AULA SBD · APOSTILA DE ESTUDO

Banco de Dados I e II

Apostila completa cobrindo do modelo conceitual ao Data Lake — modelagem, SQL, transações, projeto físico, NoSQL e engenharia de dados.

SUMÁRIO

1 Conceitos Fundamentais

Dado x Informação

Dado é um valor bruto, sem contexto — ex: `32`. **Informação** é o dado processado e contextualizado, que gera conhecimento — ex: "a temperatura hoje é 32°C". Um banco de dados armazena dados; é a aplicação (ou a consulta) que os transforma em informação.

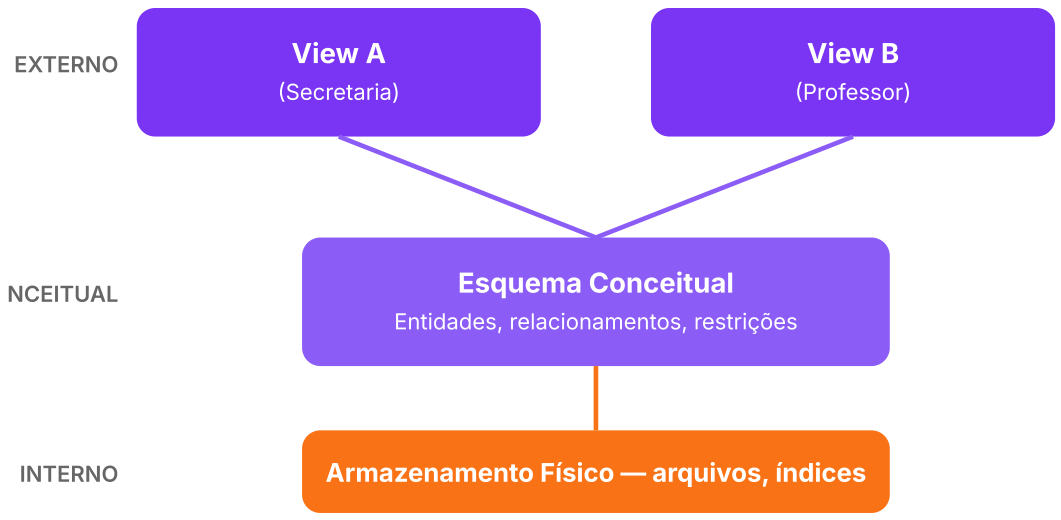
Banco de Dados e SGBD

> **Banco de Dados (BD):** coleção de dados inter-relacionados, persistentes, que representam informações sobre um domínio do mundo real (ex: uma universidade, uma loja).

> **SGBD (Sistema Gerenciador de Banco de Dados):** software que permite definir, criar, manter e controlar o acesso ao banco de dados. É a camada entre a aplicação/usuário e os arquivos físicos de dados.

Arquitetura ANSI/SPARC (3 níveis)

Nível	O que representa	Quem enxerga
Externo (Views)	Visões personalizadas dos dados para cada grupo de usuário	Usuários finais / aplicações
Conceitual	Esquema lógico completo: entidades, relacionamentos, restrições	DBA / projetistas
Interno (Físico)	Como os dados são armazenados fisicamente: arquivos, índices	SGBD / sistema operacional



Arquitetura ANSI/SPARC: cada camada isola a de cima das mudanças na de baixo.

Independência de Dados

- **Independência lógica:** mudanças no esquema conceitual não afetam os programas/visões externas.

- **Independência física:** mudanças na forma de armazenamento (ex: trocar um índice) não afetam o esquema conceitual.

Tipos de Usuários

- **DBA (Database Administrator):** administra o SGBD — segurança, performance, backup.
- **Projetistas de BD:** definem o esquema conceitual (DER, normalização).
- **Desenvolvedores de aplicação:** escrevem programas que acessam o BD.
- **Usuários finais:** casuais (consultas eventuais), paramétricos (operações repetitivas, ex: caixa de banco), sofisticados (analistas que escrevem SQL complexo).

> **Responsabilidades típicas do DBA:**
>
> - Definir e alterar o esquema do banco (DDL).
> - Gerenciar permissões de acesso (criar usuários, papéis, conceder/revogar privilégios).
> - Monitorar e otimizar performance (índices, planos de execução, configuração de memória).
> - Planejar e testar rotinas de backup e recuperação de desastres.
> - Garantir integridade e conformidade (auditoria, LGPD/compliance).

Exemplo prático dos 3 níveis ANSI/SPARC

Imagine um sistema de matrícula universitária:

- **Nível interno:** a tabela `matricula` está fisicamente armazenada em arquivos particionados por ano, com um índice B-Tree em `aluno_id`.
- **Nível conceitual:** o esquema define as entidades `Aluno`, `Disciplina`, `Matricula` e seus relacionamentos — igual para todos os sistemas que usam o banco.
- **Nível externo:** a *secretaria* enxerga uma view com dados financeiros do aluno; o *professor* enxerga apenas a lista de alunos matriculados na sua disciplina. Duas visões diferentes do mesmo esquema conceitual.

Se o DBA decidir trocar o índice B-Tree por particionamento por hash (mudança no nível interno), nem o esquema conceitual nem as views dos usuários precisam mudar — isso é **independência física** na prática.

Vantagens do SGBD sobre Sistemas de Arquivos

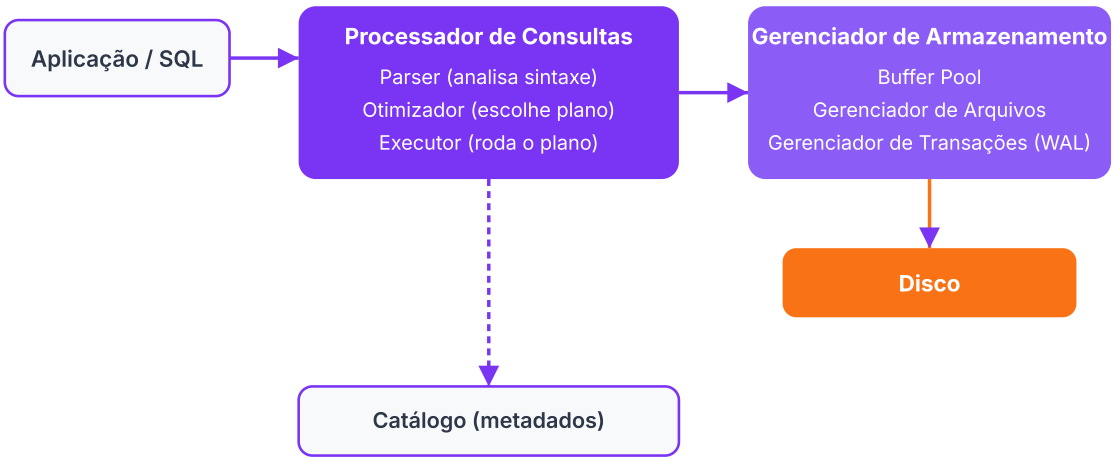
Controle de redundância | Integridade | Segurança e controle de acesso | Concorrência | Backup e recuperação | Independência de dados | Linguagem de consulta padronizada (SQL)

Em um sistema de arquivos tradicional, cada aplicação mantém seus próprios arquivos: o sistema de RH tem um arquivo com os dados do funcionário, e o sistema de folha de pagamento tem *outro* arquivo, quase igual, com o mesmo funcionário. Isso gera **redundância** (dado duplicado), risco de **inconsistência** (atualizar em um lugar e esquecer o outro) e nenhuma forma padronizada de controlar quem acessa o quê. O SGBD centraliza os dados, elimina essa duplicação e aplica as regras de integridade e segurança em um único lugar.

2 SGBDs

Arquitetura Interna de um SGBD

- **Processador de consultas (Query Processor):** parser → otimizador → executor. Recebe SQL, valida, escolhe o melhor plano de execução e o executa.
- **Gerenciador de armazenamento (Storage Manager):** controla acesso a disco, buffer, índices e arquivos de dados.
- **Gerenciador de transações:** garante ACID (ver Módulo 9).



Fluxo de uma consulta: da aplicação até o disco, passando pelo otimizador e pelo buffer pool.

Catálogo do Sistema (Dicionário de Dados)

Metadados sobre o próprio banco: tabelas, colunas, tipos, índices, chaves, permissões e views. É consultado pelo otimizador e por comandos como `information_schema` (SQL padrão) ou `pg_catalog` (PostgreSQL).

Buffer Pool

Área de memória RAM que mantém páginas de dados lidas do disco, evitando I/O repetido — acessar RAM é ordens de magnitude mais rápido que acessar disco. Cada página fica em memória até precisar ser substituída.

- **LRU (Least Recently Used):** descarta a página menos recentemente acessada.
- **Clock/Second-chance:** aproximação eficiente de LRU usada por SGBDs reais (ex: PostgreSQL usa uma variante chamada *clock-sweep*).
- **Dirty pages:** páginas alteradas em memória mas ainda não gravadas em disco — precisam ser persistidas (flush) antes de serem substituídas, ou o dado se perde.

Quanto maior o buffer pool configurado, menos I/O de disco — por isso é um dos primeiros parâmetros que um DBA ajusta ao tunar performance (ex: `shared_buffers` no PostgreSQL, `innodb_buffer_pool_size` no MySQL).

Recuperação: Write-Ahead Log (WAL) e Checkpoints

Antes de alterar uma página de dados em disco, o SGBD primeiro grava a intenção da mudança em um **log sequencial** (WAL). Se o servidor cair no meio de uma escrita, na reinicialização o SGBD relê o log e:

- **Refaz (redo)** as transações que foram commitadas mas cujas páginas não chegaram a ser gravadas em disco.
- **Desfaz (undo)** as transações que estavam em andamento e não foram commitadas.

Um **checkpoint** é um ponto em que o SGBD garante que todas as páginas sujas até aquele momento foram gravadas em disco — isso limita quanto log precisa ser relido na recuperação, acelerando o processo.

Controle de Concorrência e Segurança

- **Controle de concorrência:** mecanismos (locks, MVCC) para múltiplas transações simultâneas sem corromper dados — aprofundado no Módulo 9.
- **Segurança:** autenticação (quem é o usuário), autorização (`GRANT` / `REVOKE`), papéis (roles) e criptografia (em trânsito com TLS, em repouso com encryption at rest).

```
-- Criar um usuário e conceder permissões restritas
CREATE ROLE analista LOGIN PASSWORD 'senha_forte';
GRANT SELECT ON vendas TO analista;
GRANT INSERT, UPDATE ON estoque TO analista;
REVOKE DELETE ON estoque FROM analista;
```

Principais SGBDs do Mercado

SGBD	Tipo / Licença	Uso típico
MySQL	Relacional, open source	Aplicações web, startups
PostgreSQL	Relacional, open source, muito completo	Aplicações robustas, dados geoespaciais, JSON
Oracle	Relacional, comercial	Grandes corporações, missão crítica
SQL Server	Relacional, comercial (Microsoft)	Ambientes corporativos Windows/.NET

3 Modelagem Conceitual (DER)

O **Diagrama Entidade-Relacionamento (DER)** é a ferramenta gráfica para representar o esquema conceitual de um banco de dados antes de qualquer implementação.

Entidades e Atributos

- **Entidade forte:** existe por si só, tem chave primária própria (ex: *Cliente*).
- **Entidade fraca:** depende existencialmente de outra entidade; possui apenas uma chave parcial (ex: *Dependente* de um *Funcionário*).
- **Atributo simples x composto:** *Idade* (simples) x *Endereço* = rua + número + cidade (composto).
- **Atributo monovalorado x multivalorado:** *CPF* (um valor) x *Telefones* (vários valores).
- **Atributo derivado:** calculado a partir de outro, ex: *Idade* a partir de *Data de Nascimento*.
- **Atributo chave:** identifica unicamente uma ocorrência da entidade.

Relacionamentos e Cardinalidade

Cardinalidade	Significado	Exemplo
1:1	Uma ocorrência de A se relaciona com no máximo uma de B	Pessoa — Passaporte
1:N	Uma ocorrência de A se relaciona com várias de B	Departamento — Funcionário
N:M	Várias ocorrências de A se relacionam com várias de B	Aluno — Disciplina

Participação total: toda ocorrência da entidade participa obrigatoriamente do relacionamento.
Participação parcial: a participação é opcional.

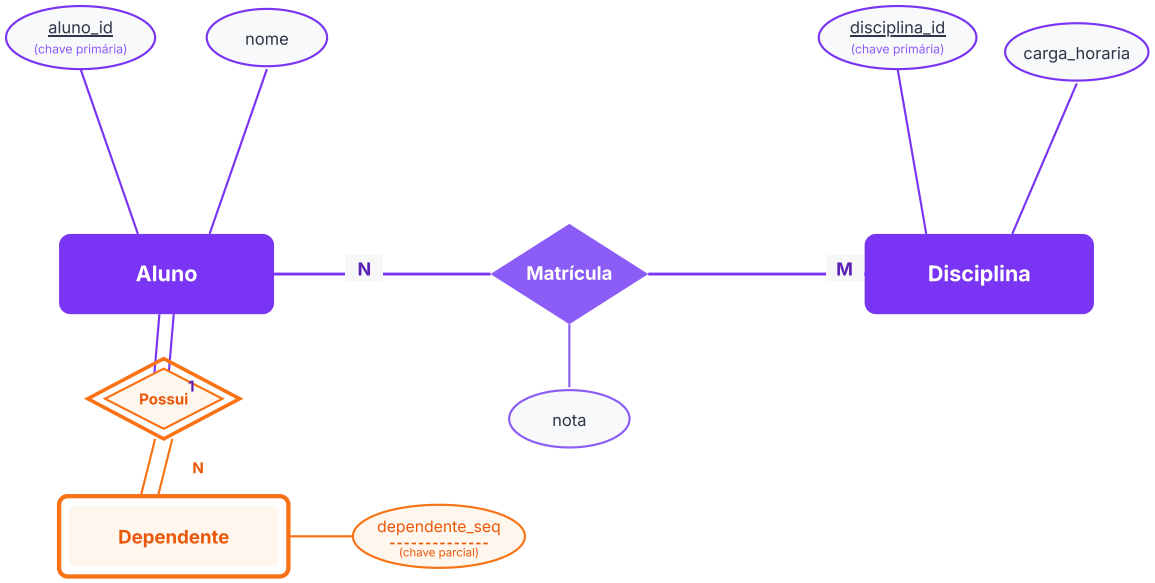
Generalização e Especialização

Modelam hierarquias "é um" (is-a): uma superclasse genérica (ex: *Veículo*) e subclasses especializadas (ex: *Carro*, *Moto*) que herdam os atributos da superclasse e podem ter atributos próprios.

- **Total x parcial:** toda instância da superclasse deve (total) ou não precisa (parcial) pertencer a uma subclasse.
- **Disjunta x sobreposta:** uma instância pode pertencer a apenas uma subclasse (disjunta) ou a várias ao mesmo tempo (sobreposta).

Exemplo completo de DER

Cenário: uma escola quer modelar **Aluno**, **Disciplina** e a matrícula entre eles (N:M), além de **Dependente**, uma entidade fraca de *Aluno* (bolsa-família, por exemplo).



Retângulo = entidade · elipse = atributo (sublinhado contínuo = chave primária, tracejado = chave parcial) · losango = relacionamento · retângulo/losango de borda dupla laranja + linha dupla = entidade fraca com relacionamento identificador de participação total.

Ao mapear para o modelo relacional (Módulo 4): `Matricula` vira uma tabela associativa com `aluno_id` + `disciplina_id` (FKs) e o atributo `nota`; `Dependente` vira uma tabela cuja chave primária é composta por `aluno_id` (FK) + um identificador parcial (ex: `dependente_seq`).

4 Modelo Relacional

Conceitos Básicos

Tabela (relação): conjunto de tuplas com a mesma estrutura de atributos.

Tupla (linha/registro): uma instância/ocorrência da relação.

Atributo (coluna): propriedade da relação.

Domínio: conjunto de valores válidos para um atributo.

Chaves

- **Superchave:** qualquer conjunto de atributos que identifica unicamente uma tupla (pode ter atributos redundantes).
- **Chave candidata:** superchave mínima (sem atributos redundantes).
- **Chave primária (PK):** a chave candidata escolhida para identificar as tuplas — não pode ser nula.
- **Chave estrangeira (FK):** atributo que referencia a chave primária de outra (ou da mesma) tabela, implementando relacionamentos.

Mapeamento DER → Relacional

Elemento do DER	Vira no Relacional
Entidade forte	Tabela com PK própria
Entidade fraca	Tabela cuja PK é a chave parcial + FK da entidade forte
Relacionamento 1:N	FK no lado "N" apontando para o lado "1"
Relacionamento N:M	Tabela associativa com FKs para as duas entidades
Atributo multivalorado	Tabela separada referenciando a entidade original

Restrições de Integridade

- **Integridade de entidade:** PK não pode ser nula nem duplicada.
- **Integridade referencial:** toda FK deve corresponder a uma PK existente (ou ser nula, se permitido).

- **Integridade de domínio:** valores devem respeitar o tipo/domínio definido (ex: `CHECK (idade ⩾ 0)`).

Do DER ao DDL — exemplo completo

Continuando o exemplo do Módulo 3 (Aluno / Disciplina / Matricula):

```
CREATE TABLE aluno (
  id            SERIAL PRIMARY KEY,
  nome          VARCHAR(120) NOT NULL,
  data_nasc     DATE NOT NULL
);

CREATE TABLE disciplina (
  id              SERIAL PRIMARY KEY,
  nome            VARCHAR(120) NOT NULL,
  carga_horaria   INT NOT NULL CHECK (carga_horaria > 0)
);

-- Relacionamento N:M vira tabela associativa
CREATE TABLE matricula (
  aluno_id       INT NOT NULL REFERENCES aluno(id),
  disciplina_id  INT NOT NULL REFERENCES disciplina(id),
  nota           DECIMAL(4,2),
  PRIMARY KEY (aluno_id, disciplina_id)
);

-- Entidade fraca: PK composta = FK do "dono" + chave parcial
CREATE TABLE dependente (
  aluno_id          INT NOT NULL REFERENCES aluno(id) ON DELETE CASCADE,
  dependente_seq    INT NOT NULL,
  nome              VARCHAR(120) NOT NULL,
  PRIMARY KEY (aluno_id, dependente_seq)
);
```

A tabela `matricula` concentra as duas FKs — é a materialização do relacionamento N:M.

5 Álgebra Relacional

A álgebra relacional é a **base teórica formal do SQL**: um conjunto de operadores que recebem relações (tabelas) e produzem novas relações.

Operadores Unários

Operador	Símbolo	Função	Equivalente em SQL
Seleção	σ (sigma)	Filtra linhas por condição	`WHERE`
Projeção	π (pi)	Seleciona colunas	`SELECT` (lista de colunas)

Operadores de Conjunto (Binários)

Operador	Símbolo	Requisito	Equivalente em SQL
União	∪	Relações compatíveis (mesmas colunas)	`UNION`
Diferença	−	Relações compatíveis	`EXCEPT`
Interseção	∩	Relações compatíveis	`INTERSECT`
Produto cartesiano	×	Nenhum	`CROSS JOIN`

Junções (Joins)

- **Theta join:** junção com condição arbitrária (θ pode ser =, >, < etc).
- **Equijoin:** theta join com condição de igualdade.
- **Natural join (⋈):** equijoin automático pelos atributos de mesmo nome, removendo colunas duplicadas.
- **Outer joins:** mantêm tuplas sem correspondência (left, right, full) — preenchendo com NULL.

Divisão (÷)

Retorna valores de um atributo que se relacionam com *todos* os valores de outro conjunto. Exemplo clássico: "alunos matriculados em *todas* as disciplinas obrigatórias".

Exemplo prático passo a passo

Considere duas relações:

Funcionario

id	nome	depto_id
1	Ana	10
2	Beto	20
3	Caio	10

Departamento

id	nome
10	TI
20	RH

1. Seleção — funcionários do departamento 10:

```
σ(depto_id = 10) (Funcionario)
→ {(1, Ana, 10), (3, Caio, 10)}
```

2. Projeção — apenas os nomes, de todos os funcionários:

```
π(nome) (Funcionario)
→ {Ana, Beto, Caio}
```

3. Junção natural — combinando funcionário com o nome do seu departamento:

```
Funcionario ⋈ Departamento
→ {(1, Ana, 10, TI), (2, Beto, 20, RH), (3, Caio, 10, TI)}
```

Em SQL, essa sequência de operadores algébricos corresponde exatamente a:

```
SELECT f.nome, d.nome AS depto
FROM funcionario f
JOIN departamento d ON f.depto_id = d.id
WHERE f.depto_id = 10;
```

Ou seja: o otimizador do SGBD (Módulo 2) traduz seu `SELECT` em uma árvore de operadores de álgebra relacional (seleção, projeção, junção) e decide a melhor ordem de execução entre eles.

6 SQL

Subconjuntos da Linguagem SQL

Sigla	Nome	Comandos
DDL	Data Definition Language	`CREATE`, `ALTER`, `DROP`, `TRUNCATE`
DML	Data Manipulation Language	`INSERT`, `UPDATE`, `DELETE`
DQL	Data Query Language	`SELECT`
DCL	Data Control Language	`GRANT`, `REVOKE`
TCL	Transaction Control Language	`COMMIT`, `ROLLBACK`, `SAVEPOINT`

JOINs

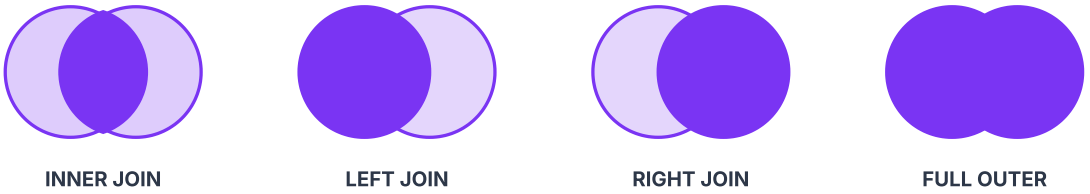
```
-- INNER JOIN: só as linhas com correspondência nas duas tabelas
SELECT a.nome, d.nome AS depto
FROM funcionario a
INNER JOIN departamento d ON a.depto_id = d.id;

-- LEFT JOIN: todas as linhas da esquerda, com NULL se não houver correspondência
SELECT a.nome, d.nome AS depto
FROM funcionario a
LEFT JOIN departamento d ON a.depto_id = d.id;

-- RIGHT JOIN: todas as linhas da direita, com NULL se não houver correspondência
SELECT a.nome, d.nome AS depto
FROM funcionario a
RIGHT JOIN departamento d ON a.depto_id = d.id;

-- FULL OUTER JOIN: união do LEFT com o RIGHT (nada se perde)
SELECT a.nome, d.nome AS depto
FROM funcionario a
FULL OUTER JOIN departamento d ON a.depto_id = d.id;

-- SELF JOIN: uma tabela relacionada a ela mesma (ex: funcionário e seu gestor)
SELECT f.nome AS funcionario, g.nome AS gestor
FROM funcionario f
LEFT JOIN funcionario g ON f.gestor_id = g.id;
```



Área roxa preenchida = linhas retornadas por cada tipo de junção.

Subconsultas

```
-- Subconsulta escalar
SELECT nome, salario
FROM funcionario
WHERE salario > (SELECT AVG(salario) FROM funcionario);

-- Subconsulta correlacionada com EXISTS
SELECT nome
FROM funcionario f
WHERE EXISTS (
  SELECT 1 FROM projeto p WHERE p.responsavel_id = f.id
);

-- IN: compara com uma lista vinda de outra consulta
SELECT nome FROM funcionario
WHERE depto_id IN (SELECT id FROM departamento WHERE nome = 'TI');

-- ANY / ALL: compara com cada valor do conjunto
SELECT nome FROM funcionario
WHERE salario > ALL (SELECT salario FROM funcionario WHERE depto_id = 20);
```

CTEs (Common Table Expressions)

Uma CTE (`WITH`) nomeia uma subconsulta para reuso e legibilidade — muito usada para quebrar consultas complexas em passos, e é a base das **consultas recursivas** (ex: percorrer uma hierarquia de gestores).

```
WITH salario_medio AS (
  SELECT depto_id, AVG(salario) AS media
  FROM funcionario
  GROUP BY depto_id
)
SELECT f.nome, f.salario, s.media
FROM funcionario f
JOIN salario_medio s ON f.depto_id = s.depto_id
WHERE f.salario > s.media;
```

Window Functions (Funções de Janela)

Diferente do `GROUP BY`, que colapsa linhas, uma *window function* calcula um valor agregado **sem perder o detalhe de cada linha**.

```
SELECT
  nome, depto_id, salario,
  RANK()      OVER (PARTITION BY depto_id ORDER BY salario DESC) AS ranking,
  AVG(salario) OVER (PARTITION BY depto_id)                      AS media_depto
FROM funcionario;
```

Funções comuns: `ROW_NUMBER()`, `RANK()`, `DENSE_RANK()`, `LAG()` / `LEAD()` (linha anterior/seguinte), `SUM()` / `AVG()` como janela.

Agregações

```
SELECT depto_id, COUNT(*) AS qtd, AVG(salario) AS media
FROM funcionario
GROUP BY depto_id
HAVING COUNT(*) > 5
ORDER BY media DESC;
```

Funções de agregação: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`. `WHERE` filtra linhas *antes* de agrupar; `HAVING` filtra grupos *depois* de agrupar.

Views e Índices Básicos

```
CREATE VIEW vw_funcionarios_ativos AS
SELECT id, nome, depto_id FROM funcionario WHERE ativo = true;

CREATE INDEX idx_funcionario_depto ON funcionario (depto_id);
```

Uma **view** é uma consulta salva, tratada como tabela virtual (não armazena dados por padrão). Um **índice** acelera buscas e ordenações à custa de espaço extra e overhead em escritas.

Constraints (Restrições de Coluna)

```
CREATE TABLE produto (
  id         SERIAL PRIMARY KEY,
  sku        VARCHAR(20) UNIQUE NOT NULL,
  nome       VARCHAR(120) NOT NULL,
  preco      DECIMAL(10,2) NOT NULL CHECK (preco > 0),
  estoque    INT NOT NULL DEFAULT 0,
  categoria_id INT REFERENCES categoria(id)
);
```

Constraint	Garante
`NOT NULL`	Coluna obrigatória
`UNIQUE`	Nenhum valor duplicado na coluna
`CHECK`	Valor precisa satisfazer uma expressão booleana
`DEFAULT`	Valor padrão quando não informado no `INSERT`
`FOREIGN KEY`	Integridade referencial com outra tabela

7 Projeto e Normalização

Dependência Funcional e Chave Candidata

> **Dependência funcional (X → Y):** o valor de X determina univocamente o valor de Y. Ex: `CPF → Nome`.

Chave candidata: conjunto mínimo de atributos que determina funcionalmente todos os demais atributos da relação.

Formas Normais

Forma	Regra
1FN	Atributos atômicos (sem grupos repetitivos, sem valores multivalorados numa célula)
2FN	1FN + nenhum atributo não-chave depende de *parte* de uma chave primária composta
3FN	2FN + nenhum atributo não-chave depende transitivamente da chave (depende de outro atributo não-chave)
BCNF	Para toda dependência funcional X → Y, X deve ser uma chave candidata (forma normal mais rígida que a 3FN)

Exemplo completo de Normalização

Tabela não normalizada de pedidos (dados reais, mistura tudo em uma linha por pedido):

pedido_id	cliente	endereco	produtos
1001	Ana Silva	Rua A, 100	Mouse (2), Teclado (1)
1002	Beto Costa	Rua B, 200	Monitor (1)

Problema: a coluna `produtos` guarda vários valores numa célula só — viola a atomicidade.

1FN — cada produto vira uma linha (atributos atômicos, sem repetição):

pedido_id	cliente	endereco	produto	quantidade
1001	Ana Silva	Rua A, 100	Mouse	2
1001	Ana Silva	Rua A, 100	Teclado	1
1002	Beto Costa	Rua B, 200	Monitor	1

Problema: a chave agora é `(pedido_id, produto)`, mas `cliente` e `endereco` dependem só de `pedido_id` — *dependência parcial*.

2FN — separa o que depende só de parte da chave:

Pedido

pedido_id	cliente	endereco
1001	Ana Silva	Rua A, 100
1002	Beto Costa	Rua B, 200

ItemPedido

pedido_id	produto	qtd
1001	Mouse	2
1001	Teclado	1
1002	Monitor	1

Problema restante: se `Pedido` também guardasse `cliente_email`, ele dependeria de `cliente` (não da chave `pedido_id` diretamente) — *dependência transitiva*.

3FN — remove a dependência transitiva, isolando `Cliente`:

```
Cliente(cliente_id PK, nome, endereco, email)
Pedido(pedido_id PK, cliente_id FK, data)
ItemPedido(pedido_id FK, produto_id FK, quantidade)  -- PK composta
Produto(produto_id PK, nome, preco)
```

Agora cada tabela guarda fatos sobre *um único assunto* — exatamente o objetivo da normalização: eliminar redundância e evitar anomalias de inserção, atualização e exclusão.

BCNF, 4FN e 5FN (breve)

BCNF é mais rígida que a 3FN: exige que *toda* determinante de uma dependência funcional seja chave candidata (resolve casos raros onde a 3FN ainda permite anomalias com chaves candidatas sobrepostas). **4FN** trata dependências multivaloradas (um atributo independente

gerando combinações redundantes) e **5FN** trata dependências de junção — ambas pouco cobradas na prática, mas costumam aparecer como pergunta teórica em prova.

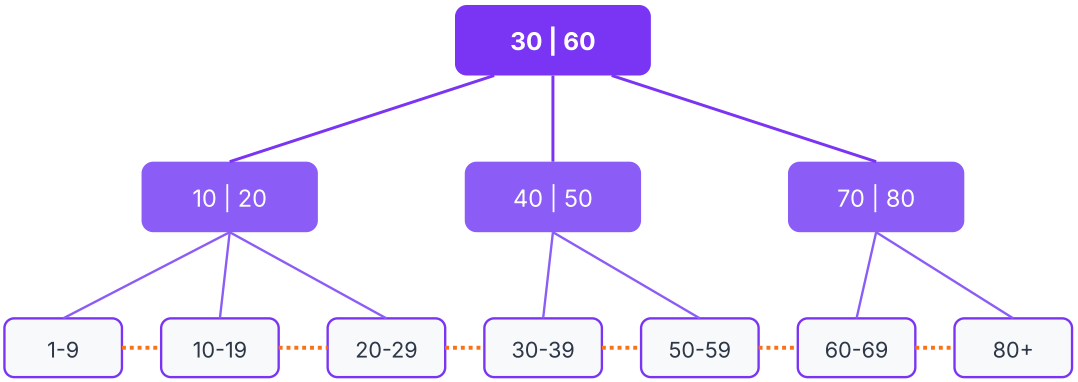
Quando Desnormalizar

Em cenários analíticos (Data Warehouse, relatórios), desnormalizar reduz a quantidade de `JOIN`s e acelera leitura, ao custo de redundância controlada — trade-off aceito nesses contextos (ver Módulo 15). Regra geral: **normalize para OLTP** (muitas escritas, poucas duplicatas), **desnormalize para OLAP** (muitas leituras agregadas).

8 Projeto Físico

Índices: B-Tree x Hash

Tipo	Bom para	Limitação
B-Tree	Igualdade e faixas (`>` , `<` , `BETWEEN`), ordenação	Um pouco mais lento que Hash para igualdade pura
Hash	Somente igualdade (`=`)	Não serve para buscas por faixa ou ordenação



B-Tree: poucos níveis para chegar a qualquer chave (busca em O(log n)); folhas ligadas (laranja) permitem varrer faixas em ordem — por isso serve tanto para `=` quanto para `BETWEEN` / `ORDER BY` .

EXPLAIN / EXPLAIN ANALYZE

```
EXPLAIN ANALYZE
SELECT * FROM pedido WHERE cliente_id = 42;

--                                    QUERY PLAN
-- Index Scan using idx_pedido_cliente on pedido
--   (cost=0.29..8.31 rows=3 width=64) (actual time=0.02..0.03 rows=3 loops=1)
--   Index Cond: (cliente_id = 42)
-- Planning Time: 0.08 ms
-- Execution Time: 0.04 ms
```

Mostra o plano de execução escolhido pelo otimizador: se usou *Seq Scan* (varredura completa, ruim para tabelas grandes) ou *Index Scan* (usa índice), custo estimado, linhas retornadas e tempo real. O otimizador é **cost-based**: usa estatísticas de distribuição de dados (atualizadas por `ANALYZE`) para estimar qual plano é mais barato antes de executar.

Índice Composto e Covering Index

```
-- Índice composto: útil para filtros combinados, na ordem das colunas
CREATE INDEX idx_pedido_cliente_data ON pedido (cliente_id, data_pedido);
-- Bom para: WHERE cliente_id = 42 AND data_pedido > '2026-01-01'
-- Também serve para: WHERE cliente_id = 42 (sozinho)
-- NÃO serve (sozinho) para: WHERE data_pedido > '2026-01-01' (sem cliente_id)

-- Covering index: inclui colunas extras para responder a consulta
-- inteiramente pelo índice, sem tocar a tabela ("index-only scan")
CREATE INDEX idx_pedido_covering ON pedido (cliente_id) INCLUDE (status, total);
```

Particionamento

- **Horizontal:** divide linhas entre partições, por *range* (ex: por data), *hash* ou *list* (por valor categórico).
- **Vertical:** divide colunas entre tabelas/arquivos diferentes (ex: separar colunas pouco acessadas).

```
CREATE TABLE pedido (
  id INT, cliente_id INT, data_pedido DATE, total DECIMAL(10,2)
) PARTITION BY RANGE (data_pedido);

CREATE TABLE pedido_2025 PARTITION OF pedido
  FOR VALUES FROM ('2025-01-01') TO ('2026-01-01');
CREATE TABLE pedido_2026 PARTITION OF pedido
  FOR VALUES FROM ('2026-01-01') TO ('2027-01-01');
```

Uma consulta filtrando por `data_pedido` num período específico só varre a partição relevante (*partition pruning*), em vez da tabela inteira.

Otimização, Tipos de Dados e Armazenamento

- Índices compostos ajudam consultas com múltiplos filtros — a ordem das colunas no índice importa.
- Escolher o tipo de dado mais estreito possível (ex: `SMALLINT` em vez de `BIGINT`) reduz espaço e melhora performance.
- **Seletividade:** um índice em coluna de baixa cardinalidade (ex: `sexo`, só 2 valores) ajuda pouco — o otimizador pode preferir Seq Scan mesmo com índice disponível.
- **Armazenamento por linha (row-based):** eficiente para OLTP (ler/escrever registros inteiros).
- **Armazenamento colunar (column-based):** eficiente para OLAP (agregações sobre poucas colunas, muitas linhas) — ver Módulos 15 e 16.

9 Transações

Propriedades ACID

Propriedade	Garantia
Atomicidade	A transação executa por completo ou não executa nada (tudo ou nada)
Consistência	Leva o BD de um estado válido a outro estado válido
Isolamento	Transações concorrentes não interferem umas nas outras
Durabilidade	Após o commit, os dados persistem mesmo em caso de falha

Níveis de Isolamento

Nível	Dirty Read	Non-Repeatable Read	Phantom Read
Read Uncommitted	Possível	Possível	Possível
Read Committed	Evitado	Possível	Possível
Repeatable Read	Evitado	Evitado	Possível
Serializable	Evitado	Evitado	Evitado

Locks e Granularidade

- **Lock compartilhado (S):** permite leitura simultânea por várias transações.
- **Lock exclusivo (X):** bloqueia leitura/escrita de outras transações enquanto ativo.

Granularidade	Concorrência	Overhead de controle
Linha (row-level)	Alta — só bloqueia a linha afetada	Maior (muitos locks a gerenciar)
Página (page-level)	Média	Médio
Tabela (table-level)	Baixa — bloqueia tudo	Menor (um único lock)

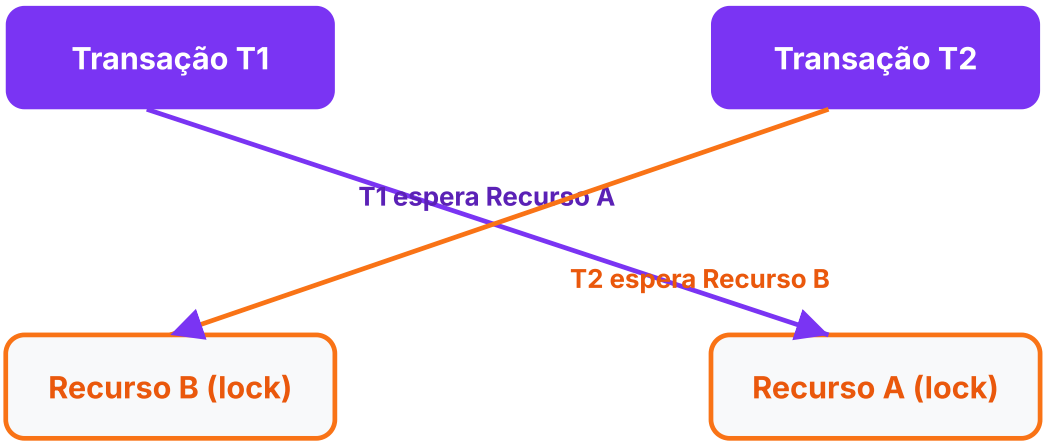
Duas Fases de Locking (2PL)

Protocolo que garante *serializabilidade*: toda transação tem uma **fase de crescimento** (só adquire locks, nunca libera) seguida de uma **fase de encolhimento** (só libera locks, nunca

adquire). Uma vez que a transação libera o primeiro lock, ela não pode mais pedir novos — isso evita que outra transação "veja" um estado intermediário inconsistente.

Deadlock

Ocorre quando duas ou mais transações esperam recursos bloqueadas mutuamente — um ciclo de espera sem saída.



T1 detém A e quer B; T2 detém B e quer A — ciclo de espera = deadlock. O SGBD detecta o ciclo (wait-for graph) e aborta uma das transações (vítima).

Prevenção: ordenar a aquisição de locks sempre na mesma sequência entre transações, ou usar *timeout* (se a espera passar de X segundos, aborta). **Detecção:** o SGBD constrói periodicamente um grafo de espera e escolhe uma vítima (geralmente a transação mais barata de desfazer) para rollback automático.

MVCC (Multiversion Concurrency Control)

Em vez de bloquear leitores, o SGBD mantém múltiplas versões de cada linha, permitindo que leituras vejam um snapshot consistente sem esperar escritores (usado por PostgreSQL, Oracle, MySQL/InnoDB).

Como funciona na prática: cada linha ganha metadados de versão (ex: `xmin` / `xmax` no PostgreSQL — a transação que criou e a que "apagou" a versão). Quando uma transação começa, ela enxerga um snapshot fixo: só linhas commitadas antes do seu início. Um `UPDATE` não sobrescreve a linha — cria uma *nova versão*; a antiga é limpa depois por um processo de *vacuum/garbage collection*. Resultado: leitores nunca bloqueiam escritores, e vice-versa.

Controle de Transação

```
BEGIN;
UPDATE conta SET saldo = saldo - 100 WHERE id = 1;
UPDATE conta SET saldo = saldo + 100 WHERE id = 2;
COMMIT; -- ou ROLLBACK; em caso de erro
```

10 Programação em Banco de Dados

Procedures x Functions

	Procedure	Function
Retorno	Opcional (ou múltiplo via `OUT`)	Obrigatório, valor único
Uso	Chamada isolada (`CALL`)	Pode ser usada dentro de um `SELECT`
Transação	Pode fazer `COMMIT` / `ROLLBACK` internamente	Não pode controlar transação

Exemplo de procedure (transferência entre contas, com controle transacional):

```
CREATE PROCEDURE transferir(origem INT, destino INT, valor DECIMAL)
LANGUAGE plpgsql AS $$
BEGIN
  UPDATE conta SET saldo = saldo - valor WHERE id = origem;
  UPDATE conta SET saldo = saldo + valor WHERE id = destino;

  IF (SELECT saldo FROM conta WHERE id = origem) < 0 THEN
    RAISE EXCEPTION 'Saldo insuficiente';
  END IF;

  COMMIT;
END;
$$;

CALL transferir(1, 2, 100.00);
```

Exemplo de function (usável dentro de um `SELECT`):

```
CREATE FUNCTION idade(data_nasc DATE) RETURNS INT
LANGUAGE plpgsql AS $$
BEGIN
  RETURN DATE_PART('year', AGE(data_nasc));
END;
$$;

SELECT nome, idade(data_nasc) FROM aluno;
```

Trigger — BEFORE x AFTER

```
-- AFTER: a ação já foi aplicada; usado para efeitos colaterais (auditoria, sincronizar
CREATE TRIGGER trg_atualiza_estoque
AFTER INSERT ON venda
FOR EACH ROW
BEGIN
  UPDATE produto SET estoque = estoque - NEW.quantidade
  WHERE id = NEW.produto_id;
END;

-- BEFORE: dá para interceptar/alterar os dados antes de gravar, ou bloquear a operação
CREATE TRIGGER trg_valida_preco
BEFORE INSERT ON produto
FOR EACH ROW
BEGIN
  IF NEW.preco ⩽ 0 THEN
    RAISE EXCEPTION 'Preço deve ser positivo';
  END IF;
  NEW.criado_em := NOW();
END;
```

	BEFORE	AFTER
Quando roda	Antes da alteração ser gravada	Depois da alteração já gravada
Pode alterar `NEW` ?	Sim	Não (já foi gravado)
Uso típico	Validação, valores default, auditoria de tentativa	Sincronizar outras tabelas, notificações, log de fato ocorrido

Disparado automaticamente por eventos (`BEFORE` / `AFTER` + `INSERT` / `UPDATE` / `DELETE`), sem chamada explícita — cuidado com triggers encadeados (um trigger disparando outro), que dificultam depuração.

Cursores e Controle de Fluxo

```
DECLARE cur CURSOR FOR SELECT id FROM cliente;
OPEN cur;
FETCH NEXT FROM cur INTO @id;
WHILE @@FETCH_STATUS = 0
BEGIN
  -- processa linha por linha
  FETCH NEXT FROM cur INTO @id;
END;
CLOSE cur;
```

Estruturas de controle disponíveis em PL/SQL, T-SQL, PL/pgSQL: `IF/ELSE`, `WHILE`, `LOOP`, `CASE`.

Tratamento de Exceções

```
BEGIN
  -- bloco de comandos
EXCEPTION
  WHEN division_by_zero THEN
    RAISE NOTICE 'Erro: divisão por zero';
  WHEN OTHERS THEN
    RAISE NOTICE 'Erro inesperado';
END;
```

13 NoSQL

Tipos de Bancos NoSQL

Tipo	Exemplo	Estrutura	Caso de uso típico
Documento	MongoDB	JSON/BSON, schema flexível	Catálogos de produto, conteúdo semi-estruturado
Chave-valor	Redis	Chave → valor, em memória	Cache, sessões, filas
Colunar	Cassandra	Famílias de colunas distribuídas	Big data, alta taxa de escrita
Grafos	Neo4j	Nós e arestas	Redes sociais, recomendação, detecção de fraude

Exemplos concretos por tipo

Documento (MongoDB) — cada registro é um documento JSON auto-contido, sem schema fixo:

```
// db.produtos.insertOne(...)
{
  "_id": "p001",
  "nome": "Teclado mecânico",
  "preco": 350.00,
  "tags": ["periferico", "gamer"],
  "especificacoes": { "switch": "blue", "layout": "ABNT2" }
}
db.produtos.find({ "tags": "gamer", preco: { $lt: 400 } });
```

Chave-valor (Redis) — acesso direto por chave, em memória:

```
SET sessao:usuario:42 "{ \"logado\": true, \"carrinho\": 3 }" EX 3600
GET sessao:usuario:42
INCR contador:visitas:home
```

Colunar (Cassandra/CQL) — otimizado para altíssimo volume de escrita distribuída:

```
CREATE TABLE eventos_sensor (
  sensor_id UUID,
  ts TIMESTAMP,
  temperatura DOUBLE,
  PRIMARY KEY (sensor_id, ts)
) WITH CLUSTERING ORDER BY (ts DESC);

SELECT * FROM eventos_sensor WHERE sensor_id = 123 LIMIT 10;
```

Grafos (Neo4j/Cypher) — modela relacionamentos como cidadãos de primeira classe:

```
CREATE (a:Pessoa {nome:'Ana'})-[:SEGUE]→(b:Pessoa {nome:'Beto'})

MATCH (p:Pessoa {nome:'Ana'})-[:SEGUE]→(amigo)-[:SEGUE]→(sugestao)
WHERE NOT (p)-[:SEGUE]→(sugestao)
RETURN DISTINCT sugestao.nome;  -- "amigos de amigos" — pesadelo em SQL, natural em graf
```

Teorema CAP

Em um sistema distribuído, só é possível garantir **duas** das três propriedades simultaneamente:

- **C**onsistency — todos os nós veem os mesmos dados ao mesmo tempo.
- **A**vailability — todo request recebe resposta (sem garantir que seja o dado mais recente).
- **P**artition Tolerance — o sistema continua funcionando mesmo com falha de comunicação entre nós.

Como falhas de rede são inevitáveis em sistemas distribuídos, a escolha real costuma ser entre **CP** (consistência) e **AP** (disponibilidade) — *P* não é opcional em um sistema realmente distribuído.

Escolha	Comportamento sob partição	Exemplos
CP	Rejeita requisições que não podem garantir dado atualizado	MongoDB (config padrão), HBase, bancos relacionais distribuídos
AP	Sempre responde, mesmo com risco de dado desatualizado (consistência eventual)	Cassandra, DynamoDB, Redis (em cluster)

BASE (Basically Available, Soft state, Eventual consistency) é o modelo alternativo ao ACID adotado por bancos AP: prioriza disponibilidade e escala, aceitando que os nós fiquem temporariamente inconsistentes até convergirem.

15 Data Warehouse

OLTP x OLAP

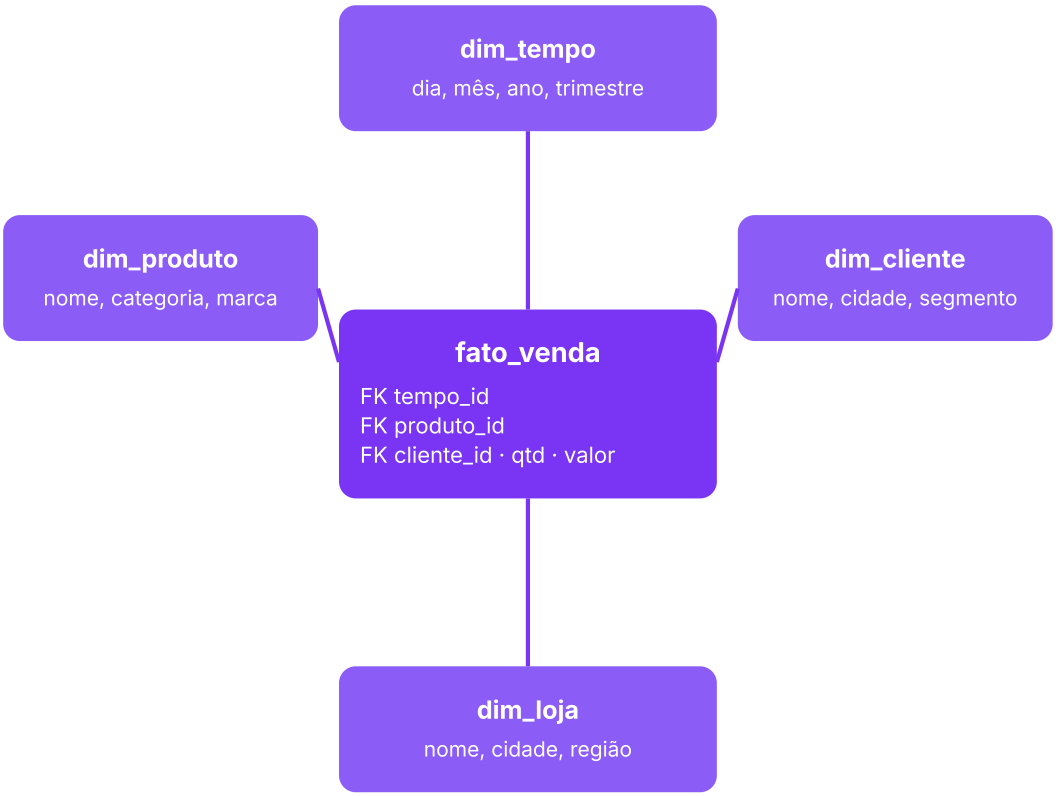
	OLTP	OLAP
Objetivo	Operações do dia a dia (transações)	Análise histórica e tomada de decisão
Modelagem	Normalizada (3FN)	Desnormalizada (Star/Snowflake)
Consultas	Curtas, muitas escritas	Complexas, agregações, poucas escritas

ETL x ELT

ETL (Extract, Transform, Load): transforma os dados antes de carregar no destino. **ELT** (Extract, Load, Transform): carrega os dados brutos primeiro e transforma dentro do próprio destino (comum com Data Warehouses/Lakes modernos e alto poder de processamento).

Star Schema x Snowflake Schema

- **Star Schema:** uma tabela fato central conectada diretamente a tabelas de dimensão desnormalizadas — simples e rápido para consultas.
- **Snowflake Schema:** as dimensões são normalizadas em sub-tabelas — economiza espaço, mas exige mais `JOIN`s.



Star Schema: fato central com métricas, dimensões desnormalizadas ao redor (formato de estrela).

Consulta típica sobre um Star Schema (agregação simples, poucos `JOIN` s):

```
SELECT dt.ano, dp.categoria, SUM(f.valor) AS total_vendido
FROM fato_venda f
JOIN dim_tempo   dt ON f.tempo_id = dt.id
JOIN dim_produto dp ON f.produto_id = dp.id
GROUP BY dt.ano, dp.categoria
ORDER BY total_vendido DESC;
```

Fatos, Dimensões e Data Mart

- **Tabela fato:** armazena métricas/eventos mensuráveis (ex: valor da venda) e chaves estrangeiras para as dimensões. Cresce muito (uma linha por evento/transação).
- **Tabela dimensão:** armazena o contexto descritivo (ex: cliente, produto, tempo, loja). Cresce pouco, muito lida.
- **Data Mart:** subconjunto do Data Warehouse voltado a um departamento específico (ex: Data Mart de Vendas).

Slowly Changing Dimensions (SCD)

Como tratar quando um atributo de dimensão muda (ex: cliente muda de cidade)?

Tipo	Estratégia	Efeito
SCD 1	Sobrescreve o valor antigo	Perde histórico — só reflete o estado atual
SCD 2	Cria uma nova linha com novo `id` substituto e datas de vigência (`valido_de` / `valido_ate`)	Preserva histórico completo — permite análises "como era na época"
SCD 3	Adiciona uma coluna extra (ex: `cidade_anterior`)	Guarda só a mudança mais recente, sem histórico completo

16 Data Lake

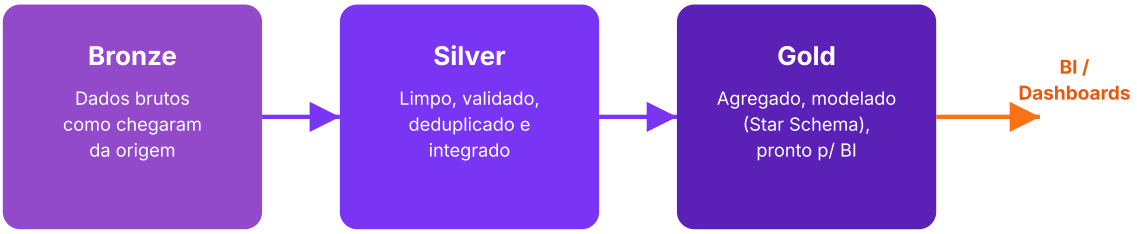
Formatos de Arquivo

- **Parquet:** formato colunar, comprimido, ideal para análise (lê só as colunas necessárias).
- **Delta Lake / Apache Iceberg:** camadas transacionais sobre arquivos do Data Lake — trazem ACID, versionamento (time travel) e schema evolution para dados que antes eram só arquivos "soltos".

Ferramentas do Ecossistema

Ferramenta	Papel
Apache Spark	Processamento distribuído de dados em larga escala
Hadoop (HDFS)	Armazenamento distribuído tolerante a falhas
Apache Airflow	Orquestração e agendamento de pipelines de dados
dbt	Transformação de dados usando SQL, com versionamento e testes
Trino	Motor de consulta SQL federado sobre múltiplas fontes

Arquitetura Medalhão (Bronze, Silver, Gold)



Cada camada refina a anterior: Bronze (raw) → Silver (limpo) → Gold (agregado, pronto para consumo).

- **Bronze:** dados brutos, exatamente como chegaram da origem — mantidos para auditoria e reprocessamento.
- **Silver:** dados limpos, validados, deduplicados e integrados entre fontes diferentes.
- **Gold:** dados agregados e modelados, prontos para consumo por BI/análises (frequentemente já em Star Schema).

Data Lake x Data Warehouse x Lakehouse

	Data Warehouse	Data Lake	Lakehouse
Dados aceitos	Estruturados	Estruturados, semi e não estruturados	Todos, com camada transacional
Schema	Schema-on-write	Schema-on-read	Schema-on-read + validação (Delta/Iceberg)
Custo de armazenamento	Mais alto	Baixo (object storage)	Baixo
Suporta ACID?	Sim	Não nativamente	Sim (via Delta Lake/Iceberg)
Exemplo	Redshift, BigQuery, Snowflake	S3 + Hadoop cru	Databricks (Delta Lake), Iceberg + Trino

🎯 Conceitos que mais caem em provas

- Modelagem ER (entidades, cardinalidade, entidades fracas)
- Normalização (1FN, 2FN, 3FN, BCNF)
- SQL: `JOIN`, `GROUP BY` e subconsultas
- Transações e propriedades ACID
- Dependências funcionais
- Índices (B-Tree x Hash, quando usar)
- Procedures / Triggers
- NoSQL x Relacional (quando usar cada um)
- ETL / ELT
- Data Warehouse x Data Lake

Apostila gerada para a Aula SBD · Banco de Dados I e II