recurso que ajuda a manter a consistência do banco e a eliminar redundâncias desnecessárias — o preço da normalização é a necessidade de recompor a informação no momento da consulta.

A aula apresenta seis formas de junção: **EQUIJOIN**, **INNER JOIN**, **LEFT JOIN**, **RIGHT JOIN**, **FULL OUTER JOIN** e **NATURAL JOIN**. Os slides indicam o recurso visual `joins.spathon.com` como apoio para entender as diferenças.

8.1 EQUIJOIN

É a forma mais simples e também a mais antiga: iguala-se, na cláusula `WHERE`, a chave estrangeira de uma tabela à chave primária de outra.

```
SELECT tabela1.coluna1, tabela2.coluna1
FROM tabela1, tabela2
WHERE tabela1.fk = tabela2.pk;
```

Para evitar redigitar nomes longos de tabelas, usam-se **apelidos** (*aliases*). O uso da palavra `AS` é opcional:

```
SELECT t1.coluna1, t2.coluna1
FROM tabela1 as t1, tabela2 as t2
WHERE t1.pk = t2.fk;
```

8.2 INNER JOIN

Nesta forma usa-se explicitamente a palavra `JOIN`, e a condição de junção migra do `WHERE` para a cláusula `ON`. A palavra `INNER` é opcional; é comum escrever apenas `JOIN`.

```
SELECT tabela1.coluna1, tabela2.coluna1
FROM tabela1 INNER JOIN tabela2
  ON tabela1.fk = tabela2.pk;
```

O resultado é equivalente ao do equijoin, mas a sintaxe separa claramente a condição de *junção* das condições de *filtro*, o que torna a consulta mais legível e menos propensa a erros — especialmente quando várias tabelas são combinadas.

8.3 LEFT JOIN

Recupera **todos** os dados da tabela à esquerda da declaração, mesmo aqueles que não possuem correspondência na tabela da direita. As colunas da tabela direita ficam nulas nesses casos.

```
SELECT tabela1.coluna1, tabela2.coluna1
FROM tabela1 LEFT JOIN tabela2
  ON tabela1.fk = tabela2.pk;
```

8.4 RIGHT JOIN

Funciona de forma espelhada: recupera todos os dados da tabela à direita, mesmo sem correspondência à esquerda.

```sql
SELECT tabela1.coluna1, tabela2.coluna1
FROM tabela1 RIGHT JOIN tabela2
  ON tabela1.fk = tabela2.pk;
```

Os exercícios deixam claro **quando** essas junções externas são necessárias: listar todos os departamentos e seus funcionários, *incluindo os departamentos que não possuem funcionários*; listar todos os funcionários e as datas de início em projetos, *incluindo os que não participaram de nenhum projeto*; listar todos os cargos e salários com os nomes dos funcionários que os exercem, *incluindo cargos vagos*. Em todos os casos, um INNER JOIN faria essas linhas simplesmente desaparecerem do resultado — o que seria um erro analítico grave.

8.5 FULL OUTER JOIN

Recupera todos os dados de ambas as tabelas, mesmo sem correspondência entre elas.

```sql
SELECT tabela1.coluna1, tabela2.coluna1
FROM tabela1 FULL OUTER JOIN tabela2
  ON tabela1.fk = tabela2.pk;
```

Uma observação prática relevante: o **MySQL não aceita esse comando**. Nesse SGBD é necessário simular o comportamento combinando `LEFT JOIN` e `RIGHT JOIN` com o operador `UNION`.

8.6 NATURAL JOIN

É a forma mais simples sintaticamente, mas exige uma condição forte: **as chaves primárias e estrangeiras precisam possuir o mesmo nome** nas duas tabelas. O SGBD infere automaticamente a condição de junção a partir das colunas homônimas.

```sql
SELECT colunas
FROM tabela1
NATURAL JOIN tabela2;
```

A conveniência tem contrapartida: qualquer coluna com nome coincidente entra na junção, mesmo que semanticamente não devesse. Por isso o `NATURAL JOIN` é elegante em modelos bem nomeados e perigoso em modelos legados.

9 Funções, Subqueries e Visões

Esta aula fecha o bloco relacional introduzindo os recursos que transformam consultas descritivas em consultas **analíticas**.

9.1 Funções de agregação

As principais funções apresentadas são:

- **SUM** — fornece o somatório de uma coluna a partir de uma query;

- **COUNT** — conta a quantidade de linhas resultantes de uma query;
- **MAX** — recupera o valor máximo de uma coluna;
- **MIN** — recupera o valor mínimo de uma coluna;
- **AVG** — recupera a média de determinado valor.

Os exercícios ilustram usos típicos: recuperar o orçamento do projeto mais caro, o salário mais baixo, as quantidades máxima e mínima de horas alocadas, a média salarial de todo o quadro funcional, a contagem de funcionários de determinado gênero alocados em projetos externos, o total de horas alocadas em um projeto específico, a média de horas em todos os projetos e a idade do funcionário mais velho.

9.2 GROUP BY

Funções de agregação tornam-se muito mais úteis quando combinadas com agrupamento. A cláusula `GROUP BY` divide o conjunto de resultados em grupos e aplica a função a cada grupo separadamente:

```
SELECT count(*), sexo
FROM funcionario
GROUP BY sexo;
```

Nesse caso, a query retorna o número de funcionários de cada gênero, em vez de um único total.

9.3 HAVING

O `HAVING` é um filtro aplicado **dentro** das consultas de agregação, isto é, sobre os grupos formados pelo `GROUP BY`:

```
SELECT count(*), coddepto
FROM funcionario
GROUP BY coddepto
HAVING count(*) > 50;
```

A query agrupa a quantidade de funcionários por departamento e retorna apenas os departamentos com mais de 50 funcionários. A distinção conceitual essencial é: **WHERE filtra linhas antes do agrupamento; HAVING filtra grupos depois do agrupamento**.

Os exercícios pedem, entre outras coisas, o número de funcionários por estado civil em ordem decrescente, os dois estados civis mais frequentes (usando `LIMIT`), a quantidade de funcionários por cargo e o total de horas alocadas por projeto exibindo apenas projetos com mais de 200 horas — este último exigindo `JOIN`, `GROUP BY` e `HAVING` simultaneamente.

9.4 Subconsultas

Uma **subconsulta** é uma consulta dentro de outra consulta. O primeiro padrão apresentado é a subconsulta na cláusula `FROM`, que produz uma tabela derivada:

```sql
SELECT max(totmultas)
FROM (
  SELECT count(*) as totmultas, placa
  FROM ocorrencia
  GROUP BY placa
) AS r;
```

A consulta interna conta multas por placa; a consulta externa toma o máximo desse resultado. Note que a tabela derivada precisa receber um apelido (`r`).

O segundo padrão usa `IN` / `NOT IN` para testar pertencimento a um conjunto produzido por outra consulta. O exemplo busca proprietários que **não** tiveram multas:

```sql
SELECT nomeproprietario
FROM proprietario
WHERE nomeproprietario NOT IN (
  SELECT DISTINCT p.nomeproprietario
  FROM ocorrencia o, carro c, proprietario p
  WHERE o.placa = c.placa
    AND c.idproprietario = p.idproprietario
)
ORDER BY nomeproprietario;
```

Os exercícios aplicam esses padrões ao banco `empresa`: nomes dos funcionários que não se alocaram em nenhum projeto (`NOT IN`), descrição do projeto de maior orçamento (subconsulta escalar), nomes de projetos cuja quantidade de horas de alocação está acima da média histórica (subconsulta com `AVG`) e valor total de orçamento dos projetos que nunca foram alocados.

9.5 Visões

Uma **visão** (*view*) é uma representação **virtual** de dados provenientes de uma ou mais tabelas. Ela **não armazena dados fisicamente**: armazena a consulta, que é executada sempre que a visão é acessada.

```sql
CREATE VIEW nome_da_visao AS
SELECT colunas
FROM tabela_ou_tabelas
WHERE condicoes;
```

As visões cumprem três papéis importantes: encapsulam a complexidade de junções múltiplas, oferecem uma interface estável mesmo quando o esquema subjacente muda e permitem controlar quais colunas cada perfil de usuário enxerga.

9.6 Visões materializadas

Uma **visão materializada** é semelhante a uma visão comum, com uma diferença fundamental: ela **armazena fisicamente** os dados resultantes da consulta. Enquanto a visão regular é apenas uma consulta virtual reexecutada a cada acesso, a materializada cria uma cópia física dos dados em uma tabela, o que pode melhorar significativamente o desempenho de consultas complexas, especialmente em sistemas com grandes volumes.

A contrapartida é a **atualização**: na maioria dos SGBDs a visão materializada não é atualizada automaticamente por padrão, embora seja possível configurar atualização automática ou programada em alguns casos.

```
CREATE MATERIALIZED VIEW nome_da_visao_materializada AS
SELECT colunas
FROM tabelas
WHERE condicoes;

-- Atualização
REFRESH MATERIALIZED VIEW nome_da_visao_materializada;
```

Os exercícios finais pedem a criação de uma visão com todos os dados do funcionário acrescidos de idade, nome do departamento, nome do cargo e salário; uma visão com data de alocação em ordem crescente, nome do funcionário, nome do projeto e horas alocadas; e, como desafio opcional, uma visão materializada com código, nome, descrição, tipo e orçamento do projeto, além do total de funcionários já alocados e do total de horas de alocação — um agregado que justifica plenamente a materialização.

10 Dados não estruturados

Esta aula marca a virada temática da disciplina. O dado de partida é contundente: segundo estimativas da IDC, até 2025 a quantidade de dados criados, capturados, copiados e consumidos no mundo atingirá **175 zettabytes**, e aproximadamente **80% correspondem a dados não estruturados** — e-mails, documentos, vídeos, áudios, posts em redes sociais, relatórios técnicos e artigos científicos.

10.1 Os três tipos de dados

Dados estruturados são organizados em formatos fixos e previsíveis, geralmente em bancos relacionais. Cada registro possui campos bem definidos, tipos específicos e relacionamentos claros. Seu esquema é **rígido e predefinido** (*schema-on-write*). Ferramentas: SQL, Excel, Power BI, Tableau. Vantagens: fácil consulta e agregação, alta consistência, suporte a transações ACID.

Dados semiestruturados não seguem um esquema rígido de tabelas, mas possuem marcadores ou tags que organizam a informação de forma hierárquica ou aninhada. Formatos: JSON, XML, YAML, logs de sistemas. Esquema **flexível e autodescritivo** (*schema-on-read*). Ferramentas: Python com `json` e `xmltodict`, bancos NoSQL como MongoDB. Exemplo típico é um JSON de produto de e-commerce, em que cada item pode ter atributos diferentes (cor, tamanho, voltagem) sem necessidade de alterar o esquema do banco.

Dados não estruturados não possuem formato ou organização predefinida. Representam a forma mais natural de comunicação humana. Formatos: texto livre, imagens, vídeos, áudios, PDFs. Esquema **inexistente** — cada documento é único. Ferramentas: NLP (spaCy, NLTK), Machine Learning (scikit-learn, TensorFlow), OCR (Tesseract). Desafios: ambiguidade linguística, ruído, variabilidade de formato e necessidade de contexto para interpretação.

O quadro comparativo consolida a distinção pela facilidade de análise: **alta** para estruturados (SQL direto), **média** para semiestruturados (parsing necessário) e **baixa** para não estruturados (NLP necessário). Em volume corporativo, a proporção é de cerca de 20%, 10% e 80% respectivamente.

10.2 Aplicações no contexto brasileiro

Os slides apresentam quatro domínios de aplicação:

- **Petrobras** — milhares de relatórios técnicos, laudos de inspeção e pareceres de engenharia por dia. Desafios: volume massivo em PDF e Word, terminologia técnica especializada, necessidade de antecipar manutenções. Soluções com NLP: extração automática de entidades (equipamentos, datas, severidade), classificação por tipo e criticidade, análise de sentimento para identificar urgência e mineração de texto para padrões de falhas recorrentes.
- **Eletrobras** — licenciamento ambiental, condicionantes regulatórias, estudos de impacto. Documentos de centenas de páginas com múltiplas obrigações e prazos. Soluções: extração automática de condicionantes e prazos, classificação de obrigações, alertas automáticos e sumarização executiva.
- **Mercado financeiro** — comunicados, balanços trimestrais, notícias e relatórios de analistas. Soluções: análise de sentimento para prever movimentos de ações, extração de indicadores de balanços não estruturados, detecção de eventos relevantes (fusões, aquisições) e monitoramento de redes sociais.
- **Pesquisa científica** — mais de 3 milhões de artigos publicados anualmente. Soluções: extração de metodologias e resultados, mapeamento de redes de citações, análise de tendências temáticas, sumarização automática e busca semântica.

10.3 Desafios linguísticos

Ambiguidade e polissemia. A mesma palavra assume significados diferentes conforme o contexto. Os exemplos clássicos em português são "banco" (instituição financeira ou assento público), "manga" (fruta ou parte da roupa), "vela" (objeto de cera, tecido de barco ou verbo velar) e "letra" (caractere, texto de canção ou documento financeiro). Sistemas baseados apenas em palavras isoladas (*bag-of-words*) não conseguem distinguir esses significados; é necessário analisar o contexto completo. Modelos modernos como BERT e GPT capturam essas nuances através de mecanismos de atenção, identificando, por exemplo, que "sacar" e "dinheiro" indicam o sentido financeiro de "banco", enquanto "sentei" e "praça" indicam o assento.

Ruído. Textos reais raramente são perfeitos. Os tipos de ruído catalogados são: erros de digitação, abreviações ("vc", "tb", "pq", "msg"), gírias e neologismos ("textão", "mitou", "lacrou", "crush"), caracteres especiais e emojis, hashtags e menções, e formatação inconsistente (maiúsculas excessivas, espaços múltiplos, pontuação repetida). O impacto é triplo: modelos treinados em texto limpo falham diante de variações não previstas; o vocabulário expande porque cada variação vira um token diferente; e a precisão das análises cai. As soluções são correção ortográfica (PySpellChecker, TextBlob), normalização de texto, dicionários customizados e treinamento em dados ruidosos reais.

Vocabulário especializado. Cada área possui siglas e acrônimos próprios — em medicina, engenharia, finanças e tecnologia. O problema é que a mesma sigla pode ter significados distintos em contextos diferentes: "API" pode ser *Application Programming Interface* em tecnologia ou *Active*

Pharmaceutical Ingredient em farmácia. As soluções são glossários de domínio, treinamento em corpus específicos da área e sistemas de desambiguação baseados em contexto.

Morfologia do português. O português tem morfologia rica: múltiplas conjugações verbais ("comer", "comendo", "comi", "comerei", "comeria", "comesse"), flexão de gênero e número, variações regionais ("aipim", "macaxeira", "mandioca") e colocação pronominal ("me diga" versus "diga-me"). A consequência prática é direta: **modelos treinados em inglês não funcionam bem em português sem adaptação**. As soluções são lematização, stemming, modelos multilíngues (mBERT, XLM-R) e recursos específicos como os modelos `pt_core_news` do spaCy.

10.4 Pré-processamento

Antes de aplicar qualquer algoritmo, é essencial preparar os textos. O pré-processamento transforma texto bruto em formato adequado para análise computacional, começando pela **limpeza** — remoção de elementos que não contribuem para a análise.

10.4.1 Tokenização

Tokenização é o processo de dividir um texto em unidades menores chamadas **tokens**. Os quatro tipos apresentados são:

1. **Por palavras**: "O gato subiu no telhado" produz cinco tokens, um por palavra.
2. **Por subpalavras** (BPE, WordPiece): "inacreditável" produz `["in", "acredit", "ável"]`.
3. **Por caracteres**: "gato" produz `["g", "a", "t", "o"]`.
4. **Por sentenças**: divide o texto em frases completas.

Os desafios práticos incluem contrações (em português, "não" geralmente permanece como token único), hifenização ("guarda-chuva" costuma ser mantido como um token para preservar semântica) e pontuação (abreviações como "Dr." devem manter o ponto, exigindo regras específicas).

As ferramentas disponíveis são **NLTK** (com `word_tokenize()` e `sent_tokenize()`, ideal para projetos educacionais e prototipagem), **spaCy** (tokenização rápida e precisa, integrada a um pipeline completo, adequada para produção) e **Hugging Face Tokenizers** (tokenizadores modernos compatíveis com transformers).

A escolha do tipo depende da tarefa: palavras para análise de frequência e bag-of-words; subpalavras para modelos transformers e vocabulários limitados; caracteres para detecção de idioma, correção ortográfica e textos ruidosos; sentenças para sumarização e segmentação de documentos.

10.4.2 Por que tokens são fundamentais

Os slides sintetizam com uma metáfora: *se o texto é uma molécula, o token é o átomo.* Cinco razões justificam a centralidade dos tokens:

1. **Unidade computacional básica** — computadores não processam "texto", processam sequências de tokens, cada um identificado por um ID único no vocabulário.
2. **Base para representação numérica** — tokens são convertidos em *embeddings*, vetores numéricos que capturam significado semântico. Sem tokenização, não há embeddings.

3. **Controle de vocabulário** — a estratégia de tokenização determina o tamanho do vocabulário e a capacidade de lidar com palavras raras.
4. **Limitação de contexto** — modelos têm limite de tokens; exceder esse limite significa perder informação ou dividir o texto.
5. **Custo computacional e precificação** — APIs de modelos de linguagem cobram **por token**. Otimizar tokenização é economizar recursos.

O quadro comparativo das estratégias é instrutivo. **Palavras**: intuitivas e preservam significado completo, mas geram vocabulário enorme e palavras raras viram tokens desconhecidos. **Subpalavras**: vocabulário moderado (30 a 50 mil), lidam bem com palavras raras e capturam morfologia, mas são menos intuitivas e produzem sequências mais longas. **Caracteres**: vocabulário mínimo (cerca de 100), nunca encontram token desconhecido e são robustas a erros ortográficos, mas geram sequências muito longas e perdem semântica.

10.4.3 Stopwords

Stopwords são palavras muito frequentes que geralmente não carregam significado semântico relevante: artigos, preposições, conjunções, pronomes e alguns advérbios. Em português: "o", "a", "de", "em", "para", "e", "ou", "mas", "que", "muito", "mais", "já".

O exemplo prático mostra o efeito: a frase "O processamento de linguagem natural é uma área muito importante da inteligência artificial", com 14 palavras, reduz-se a "processamento linguagem natural área importante inteligência artificial" — 7 palavras, uma redução de 50%.

Mas a remoção nem sempre é desejável. **Remover** quando o objetivo é análise de frequência, classificação de texto, topic modeling ou busca e recuperação. **Manter** em análise de sentimento (a diferença entre "não gostei" e "gostei" está justamente na stopword), tradução automática, geração de texto e modelos contextuais como BERT e GPT, que já capturam contexto por conta própria.

10.4.4 Lematização

Lematização reduz palavras à sua forma canônica (o **lema**) usando análise morfológica e dicionários. Diferentemente do stemming, sempre produz palavras válidas do idioma. Exemplos: "correndo", "correu" e "correrá" reduzem-se a "correr"; "gatos", "gata" e "gatas" a "gato"; "melhores" a "bom".

O processo tem três etapas: análise morfológica para identificar a classe gramatical, consulta ao dicionário para buscar a forma canônica no léxico e retorno de uma palavra real do idioma.

```
import spacy

nlp = spacy.load("pt_core_news_sm")
doc = nlp("correndo rapidamente")
for token in doc:
    print(token.text, "->", token.lemma_)
# correndo -> correr
# rapidamente -> rapidamente
```

As ferramentas são spaCy (lematizador robusto para português, nos modelos `pt_core_news_sm`, `md` e `lg`), NLTK (`WordNetLemmatizer`, melhor para inglês e limitado em português) e Stanza. A lematização é ideal para análise semântica, busca por significado, classificação e análise de sentimento, e deve ser evitada quando performance é crítica.

10.4.5 Stemming

Stemming é uma técnica baseada em regras heurísticas que remove sufixos para reduzir variações morfológicas à raiz (*stem*). É mais rápido que a lematização, mas menos preciso: "correndo" vira "corr", "rapidamente" vira "rapid", "desenvolvimento" vira "desenvolv", "compreensão" vira "compre".

Os algoritmos populares são o **Porter Stemmer** para inglês e o **RSLP Stemmer** (Removedor de Sufixos da Língua Portuguesa), adaptado à morfologia do português.

```
from nltk.stem import RSLPStemmer

stemmer = RSLPStemmer()
print(stemmer.stem("correndo"))          # corr
print(stemmer.stem("desenvolvimento"))   # desenvolv
```

O exemplo comparativo é esclarecedor. Para a entrada "estudando, estudou, estudará", o stemming produz "estud", "estud", "estudar" — inconsistente, pois a forma futura não foi reduzida corretamente. A lematização produz "estudar", "estudar", "estudar" — consistente, todas as formas reduzidas ao infinitivo.

A regra prática: para protótipos ou corpus pequenos, comece com lematização; para big data ou alta performance, opte por stemming; em muitos casos, testar ambas e comparar resultados é a melhor abordagem.

10.5 Evolução histórica do NLP

Os slides organizam a trajetória em cinco eras:

- **1950-1980 — Regras manuais.** Linguistas criavam regras gramaticais manualmente. Sistemas como ELIZA (1966) e SHRDLU (1970) usavam padrões fixos. Limitados, mas pioneiros.
- **1980-2000 — Abordagens estatísticas.** Modelos probabilísticos ganham força: n-gramas, Hidden Markov Models. Foco em corpus grandes e frequências.
- **2000-2013 — Machine learning clássico.** SVM, Naive Bayes e árvores de decisão dominam, com engenharia manual de features (TF-IDF, bag-of-words).
- **2013-2017 — Deep learning e representações distribuídas.** Word2Vec (2013) revoluciona embeddings; RNNs e LSTMs capturam sequências; surge o mecanismo de atenção.
- **2017 até o presente — Transformers e modelos foundation.** Os Transformers (2017) eliminam a recorrência; BERT (2018) e GPT (2019-2023) dominam; pré-treinamento em escala massiva seguido de fine-tuning.

10.6 Transformers e self-attention

A limitação crítica de RNNs e LSTMs era o **processamento sequencial obrigatório**, que impede paralelização e torna o treinamento lento. Os Transformers eliminam essa restrição com o mecanismo de **self-attention**: cada palavra "atende" a todas as outras simultaneamente, tornando o modelo totalmente paralelizável.

O mecanismo funciona assim: cada palavra gera três vetores — **Query (Q)**, **Key (K)** e **Value (V)**. A atenção calcula a similaridade entre o Q de uma palavra e os K de todas as outras, gerando pesos que ponderam os V. A formulação apresentada nos slides é a fórmula canônica do artigo *Attention is All You Need*, em que o produto QK^T é escalado pela raiz da dimensão das chaves e passado por uma softmax antes de ponderar V.

O exemplo prático: na frase "O banco quebrou ontem", o self-attention permite que "banco" atenda simultaneamente a "quebrou" e "ontem", capturando que se trata de instituição financeira e não de assento de praça.

As vantagens são paralelização massiva em GPUs e TPUs, captura de relações globais entre palavras, escalabilidade para bilhões de parâmetros e o fato de servirem de base para BERT, GPT, T5 e todos os LLMs modernos. A limitação apontada é a complexidade quadrática da atenção em relação ao comprimento da sequência.

10.7 Transfer learning e as três famílias de modelos

Transfer learning consiste em pré-treinar modelos em bilhões de palavras (Wikipedia, livros, web) e depois fazer *fine-tuning* para tarefas específicas com poucos dados, reduzindo custo e tempo de treinamento de meses para horas.

Três famílias dominam:

- **BERT** (Google, 2018) — *encoder* bidirecional, lê o contexto à esquerda e à direita simultaneamente. Pré-treinamento por **Masked Language Modeling**: prediz palavras mascaradas. Melhor para classificação de texto, reconhecimento de entidades nomeadas, question answering e análise de sentimento.
- **GPT** (OpenAI, 2018-2023) — *decoder* autoregressivo, gera texto palavra por palavra da esquerda para a direita. Pré-treinamento por **Next Token Prediction**. Melhor para geração de texto, completamento de código, escrita criativa e chatbots.
- **T5** (Google, 2019) — *encoder-decoder* completo, com framework **text-to-text**: toda tarefa é formulada como geração de texto. Pré-treinamento por **Span Corruption**. Melhor para tradução, sumarização e abordagens multitarefa unificadas.

10.8 Ecossistema Python para NLP

Cinco bibliotecas fundamentais são apresentadas:

- **NLTK** — biblioteca clássica e educacional, com corpora, tokenizadores, stemmers e algoritmos tradicionais. Melhor para ensino, pesquisa acadêmica e pré-processamento básico.

- **spaCy** — biblioteca industrial otimizada para produção, extremamente rápida, com pipelines pré-treinados para mais de 60 idiomas, NER e dependency parsing. Melhor para aplicações em produção e processamento em larga escala.
- **Hugging Face Transformers** — acesso a milhares de modelos pré-treinados com interface unificada para transfer learning. Melhor para estado da arte e fine-tuning.
- **Gensim** — especializada em topic modeling e word embeddings; implementa Word2Vec, FastText e LDA. Melhor para topic modeling e análise semântica.
- **scikit-learn** — machine learning clássico, com vetorização de texto (TF-IDF, CountVectorizer) e classificadores. Melhor para classificação de texto e extração de features, oferecendo baselines interpretáveis que treinam em minutos.

10.9 Pipeline completo de análise de textos

O exemplo integrador é a análise de 10 mil avaliações de clientes de um e-commerce, organizada em cinco etapas: **coleta** (API do e-commerce, com texto, nota e data), **pré-processamento** (limpeza de HTML, tokenização, remoção de stopwords, lematização, normalização), **análise** (frequência de palavras, TF-IDF, bigramas e trigramas), **modelagem** (análise de sentimento, topic modeling com LDA, NER, classificação) e **visualização** (nuvem de palavras, gráficos de sentimento, dashboard, relatório executivo).

```
import pandas as pd
from transformers import pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# 1. Coleta
df = pd.read_csv('avaliacoes_clientes.csv')

# 2. Pre-processamento
df['texto_limpo'] = df['avaliacao'].str.lower()

# 3. Analise: TF-IDF
tfidf = TfidfVectorizer(max_features=100)
tfidf_matrix = tfidf.fit_transform(df['texto_limpo'])

# 4. Modelagem: analise de sentimento
sentiment_analyzer = pipeline('sentiment-analysis')
df['sentimento'] = df['texto_limpo'].apply(
    lambda x: sentiment_analyzer(x[:512])[0]['label'])

# 5. Visualizacao
wordcloud = WordCloud(width=800, height=400).generate(
    ' '.join(df['texto_limpo']))
```

Os resultados relatados: 68% de avaliações positivas, 22% neutras e 10% negativas, com cinco tópicos principais identificados (entrega, qualidade, preço, atendimento e embalagem). Os insights acionáveis: 80% das avaliações negativas mencionam “atraso na entrega”; clientes elogiam a qualidade mas reclamam do preço; a ação recomendada é priorizar logística e criar uma linha econômica.

10.10 Práticas em Python

Análise de frequência. O exercício usa um trecho do discurso "Eu Tenho um Sonho" traduzido para o português:

```
import nltk
from collections import Counter
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Passo 1: tokenizacao
tokens = word_tokenize(texto.lower(), language='portuguese')

# Passo 2: manter apenas palavras alfabeticas
palavras = [t for t in tokens if t.isalpha()]

# Passo 3: remover stopwords
stop_words = set(stopwords.words('portuguese'))
palavras_filtradas = [p for p in palavras if p not in stop_words]

# Passo 4 e 5: contar e exibir as 10 mais frequentes
frequencia = Counter(palavras_filtradas)
for palavra, freq in frequencia.most_common(10):
    print(palavra, freq)
```

O resultado destaca "sonho" e "tenho" com três ocorrências cada, refletindo o tema central do discurso. Stopwords como "de", "um" e "em" foram removidas, revelando as palavras com significado real. Os quatro conceitos-chave consolidados são tokenização, stopwords, `Counter` e normalização.

Nuvem de palavras. Serve para comunicar insights visualmente em apresentações e relatórios executivos:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.corpus import stopwords

stop_words = set(stopwords.words('portuguese'))

wordcloud = WordCloud(
    width=1200, height=600,
    background_color='white',
    stopwords=stop_words,
    max_words=50,
    colormap='viridis',
    relative_scaling=0.5,
    min_font_size=10
).generate(texto)

plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
wordcloud.to_file('nuvem_palavras.png')
```

As limitações são explicitadas: a nuvem não mostra contexto, ignora relações entre palavras, pode ser enganosa porque o tamanho exagera diferenças, e não substitui análise quantitativa. A recomendação é combiná-la com tabelas de frequência.

Análise de sentimento por léxico. A abordagem mais simples consiste em manter listas de palavras positivas e negativas e computar a diferença entre as contagens:

```
from nltk.tokenize import word_tokenize

palavras_positivas = {'excelente', 'otimo', 'maravilhoso',
                      'perfeito', 'adorei', 'recomendo'}
palavras_negativas = {'pessimo', 'horrivel', 'terrivel',
                      'ruim', 'odiei', 'defeito'}

def analisar_sentimento(texto):
    tokens = word_tokenize(texto.lower(), language='portuguese')
    score_pos = sum(1 for p in tokens if p in palavras_positivas)
    score_neg = sum(1 for p in tokens if p in palavras_negativas)
    score = score_pos - score_neg
    if score > 0:
        return "POSITIVO", score
    elif score < 0:
        return "NEGATIVO", score
    return "NEUTRO", score
```

As limitações são substanciais e devem ser compreendidas: o método **não captura negações** ("não é bom" seria classificado como positivo), não detecta sarcasmo e ignora contexto. A acurácia típica fica entre 60% e 75%. As melhorias sugeridas são léxicos maiores como o SentiLex-PT, tratamento de intensificadores ("muito bom") e tratamento explícito de negações. O desafio proposto é evoluir para VADER ou TextBlob, buscando acurácia acima de 85%.

10.11 Desafios éticos e fronteiras de pesquisa

Três desafios técnicos atuais são apontados: **viés e fairness** (modelos treinados em dados históricos reproduzem e amplificam preconceitos sociais, como a associação automática entre "médico" e gênero masculino), **explicabilidade** (modelos de deep learning são caixas-pretas, o que gera problemas de confiança, dificuldade de depuração e conflito com requisitos regulatórios da LGPD e do GDPR) e **eficiência** (modelos muito grandes requerem recursos computacionais massivos, com custo ambiental, barreira financeira de entrada e latência em aplicações reais).

As tendências futuras incluem **multimodalidade** (integração de texto, imagem, áudio e vídeo em modelos unificados), **few-shot learning** (aprender novas tarefas com pouquíssimos exemplos via prompting), **NLP para idiomas de baixo recurso**, **RAG (Retrieval-Augmented Generation)**, modelos pequenos e eficientes obtidos por destilação e quantização, aprendizado contínuo e raciocínio passo a passo (*chain-of-thought*).

Os **três pilares da IA responsável** em NLP são **privacidade** (modelos memorizam dados de treino, com risco de vazamento; boas práticas incluem conformidade com LGPD e GDPR, anonimização, privacidade diferencial e minimização de coleta), **viés e equidade** (auditoria de viés em datasets, balanceamento de representação, métricas de fairness e equipes diversas) e **transparência** (explicabilidade com LIME e SHAP, documentação completa via model cards, rastreabilidade e comunicação clara). Os princípios éticos enunciados são beneficência, não-maleficência, autonomia e justiça.

11 Dados Não Estruturados 2

Material de slides não disponível para esta aula.

Os arquivos de apoio registrados no Classroom indicam claramente a direção desta aula: apresentações sobre *Processamento de linguagem natural* e *NLP Representações e classificação*, além de notebooks sobre **Bag of Words**, **TF-IDF e Similaridade**, um **Laboratório de Análise de Sentimentos** e uma análise de sentimentos aplicada a resenhas de filmes, apoiada no dataset `imdb-reviews-pt-br.csv`.

Os conceitos correspondentes a esse material aparecem, no contexto disponível, nos slides da aula seguinte, e são tratados na seção "Extração de Informação de Imagens" a seguir — em particular a passagem de representações por frequência (Bag of Words e TF-IDF) para representações contextuais (embeddings), e as duas grandes operações que essas representações viabilizam: cálculo de similaridade e classificação de texto.

12 Extração de Informação de Imagens

Material de slides não disponível especificamente sobre extração de informação de imagens. Os slides desta aula (`NLP Representações e classificação.pdf`) tratam de representações vetoriais de texto e word embeddings. Os arquivos de apoio incluem notebooks sobre representação de palavras com **FastText**, vetores pré-treinados **GloVe** e tarefas de NLP com modelos, além de um estudo sobre um corpus jornalístico. O conteúdo efetivamente registrado é apresentado a seguir.

12.1 O problema central: texto como números

A questão fundamental é enunciada logo no início: **por que precisamos transformar texto em números?** Porque algoritmos de machine learning só entendem matemática. A pergunta que decorre disso é qual a melhor forma de representar uma palavra como um número ou como um vetor de números.

O fluxograma da disciplina organiza o trabalho em três blocos: **pré-processamento** (tokenização, limpeza, stop words, lematização, stemming), **representação** (Bag of Words, TF-IDF) e **operações** (similaridade e classificação). O **Text Mining** é definido como o conjunto de técnicas que fornece os filtros para limpar e estruturar o texto antes de qualquer análise — sem isso, nas palavras dos slides, jogaríamos lixo nos modelos.

12.2 Bag of Words

O **Bag of Words** é a representação mais simples. Para cada documento cria-se um vetor em que cada posição corresponde a uma palavra do vocabulário, e o valor é a **contagem** daquela palavra no documento.

Sua limitação é estrutural: usa codificação *one-hot*, em que cada palavra do vocabulário é representada por um bit em um vetor. Se o vocabulário tem 10 mil palavras e "Olá" é a quarta palavra do dicionário, ela é representada por um vetor de 10 mil posições com o valor 1 apenas na quarta. **Informações de contexto não são utilizadas.**

12.3 TF-IDF

O **TF-IDF** resolve parcialmente o problema das palavras muito frequentes. Em vez de contar puro, ele **pondera**: quanto mais rara a palavra no corpus, maior o peso que ela recebe.

O exemplo numérico dos slides é esclarecedor. Se a palavra "de" aparece em todos os 1000 documentos de um corpus, seu IDF é $\log(1000/1000) = 0$ — ela simplesmente some do modelo. Se a palavra "backpropagation" aparece em apenas 2 documentos, seu IDF é $\log(1000/2)$, aproximadamente 6,2 — ela vale muito.

Esse mecanismo faz o TF-IDF descartar automaticamente stopwords sem precisar de uma lista explícita, e ao mesmo tempo destacar os termos discriminativos de cada documento.

12.4 Operações: similaridade e classificação

Com representações numéricas em mãos, duas grandes classes de operações tornam-se possíveis.

Similaridade entre documentos. A medida mais usada é a **similaridade de cosseno**, que mede o ângulo entre dois vetores. Para documentos, o ângulo é mais informativo que a distância euclidiana, pois é insensível ao comprimento do documento. A interpretação é direta:

- Vetores paralelos (ângulo próximo de 0 grau, cosseno próximo de 1): documentos idênticos ou muito similares;
- Vetores ortogonais (ângulo de 90 graus, cosseno igual a 0): documentos sem relação;
- Vetores opostos (ângulo de 180 graus, cosseno igual a -1): documentos com sentidos opostos.

Classificação de texto. Uma vez obtidas as representações numéricas, é possível utilizar **qualquer** algoritmo de classificação já conhecido de mineração de dados. A única diferença está na representação numérica das palavras. As aplicações citadas são análise de sentimentos (classificar avaliações como positivo, neutro ou negativo), análise de opinião (detectar posicionamentos em textos políticos ou jornalísticos) e classificação de spam.

12.5 A limitação das representações por frequência

Aqui está o ponto de virada conceitual da aula. Tanto o Bag of Words quanto o TF-IDF geram **vetores horizontais** em que cada palavra olha apenas para si mesma. Como os slides colocam: " 'Pai' e 'filho' estão em posições completamente diferentes e independentes do vetor. O modelo não sabe que eles têm relação."

Conseguimos extrair a frequência da palavra e saber se ela está ou não contida no texto, mas praticamente nenhum **significado semântico**. Duas palavras sinônimas são tão distantes entre si quanto duas palavras sem qualquer relação.

12.6 Representações distribuídas

A alternativa é a **representação distribuída**. Na representação localizada (*one-hot*), cada conceito ocupa exatamente um bit. Na distribuída, cada conceito é descrito por **múltiplas características ao mesmo tempo** — o exemplo dado é o de uma "bomba" descrita simultaneamente como "perigosa" e "pequena".

A pergunta que abre o caminho é: *e se usarmos um algoritmo de machine learning para gerar os vetores de representação automaticamente, a partir do texto?*

12.7 Word2Vec

Word Embedding é uma das representações mais populares do vocabulário de documentos. É capaz de capturar o contexto de uma palavra em um documento, semelhança semântica e sintática e relação com outras palavras. **Word2Vec** é uma das técnicas mais populares para aprender embeddings, desenvolvida por **Tomas Mikolov** em 2013 no Google, e referenciada nos slides pelo artigo *Distributed Representations of Words and Phrases and their Compositionality* (Mikolov, Sutskever, Chen, Corrado e Dean, NIPS 2013).

Em vez de vetores esparsos de tamanho 100.000 (o tamanho do vocabulário), usam-se **vetores densos de 50 ou 100 dimensões** — muito mais eficientes e ricos. Cada palavra ocupa um ponto no espaço vetorial, e distância e ângulo entre vetores são as medidas de similaridade.

O procedimento geral é:

- Dividir as palavras em tokens;
- Escolher um tamanho de vetor, tipicamente de 50 a 100 dimensões;
- Definir uma **janela de contexto**, isto é, quantas palavras ao redor serão observadas, geralmente de 3 a 5;
- Aplicar um dos dois modelos: **CBOW** ou **Skip-grams**.

CBOW (Continuous Bag of Words) pega n palavras anteriores e n palavras posteriores para determinar a palavra central. O exemplo dos slides usa a frase "O fluminense flamengo é o melhor time do Rio de Janeiro" com janela de 1 token: entradas "O flamengo o melhor" produzem o alvo "é"; entradas "fluminense é melhor time" produzem o alvo "o".

Skip-grams faz o caminho inverso: os pares de palavra central e palavra de contexto são chamados de skip-grams, com distâncias típicas de 3 a 5 posições.

O insight central é que **o modelo é treinado para ser bom nessa tarefa de previsão, e como subproduto do treinamento os pesos internos da rede se tornam os próprios embeddings das palavras**. Os vetores resultantes de CBOW e Skip-grams são geralmente semelhantes, com diferenças sutis e diferenças de tempo computacional.

O algoritmo, resumido nos slides: tokenizar o corpus, escolher um tamanho de vetor (50 ou 100), gerar aleatoriamente os primeiros vetores e percorrer todo o texto gerando vetores para todas as palavras.

12.8 Analogias vetoriais

O resultado mais notável dos embeddings é a possibilidade de fazer **álgebra vetorial com significados**. Os slides apresentam as analogias clássicas:

- rei menos homem mais mulher aproxima-se de rainha;
- caminhou menos caminhando mais nadando aproxima-se de nadou;
- França menos Paris mais Roma aproxima-se de Itália.

A leitura é direta: a diferença entre "king" e "queen" é aproximadamente a mesma que entre "man" e "woman" — **o modelo aprendeu o conceito de gênero**. Paris está para a França assim como Roma está para a Itália — **o modelo aprendeu relações geográficas**. Os testes de linha de base padrão para avaliar embeddings incluem analogias por álgebra vetorial, classificações de similaridade de palavras, tempos verbais e relações país-capital.

12.9 A hipótese distribucional

O fundamento teórico é enunciado explicitamente: os **modelos de espaço vetorial (VSMs)** representam palavras em um espaço vetorial contínuo, onde palavras semanticamente semelhantes são mapeadas para pontos próximos. Todos esses métodos dependem, de uma maneira ou de outra, da **Hipótese da Distribuição**, que afirma que *as palavras que aparecem nos mesmos contextos compartilham significado semântico.*

12.10 O contexto do corpus de treino importa

Um alerta prático fundamental fecha a discussão: **estes modelos levam o contexto do corpus em conta**. Se o modelo é treinado com textos sobre petróleo, esse será o seu contexto; se treinado com tweets, esse será o contexto. O exemplo dado é definitivo: a palavra "LULA" pode ser um campo do pré-sal ou um político, dependendo inteiramente do corpus de treinamento.

Os slides citam três modelos com corpora distintos: **Petroles** (baseado em petróleo), **Wikipedia GloVe** (baseado na Wikipedia) e **Google Twitter** (baseado em tweets). A escolha do modelo pré-treinado é, portanto, uma decisão de domínio, não apenas de desempenho.

12.11 Word Mover's Distance

Com vetores densos, duas medidas de similaridade são usadas:

- **Similaridade de cosseno** — agora calculada sobre vetores densos, muito mais precisa do que com Bag of Words.
- **Word Mover's Distance (WMD)** — mede a distância entre duas frases ou documentos **mesmo quando elas não têm nenhuma palavra em comum**.

O exemplo é o clássico da literatura: “Obama speaks to the media” e “The president greets the press” não compartilham nenhuma palavra, mas são semanticamente muito similares. O WMD calcula quanto cada palavra de um documento precisa “viajar” no espaço vetorial para se tornar uma palavra do outro documento, usando uma **matriz de transportação** que indica quanto a palavra i precisa viajar para chegar à palavra j. O método utiliza justamente os vetores de representação produzidos pelo Word2Vec.

13 Síntese da Disciplina

A disciplina TDP percorre um arco completo que vai da abstração do mundo real até a extração de significado de textos livres, e o que dá unidade a esse percurso é a ideia de **representação**.

Na primeira metade, a representação é **tabular e explícita**. Parte-se de um minimundo descrito em linguagem natural — uma empresa com funcionários e projetos, uma clínica com pacientes e exames, uma corretora com investidores e ações — e chega-se a um esquema formal em três etapas sucessivas de refinamento: o **modelo conceitual** (entidades, atributos, relacionamentos, cardinalidades, especializações), o **modelo lógico** (tabelas, chaves primárias e estrangeiras, tabelas associativas para relacionamentos M:N) e o **modelo físico** (comandos DDL executáveis em um SGBD real). Cada etapa tem regras de mapeamento precisas, e o aluno que domina essas regras consegue traduzir qualquer descrição de negócio em um banco de dados consistente.

Sobre esse esquema, a linguagem SQL é apresentada em camadas cumulativas: **DDL** cria e altera estruturas, com restrições (`PRIMARY KEY`, `FOREIGN KEY`, `CHECK`, `NOT NULL`, `DEFAULT`) que codificam regras de negócio diretamente no banco; **DML** insere, atualiza e remove dados, com a advertência recorrente sobre o poder destrutivo de um `WHERE` esquecido; **DQL** consulta, filtrando com `WHERE` e `LIKE`, ordenando com `ORDER BY` e eliminando duplicatas com `DISTINCT`; os **joins** recompõem a informação que a normalização havia distribuído, com a distinção crítica entre junções internas (que descartam linhas sem correspondência) e externas (que as preservam); e finalmente as **funções de agregação com GROUP BY e HAVING**, as **subconsultas** e as **visões** elevam a consulta ao patamar analítico.

Na segunda metade, a representação torna-se **vetorial e aprendida**. O ponto de partida é o reconhecimento de que a maior parte dos dados corporativos — cerca de 80% — não cabe em tabelas. Textos exigem um pipeline próprio: limpeza, tokenização, remoção de stopwords, lematização ou stemming. Em seguida vem a conversão em números, e aqui a disciplina traça uma progressão histórica e conceitual clara. **Bag of Words** conta palavras, mas trata cada uma como um símbolo isolado. **TF-IDF** pondera pela raridade, descartando automaticamente termos onipresentes e destacando os discriminativos, mas continua sem noção de significado. **Word embeddings** (Word2Vec, com suas variantes CBOW e Skip-grams) rompem essa barreira: ao treinar uma rede para prever palavras a partir de seu contexto, os pesos internos tornam-se representações densas em que a proximidade geométrica corresponde à proximidade semântica. É o que torna possível a álgebra de analogias e a Word Mover’s Distance entre frases sem palavras em comum. E os **Transformers**, com o mecanismo de self-attention, levam essa lógica adiante, produzindo representações que dependem do contexto específico de cada ocorrência.

As duas metades conversam de forma mais profunda do que parece. Ambas enfrentam o mesmo problema de fundo: **como impor estrutura ao mundo para que ele possa ser computado**. A modelagem E-R faz isso por decisão humana explícita; os embeddings fazem isso por aprendizado

estatístico a partir de dados. Ambas enfrentam o problema da ambiguidade: no banco relacional, ela é resolvida por chaves e restrições de integridade; em NLP, pelo contexto e pelos mecanismos de atenção. E ambas ensinam a mesma lição sobre corpus e domínio: um banco de dados só é útil para o mini-mundo que modela, assim como um embedding treinado em textos de petróleo entenderá "LULA" de forma completamente diferente de um treinado em tweets.

O curso também não se esquiva das questões práticas e éticas. Reforça repetidamente a importância da integridade referencial e do cuidado com comandos destrutivos; alerta para as limitações reais das técnicas simples de NLP, como a incapacidade de léxicos de sentimento lidarem com negações e sarcasmo, com acurácia típica de apenas 60% a 75%; e dedica espaço explícito aos três pilares da IA responsável — privacidade, equidade e transparência — com referência direta à LGPD e ao GDPR.

O trabalho final, o *Case Negociações na Bolsa de Valores*, é a síntese natural desse percurso na dimensão relacional: exige modelar entidades com identificadores alternativos, resolver um relacionamento M:N com atributos, tratar dados históricos de séries temporais e distinguir dados transacionais de dados derivados. Quem o realiza corretamente demonstrou dominar todo o encadeamento conceitual-lógico-físico que a disciplina construiu aula após aula.

Ao final, a mensagem enunciada nas competências da primeira aula ganha sentido pleno: **SQL organiza e estrutura os dados, a IA os interpreta e gera recomendações, e juntos eles transformam dados em ações inteligentes.**