Aula 7

1. Introdução à Modelagem

A modelagem de dados é uma etapa fundamental em qualquer projeto de banco de dados. Seu principal objetivo é representar, de forma organizada e lógica, as informações de um domínio de negócio, permitindo que os dados sejam armazenados, consultados e analisados de maneira eficiente.

- **Definição e objetivo de modelagem de dados:** A modelagem de dados é o processo de **estruturar e organizar dados** para refletir as entidades, atributos e relacionamentos de um contexto real. Ela cria uma *ponte* entre o mundo conceitual (como os usuários enxergam os dados) e o mundo físico (como os dados são armazenados no banco). Um bom modelo de dados facilita tanto o uso transacional quanto a análise estratégica das informações.
- **Diferença entre Modelagem Transacional (OLTP) e Analítica (OLAP):**
  - **OLTP (Online Transaction Processing):** Voltado para o **registro contínuo de transações operacionais**. São bancos que priorizam desempenho em inserções, atualizações e exclusões frequentes. Exemplos típicos incluem sistemas de vendas, cadastros de clientes e controle de estoque. **Características principais:**
    - Estrutura altamente normalizada (para evitar redundâncias);
    - Tabelas pequenas e com muitos relacionamentos;
    - Consultas rápidas e simples, voltadas à operação diária.
  - **OLAP (Online Analytical Processing):** Focado em **análises e consultas complexas** sobre grandes volumes de dados históricos. O objetivo é responder perguntas de negócio e apoiar a tomada de decisão. Exemplos incluem relatórios gerenciais, dashboards e análises de séries temporais. **Características principais:**
    - Estruturas desnormalizadas (como esquemas estrela e floco de neve);
    - Uso intenso de funções de agregação (`SUM`, `AVG`, `COUNT`);
    - Consultas complexas e multidimensionais.
- **Papel do modelo de dados:** O modelo de dados serve como **mapa estrutural** do sistema, definindo como as informações se relacionam e se transformam em conhecimento. É ele que permite que analistas e cientistas de dados consigam extrair **insights** a partir de consultas analíticas e cruzamentos entre tabelas. Além disso, um modelo bem planejado garante:

- Integridade e consistência dos dados;
- Facilidade de manutenção e expansão do banco;
- Desempenho otimizado para diferentes tipos de consulta.

Exemplo Prático: Diferença entre OLTP e OLAP

Exemplo OLTP (Sistema de Vendas):

```
CREATE TABLE vendas (
    id_venda SERIAL PRIMARY KEY,
    id_cliente INT,
    data_venda DATE,
    valor NUMERIC(10,2)
);
```

Exemplo OLAP (Fato de Vendas e Dimensões):

```
-- Tabela fato (dados quantitativos)
CREATE TABLE fato_vendas (
    id_cliente INT,
    id_produto INT,
    id_tempo INT,
    valor_venda NUMERIC(10,2)
);

-- Tabelas dimensão (dados qualitativos)
CREATE TABLE dim_cliente (id_cliente INT, nome TEXT, cidade TEXT);
CREATE TABLE dim_produto (id_produto INT, categoria TEXT);
CREATE TABLE dim_tempo (id_tempo INT, ano INT, mes INT, dia INT);
```

Esses dois exemplos mostram como o propósito muda: enquanto o primeiro modelo é voltado à operação diária de vendas, o segundo é voltado à análise de desempenho (ex: total de vendas por categoria, mês ou cliente).

—

Boas Práticas de Modelagem

- Comece com um **modelo conceitual** (entidades e relacionamentos principais);
- Normalize o modelo transacional até a **3ª forma normal** (evitando redundâncias);
- Para modelos analíticos, prefira estruturas **desnormalizadas** (para otimizar consultas);
- Documente todos os relacionamentos e chaves (primárias e estrangeiras);

- Pense no modelo como uma **base para o crescimento futuro** — mantenha flexibilidade.

—

Resumo Intuitivo

A modelagem de dados é o alicerce de qualquer projeto de banco de dados. Um modelo bem estruturado define como os dados serão armazenados, acessados e analisados, garantindo qualidade, consistência e desempenho em todo o ciclo de vida da informação. Compreender as diferenças entre ambientes OLTP e OLAP é essencial para projetar soluções que atendam tanto às necessidades operacionais quanto analíticas de uma organização.

2. Arquitetura Principal: Star Schema

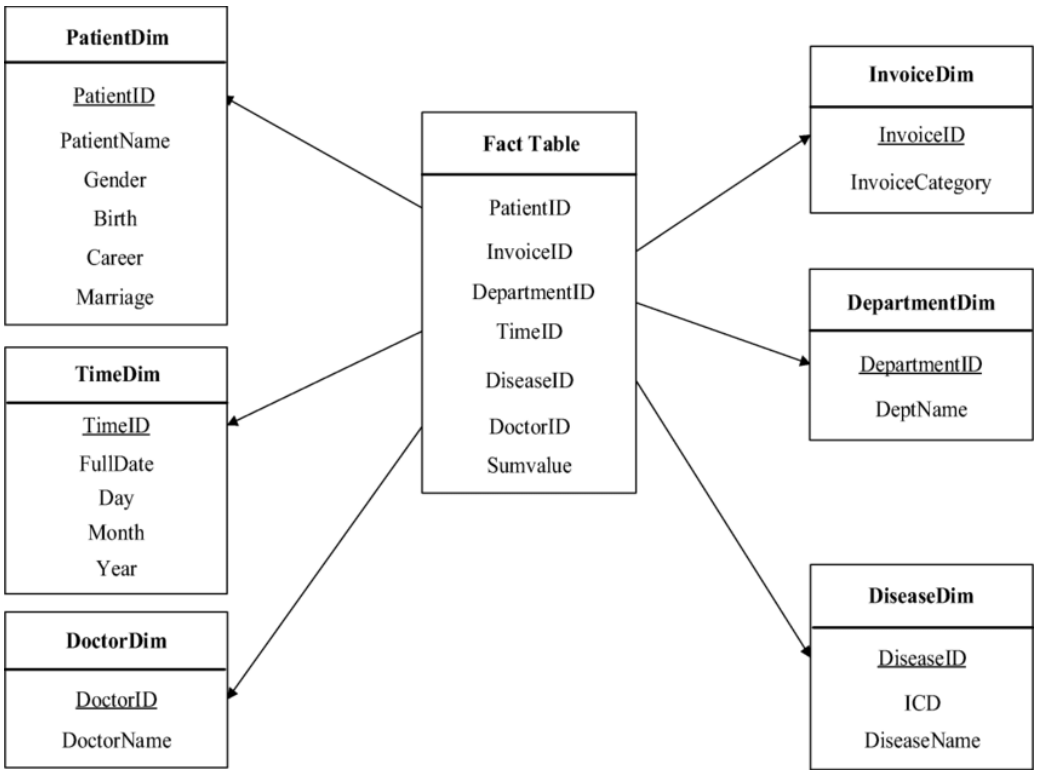
A arquitetura em estrela (esquema de estrela) é um dos modelos de organização de dados em databases mais comuns e utilizados. Seu foco principal está em otimizar a performance de consultas e a análise de grandes quantidades de dados, configurando-se como um sistema OLAP. Sua estrutura geral é dividida em dois tipos de tabelas:

- Tabela Fato: é uma tabela central do modelo. Ela armazena dados quantitativos e numéricos do negócio, os quais serão alvos de funções de soma, média e contagem, por exemplo. Além disso, a tabela fato armazena chaves estrangeiras relacionadas às demais tabelas do modelo.
- Tabela Dimensão: é um conjunto de tabelas que se organizam em camadas ao redor da tabela fato. Elas focam em descrever os dados armazenados na tabela central, possuindo chaves primárias e atributos variados.

Em geral, a tabela fato possui um número grande de linhas e um número baixo de colunas. Por outro lado, as tabelas de dimensão possuem menos linhas e mais colunas. Essas útlimas, ademais, possuem uma propriedade importante: não são normalizadas. Isso significa que permitimos a existência de redundâncias em uma mesma relação, resultando em menos tabelas e maior velocidade de queries. Uma tabela Produtos, por exemplo, poderia conter atributos de categoria e subcategoria, os quais se repetiriam diversas vezes. Neste caso, a versão normalizada seria criar relações próprias para a categoria e para a subcategoria.

Para exemplificar, suponha que temos um sistema relacionado a atendimentos em um hospital. Neste cenário, teríamos tabelas de dimensão variadas, como médico, paciente, doença e data. Na tabela central, poderíamos ter, além das chaves estrangeiras, o custo que um paciente teve que pagar para uma consulta com um doutor por conta de uma doença em uma data. Pensando nisso, segue uma arquitetura em estrela possível.

Outro conceito relacionado ao esquema de estrela é o de granularidade. Granularidade refere-se ao nível de detalhamento dos registros da tabela fato. Quanto maior a granularidade, maior o nível de detalhamento. Em um negócio que possui vendas em diferentes lojas, a granularidade pode ser da seguinte forma:



- Alta granularidade: quando ocorre a compra de vários produtos por um cliente, é gerada uma linha por compra.
- Baixa granularidade: quando ocorre a compra de vários produtos por um cliente, é gerada apenas uma linha com o resumo total de compras.

Observe que a alta granularidade possibilita análises mais completas, como análises de vendas individuais de produtos. Todavida, isso tende a aumentar não só o tamanho do banco de dados (mais memória ocupada), mas também o tempo de execução de consultas, já que mais agregações de registros são necessárias. Com isso, devemos analisar sempre o trade-off entre a explicabilidade e o detalhamento da alta granularidade e a otimização e menor detalhamento da baixa granularidade.

Por fim, podemos observar algumas vantagens e problemas relacionados à arquitetura star. A princípio, pela não normalização, temos uma menor necessidade de JOINs, o que permite maior otimização e simplificação das consultas. Ademais, a análise torna-se mais fácil, já que o entendimento da estrutura e da hierarquia de relações é simples. Por outro lado, a falta de normalização pode gerar problemas de integridade pela atualização de apenas uma parcela dos dados duplicados. Não obstante, podemos ter mais uso de memória e mais trabalho em processos de update.

3. Tipos de Tabelas Fato

As tabelas fato são componentes centrais em um modelo dimensional de banco de dados, armazenando medidas quantitativas que refletem eventos de negócio. Elas podem ser classificadas de acordo com o tipo de evento ou processo que representam:

- **Fato Transacional:** Registra eventos individuais e detalhados, como uma venda, uma transação

bancária ou uma solicitação de serviço. Cada linha da tabela representa um evento específico que ocorreu em determinado momento.

- **Fato Snapshot Periódico:** Captura o estado de um processo ou entidade em intervalos regulares de tempo, como o estoque diário, o saldo mensal de contas ou o número de funcionários ativos por semana. É útil para análises históricas e acompanhamento de desempenho.
- **Fato Snapshot Acumulativo:** Representa o ciclo de vida de um processo com início e fim definidos, como o acompanhamento do processamento de um pedido, de um projeto ou de um funil de vendas. Permite observar a progressão de um processo ao longo do tempo.

Tabela 1: Exemplo de Tabela Fato Transacional — Vendas

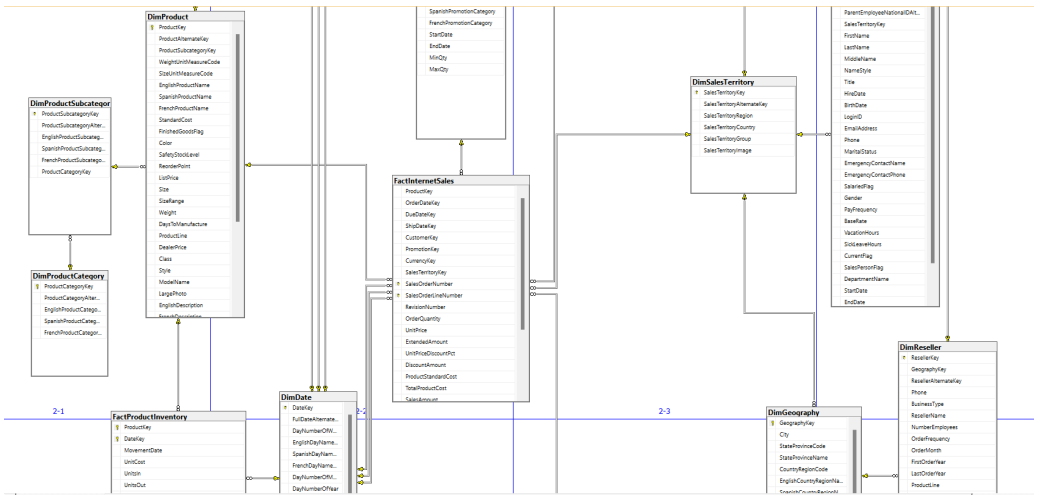
ID Venda	Data	Produto	Quantidade	Valor Total (R$)
001	10/03/2025	Notebook	2	8.000,00
002	10/03/2025	Teclado	5	750,00
003	11/03/2025	Monitor	3	2.700,00
004	12/03/2025	Mouse	4	400,00

Neste exemplo, cada linha representa uma **transação de venda**, com seus respectivos atributos descritivos (como data e produto) e medidas quantitativas (como quantidade e valor total). Esse tipo de estrutura permite análises rápidas, como o total de vendas por período ou por categoria de produto.

4. Variações de Arquitetura: Galaxy Schema

O modelo em estrela, como vimos, é simples. Apesar disso trazer benefícios, também contritubi para limitações de representação. Suponha que uma empresa de varejo queira observar os seguintes dados: vendas online a clientes finais, vendas a revendedores e metas a serem batidas por um vendedor. Note que tais tabelas utilizam chaves estrangeiras de outras relações (cliente, data, produto, vendedor, revendedor, etc.) e possuem seus próprios valores numéricos para cálculos estatísticos (valores de venda, revenda e de metas). Logo, todas se comportam conforme uma tabela fato. Contudo, o modelo estrela não permite mais do que uma tabela fato. Para tanto, podemos utilizar um modelo modificado da arquitetura estrela: a arquitetura de galaxia.

A arquitetura de galáxia surge como uma tentativa de modelar sistemas mais complexos e realistas de forma a manter a otimização do modelo star e ganhar mais consistência e representação dos dados. Para isso, utilizamos duas ou mais tabelas fato, cada qual com sua granularidade e tabelas de dimensão específicas. Note que uma tabela de dimensão pode ser compartilhada entre tabelas fato. Para exemplificar, podemos usar o database AdventureWorks da Microsoft, que simula uma companhia de vendas de bicicletas e de acessórios de ciclismo. A estrutura está representada abaixo (zoom recomendado). Observe que neste recorte, temos duas tabelas de fatos: uma para o inventário da companhia e outra para as vendas feitas pela internet. Note que, para ambas, temos chaves estrangeiras e dados numéricos, o que reforça a ideia de serem tabelas fato.



5. Variações de Arquitetura: Snowflake Schema

O *Snowflake Schema* (ou esquema em floco de neve) é uma variação do modelo em estrela que busca reduzir redundâncias por meio da normalização das dimensões. Em vez de manter uma única tabela de dimensão contendo todas as informações relacionadas, o modelo snowflake divide essas dimensões em múltiplas tabelas inter-relacionadas, seguindo as regras de normalização. Essa estrutura cria um formato visual semelhante a um floco de neve, com ramificações a partir das tabelas de dimensão.

- **Conceito:** O Snowflake Schema normaliza as dimensões de um Star Schema, fragmentando-as em subdimensões relacionadas. Assim, evita-se a duplicação de dados, embora o número de junções aumente.

A Figura 1 abaixo mostra uma comparação visual entre o *Star Schema* e o *Snowflake Schema*. Observe que o modelo estrela (à esquerda) apresenta uma única camada de dimensões ligadas diretamente à tabela fato, enquanto o modelo floco de neve (à direita) subdivide as dimensões em hierarquias adicionais, normalizando a estrutura.

Análise Comparativa

- **Star Schema:**
  - Estrutura simples e de fácil compreensão.
  - Consultas mais rápidas, pois requerem menos junções.
  - Redundância de dados nas dimensões.
- **Snowflake Schema:**
  - Estrutura mais normalizada, com menor redundância.
  - Consultas mais complexas e potencialmente mais lentas devido ao número maior de junções.
  - Facilita manutenção e atualização de dados dimensionais.

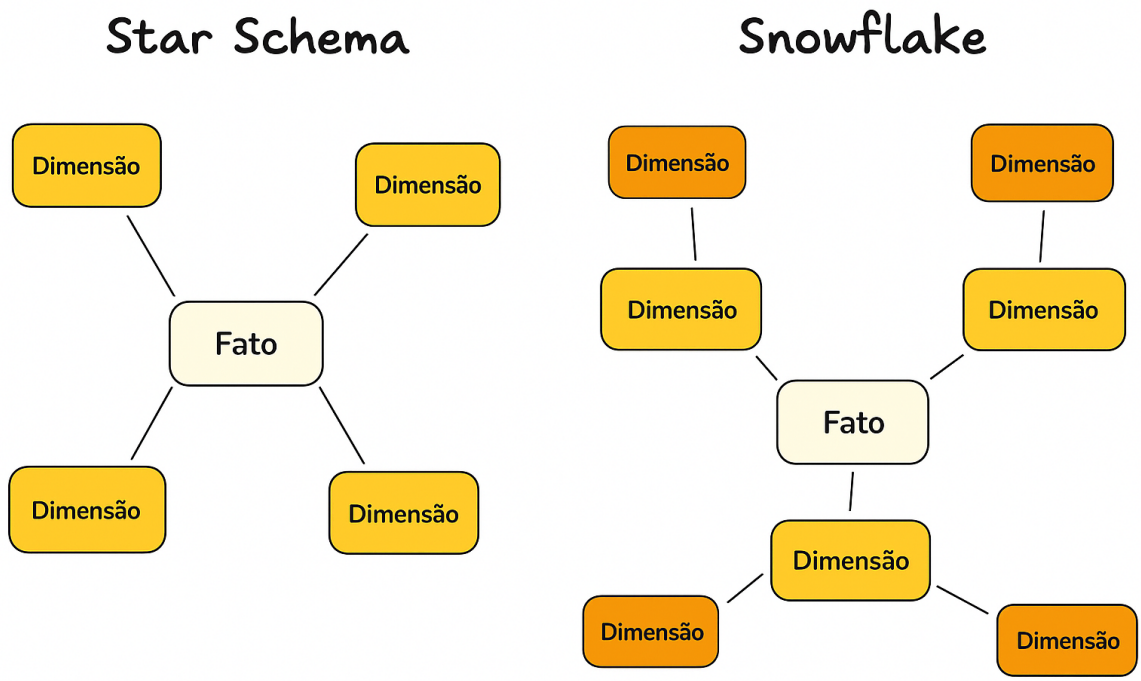


Figura 1: Comparação entre o Star Schema e o Snowflake Schema.

Exemplo Prático

Suponha que uma consulta deseje obter o total de vendas por categoria de produto e país. No modelo estrela, bastaria relacionar a tabela fato com `DimProduct` e `DimGeography`. Já no modelo snowflake, seria necessário incluir junções adicionais com `DimProductSubcategory` e `DimProductCategory`, como ilustrado abaixo:

```
SELECT c.EnglishCountryRegionName, pc.EnglishProductCategoryName,
       SUM(fs.SalesAmount) AS TotalSales
FROM FactInternetSales fs
JOIN DimProduct p ON fs.ProductKey = p.ProductKey
JOIN DimProductSubcategory ps ON p.ProductSubcategoryKey = ps.ProductSubcategoryKey
JOIN DimProductCategory pc ON ps.ProductCategoryKey = pc.ProductCategoryKey
JOIN DimCustomer cu ON fs.CustomerKey = cu.CustomerKey
JOIN DimGeography c ON cu.GeographyKey = c.GeographyKey
GROUP BY c.EnglishCountryRegionName, pc.EnglishProductCategoryName;
```

Essa diferença ilustra bem o trade-off entre simplicidade (Star Schema) e normalização (Snowflake Schema), ambos amplamente utilizados em projetos de *Data Warehousing*.

6. Modelagem de Dados Históricos: Slowly Changing Dimensions (SCD)

Uma das preocupações na modelagem de dados é como as dimensões mudam. Uma delas é a SCD (Slowly Changing Dimensions), que lidam com dados que mudam lentamente (por exemplo, dados

sobre nomes de ruas). Elas constrastam com as RCP (Rapidly Changing Dimensions), que lidam com dados que mudam constantemente, como preços de um produto.

Existem diversos tipos de técnicas para lidarem com as SCD. Por exemplo, numa tabela de funcionários, vamos supor que nós queremos lidar com os cargos deles. Este é um dado que costuma mudar lentamente, e nós queremos inserir a nova informação que um dos funcionários foi promovido a gerente.

O primeiro tipo de gerenciamento dessa SCD seria simplesmente sobreescrever o dado antigo. Em SQL, isso seria feito com o comando UPDATE. O problema desse tipo é que ele não preserva o histórico dos dados. Ninguém poderia saber qual foi minha carreira dentro da empresa, somente meu cargo atual.

Por isso, existe o tipo 2, que cria uma linha na nossa tabela com meu estado atual de cargo. Para diferenciar entre as linhas com meu nome, poderia ser criado uma nova coluna com quantas vezes eu troquei de cargo até o momento da inserção dessa nova linha, ou mesmo criar duas colunas com o intervalo de tempo em que eu servi cada cargo.

Além desses, existem diversos outros tipo. Dois bem comuns são:

- Tipo 3 - Criar novas colunas. Você poderia criar duas colunas: cargo antigo e cargo novo.
- Tipo 4 - Criar uma tabela de histórico. Agora, minha linha possui um ID que aponta para uma tabela com o histórico de cargos, e o campo do meu cargo propriamente dito é sobreescrito (como no tipo 1)

7. Exemplo: Transformação de um Modelo Genérico

Para entender na prática as diferenças entre os modelos analíticos, vamos tomar como ponto de partida um modelo de dados genérico, tipicamente encontrado em sistemas transacionais (OLTP), e transformá-lo em um **Star Schema** e em um **Snowflake Schema**.

7.1 O Modelo Genérico (Transacional - OLTP)

Sistemas de produção (como um e-commerce ou um sistema de gestão) são otimizados para *transações*: inserir, atualizar e deletar registros de forma rápida e consistente. Para isso, eles usam a **normalização** (como a 3ª Forma Normal), quebrando os dados em muitas tabelas pequenas para evitar redundância de dados.

Vamos imaginar um modelo OLTP simplificado para um sistema de vendas:

- `Clientes` (id_cliente, nome, cidade, estado, pais)
- `Categorias` (id_categoria, nome_categoria)
- `Subcategorias` (id_subcategoria, nome_subcategoria, id_categoria)
- `Produtos` (id_produto, nome_produto, id_subcategoria)
- `Vendas` (id_venda, data_venda, id_cliente)

- `ItensVenda` (id_item, id_venda, id_produto, quantidade, preco_unitario)

Problemas deste modelo para Análise (OLAP):

Embora perfeito para o dia a dia, esse modelo é muito ineficiente para relatórios gerenciais. Uma simples pergunta de negócio como:

"Qual a receita total por categoria de produto e por estado do cliente no último trimestre?"

... exigiria uma consulta SQL extremamente complexa e lenta:

1. `Vendas` JOIN `Clientes` (para pegar o estado)
2. `Vendas` JOIN `ItensVenda` (para pegar os produtos e valores)
3. `ItensVenda` JOIN `Produtos` (para pegar a subcategoria)
4. `Produtos` JOIN `Subcategorias` (para pegar a categoria)
5. `Subcategorias` JOIN `Categorias` (para pegar o nome da categoria)
6. Filtrar por `data_venda`
7. Agrupar (`GROUP BY`) por `estado` e `nome_categoria`

Essa consulta envolve **5 JOINs** apenas para responder uma pergunta simples. Em um banco de dados com milhões de registros na tabela `ItensVenda`, essa consulta seria inviável para um dashboard.

7.2 Transformação para Star Schema (Foco em Desempenho)

O objetivo do Star Schema é a **simplicidade e velocidade**. Criamos uma **Tabela Fato** central para as métricas (o que queremos medir?) e **Tabelas Dimensão** ao redor (como queremos medir?).

Para isso, nós **desnormalizamos** os dados de contexto (as dimensões).

- **Tabela Fato (Métricas):** O evento central é o item vendido.
  - `fato_vendas` (id_data, id_cliente, id_produto, quantidade_vendida, receita_total)
- **Tabelas Dimensão (Contexto):**
  - `dim_data` (id_data, data_completa, ano, mes, dia, trimestre, dia_da_semana)
  - `dim_cliente` (id_cliente, nome, cidade, estado, pais)
  - `dim_produto` (id_produto, nome_produto, **nome_subcategoria, nome_categoria**)

Mudança principal: Observe a `dim_produto`. Nós "achatamos"a hierarquia. As tabelas `Categorias` e `Subcategorias` foram fundidas diretamente na dimensão de produto. Há redundância (o nome "Eletrônicos"vai se repetir para cada produto dessa categoria), mas isso é **intencional** para eliminar JOINs.

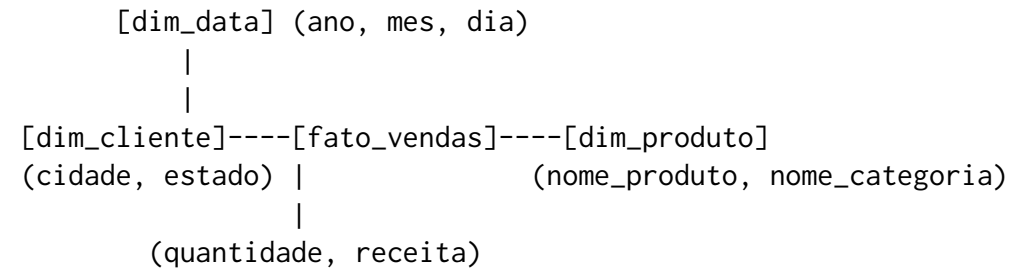


Figura 2: Estrutura de um Star Schema.

Vantagem: A mesma consulta ("Receita por categoria e estado") agora é trivial e muito rápida:

1. `fato_vendas` JOIN `dim_cliente`
2. `fato_vendas` JOIN `dim_produto`
3. Agrupar por `estado` e `nome_categoria`

Reduzimos de 5 JOINs para apenas 2.

7.3 Transformação para Snowflake Schema (Foco em Normalização)

O Snowflake Schema é um "meio-termo". Ele parte do Star Schema, mas aplica a **normalização** nas dimensões para reduzir a redundância de dados e facilitar a manutenção.

- **Tabela Fato:** Permanece idêntica à do Star Schema.
  - `fato_vendas` (id_data, id_cliente, id_produto, quantidade_vendida, receita_total)
- **Tabelas Dimensão (Normalizadas):**
  - `dim_data` (Idêntica)
  - `dim_cliente` (Idêntica)
  - `dim_produto` (id_produto, nome_produto, **id_subcategoria**)
  - `dim_subcategoria` (id_subcategoria, nome_subcategoria, **id_categoria**)
  - `dim_categoria` (id_categoria, nome_categoria)

Mudança principal: A dimensão `Produto` foi "quebrada"(normalizada). As informações de subcategoria e categoria foram movidas para suas próprias tabelas, criando as ramificações que dão nome ao modelo ("floco de neve").

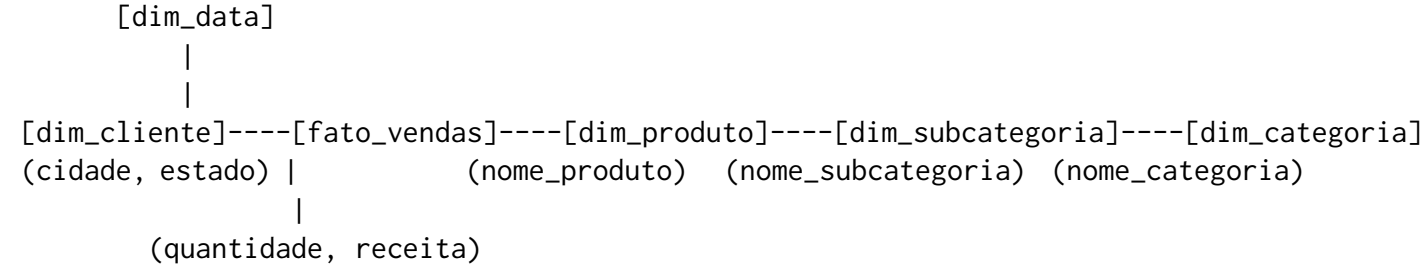


Figura 3: Estrutura de um Snowflake Schema.

Vantagem: Há menos redundância. Se o nome de uma categoria mudar (ex: "Computadores"para "Informática"), só precisamos atualizar um registro na `dim_categoria`. No Star Schema, teríamos que atualizar milhares de linhas na `dim_produto`.
Desvantagem: A nossa consulta ("Receita por categoria e estado") agora exige mais JOINs novamente, tornando-a mais lenta que no Star Schema (mas ainda muito mais rápida que no modelo OLTP original).

7.4 Exemplo Prático de Transformação (SQL)

A seguir, apresentamos os comandos SQL (em PostgreSQL) necessários para executar a transformação do modelo OLTP genérico para o nosso **Star Schema** alvo. Este processo é um exemplo de **ETL (Extract, Transform, Load)**.
Modelo OLTP (Origem):

- `Clientes`, `Categorias`, `Subcategorias`, `Produtos`, `Vendas`, `ItensVenda`

Modelo Star Schema (Destino):

- `dim_data`, `dim_cliente`, `dim_produto`, `fato_vendas`

Passo 1: Criando as Tabelas do Star Schema (CREATE)

Primeiro, criamos as estruturas vazias do nosso modelo analítico. Note as chaves primárias (PK) nas dimensões e as chaves estrangeiras (FK) na tabela fato.

```
-- Dimensão de Data
CREATE TABLE dim_data (
    id_data INT PRIMARY KEY, -- Formato YYYYMMDD
    data_completa DATE NOT NULL,
    ano INT NOT NULL,
    mes INT NOT NULL,
    dia INT NOT NULL,
    trimestre INT NOT NULL,
    dia_da_semana INT NOT NULL
);

-- Dimensão de Cliente
CREATE TABLE dim_cliente (
    id_cliente INT PRIMARY KEY,
    nome VARCHAR(255),
    cidade VARCHAR(100),
    estado VARCHAR(50),
    pais VARCHAR(50)
);
```

```
-- Dimensão de Produto (Desnormalizada)
CREATE TABLE dim_produto (
    id_produto INT PRIMARY KEY,
    nome_produto VARCHAR(255),
    nome_subcategoria VARCHAR(100),
    nome_categoria VARCHAR(100)
);

-- Tabela Fato
CREATE TABLE fato_vendas (
    -- Chaves Estrangeiras (FKs)
    id_data INT REFERENCES dim_data(id_data),
    id_cliente INT REFERENCES dim_cliente(id_cliente),
    id_produto INT REFERENCES dim_produto(id_produto),

    -- Metricas
    quantidade_vendida INT,
    receita_total NUMERIC(10, 2),

    -- Chave primária composta da fato
    PRIMARY KEY (id_data, id_cliente, id_produto)
);
```

Passo 2: Carregando as Tabelas Dimensão (Extract, Transform, Load)

Agora, populamos as dimensões extraindo e transformando os dados do modelo OLTP.

Carregando dim_cliente (Transformação simples 1:1)

```
INSERT INTO dim_cliente (
    id_cliente, nome, cidade, estado, pais
)
SELECT
    id_cliente, nome, cidade, estado, pais
FROM Clientes;
```

Carregando dim_produto (Transformação com "achatamento"/JOIN)

Aqui, nós "achatamos"a hierarquia (Produto → Subcategoria → Categoria) em uma única tabela, usando `JOIN`s.

```
INSERT INTO dim_produto (
    id_produto, nome_produto, nome_subcategoria, nome_categoria
```

```
)
SELECT
    p.id_produto,
    p.nome_produto,
    s.nome_subcategoria,
    c.nome_categoria
FROM Produtos p
LEFT JOIN Subcategorias s ON p.id_subcategoria = s.id_subcategoria
LEFT JOIN Categorias c ON s.id_categoria = c.id_categoria;
```

Carregando dim_data (Transformação a partir das datas existentes)

Criamos nossa dimensão de datas dinamicamente, buscando todas as datas únicas que existem na tabela `Vendas` e extraindo seus componentes.

```
INSERT INTO dim_data (
    id_data, data_completa, ano, mes, dia, trimestre, dia_da_semana
)
SELECT
    -- Cria a PK no formato YYYYMMDD
    TO_CHAR(data_dia, 'YYYYMMDD')::INT AS id_data,
    data_dia AS data_completa,
    EXTRACT(YEAR FROM data_dia) AS ano,
    EXTRACT(MONTH FROM data_dia) AS mes,
    EXTRACT(DAY FROM data_dia) AS dia,
    EXTRACT(QUARTER FROM data_dia) AS trimestre,
    EXTRACT(DOW FROM data_dia) AS dia_da_semana -- DOW = Day of Week (0=Domingo)
FROM (
    -- Busca todas as datas únicas onde ocorreram vendas
    SELECT DISTINCT DATE_TRUNC('day', data_venda) AS data_dia
    FROM Vendas
) AS datas_unicas;
```

Passo 3: Carregando a Tabela Fato (Extract, Transform, Load)

Este é o passo final e mais pesado. Nós unimos `Vendas` e `ItensVenda` para calcular as métricas (`receita_total`) e buscar as chaves estrangeiras corretas.

```
INSERT INTO fato_vendas (
    id_data,
    id_cliente,
    id_produto,
    quantidade_vendida,
```

```
    receita_total
)
SELECT
    -- Converte a data da venda para a FK (YYYYMMDD)
    TO_CHAR(v.data_venda, 'YYYYMMDD')::INT AS id_data,
    v.id_cliente,
    iv.id_produto,

    -- Métricas
    iv.quantidade AS quantidade_vendida,
    (iv.quantidade * iv.preco_unitario) AS receita_total

FROM Vendas v
JOIN ItensVenda iv ON v.id_venda = iv.id_venda;
```

Resultado: Após executar esses scripts, o nosso Star Schema está completo e pronto para ser consultado. A consulta "Receita por categoria e estado"agora é extremamente rápida, como vimos no exemplo conceitual.