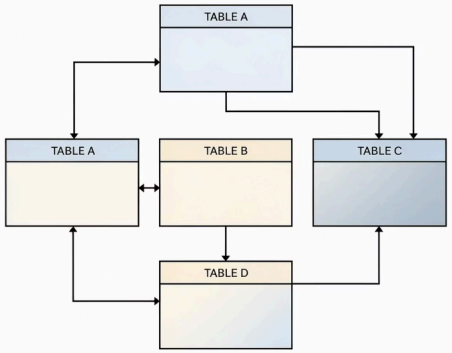
CEFET/RJ

SQL: Structured Query Language

A Linguagem de Consulta Estruturada (SQL) é a linguagem padrão para bancos de dados relacionais. Muitas das características do SQL foram inspiradas na álgebra relacional, tornando-a uma ferramenta poderosa e essencial para gerenciamento de dados.

Eduardo Ogasawara

eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara



FUNDAMENTOS

SQL: Visão Geral

Linguagem Declarativa

O usuário especifica o que deseja, não como obter. Foco nos resultados, não no processo.

DDL

Linguagem de Definição de Dados - define estruturas e esquemas do banco.

DML

Linguagem de Manipulação de Dados - consulta, insere, atualiza e exclui dados.

DDL

Linguagem de Definição de Dados

DDL (Data Definition Language) é o subconjunto do SQL responsável por definir e gerenciar a estrutura do banco de dados.

Esquema das Relações

Define a estrutura das tabelas e seus relacionamentos.

Domínios dos Atributos

Especifica os tipos de dados permitidos para cada coluna.

Restrições de Integridade

Garante a consistência e validade dos dados armazenados.

Índices e Segurança

Otimiza consultas e controla o acesso aos dados.

Tipos Básicos de Domínio em SQL

Tipos Textuais

- char(n) - texto de tamanho fixo
- varchar(n) - texto de tamanho variável

Tipos Numéricos

- int - números inteiros
- float, double - números decimais

Tipos Temporais

- date - datas
- time - horários
- timestamp - data e hora combinados

Tipos textuais e binários variam conforme o SGBD utilizado.

EXEMPLO

Esquema Bancário de Exemplo

As tabelas utilizadas nos exemplos correspondem ao esquema de um sistema bancário completo:

agencia

nomeAgencia, cidadeAgencia, ativo

cliente

nomeCliente, ruaCliente, cidadeCliente

conta

numeroConta, nomeAgencia, saldo

emprestimo

numeroEmprestimo, nomeAgencia, quantia

depositante

nomeCliente, numeroConta

tomador

nomeCliente, numeroEmprestimo

DML

Linguagem de Manipulação de Dados

DML (Data Manipulation Language) é o subconjunto do SQL que permite interagir diretamente com os dados armazenados no banco.

Inserir

Adicionar novos dados

Consultar

Recuperar informações

Atualizar

Modificar dados existentes

Excluir

Remover dados

Projeção Generalizada

Quando não precisamos de todos os atributos, podemos especificar apenas aqueles que desejamos visualizar. Isso é chamado de projeção.

```
SELECT nomeCliente, cidadeCliente
FROM cliente;
```

Equivalente em Álgebra Relacional: $\pi_{\text{nomeCliente, cidadeCliente}}(\text{cliente})$

Este comando seleciona apenas os atributos nomeCliente e cidadeCliente, ignorando ruaCliente.

FILTRAGEM

Cláusula WHERE

A cláusula WHERE permite filtrar tuplas com base em condições específicas, retornando apenas os registros que atendem aos critérios estabelecidos.

Exemplo 1: Filtro Numérico

```
SELECT *
FROM conta
WHERE saldo > 1200;
```

Álgebra Relacional: $\sigma_{saldo>1200}(conta)$

Seleciona apenas contas com saldo superior a 1200.

Exemplo 2: Filtro em Empréstimos

```
SELECT *
FROM emprestimo
WHERE quantia >= 1000;
```

Álgebra Relacional: $\sigma_{quantia\geq1000}(emprestimo)$

Retorna empréstimos com quantia maior ou igual a 1000.

Operadores Especiais em WHERE

1

BETWEEN - Intervalo de Valores

```
SELECT numeroConta, saldo
FROM conta
WHERE saldo BETWEEN 500
AND 2000;
```

Equivalente em Álgebra Relacional:

$$\pi_{numeroConta,saldo}(\sigma_{saldo \geq 500 \wedge saldo \leq 2000}(conta))$$

Seleciona contas com saldo entre 500 e 2000, inclusive.

2

IS NULL - Valores Nulos

```
SELECT nomeCliente
FROM cliente
WHERE ruaCliente IS NULL;
```

Equivalente em Álgebra Relacional:

$$\pi_{nomeCliente}(\sigma_{ruaCliente \text{ IS NULL}}(cliente))$$

Retorna clientes que não possuem endereço cadastrado.

3

LIKE - Correspondência de Padrões

```
SELECT nomeCliente
FROM cliente
WHERE nomeCliente LIKE 'S%';
```

Equivalente em Álgebra Relacional:

$$\pi_{nomeCliente}(\sigma_{nomeCliente \text{ LIKE' } S\%'}(cliente))$$

Busca clientes cujo nome começa com a letra 'S'.

ORDENAÇÃO

Operações de Ordenação

A cláusula ORDER BY permite organizar os resultados em ordem crescente (ASC) ou decrescente (DESC).

```
SELECT numeroConta, saldo
FROM conta
ORDER BY saldo DESC;
```

Equivalente em Álgebra Relacional: $\pi_{numeroConta,saldo}(\text{conta})\ (\text{ordenado por saldo DESC})$

Junção Implícita: Exemplo Simples

Consulta com Duas Tabelas

```
SELECT nomeCliente, numeroConta
FROM depositante, conta
WHERE depositante.numeroConta =
    conta.numeroConta;
```

Equivalente em Álgebra Relacional:

$$\pi_{\text{nomeCliente, numeroConta}}(\text{depositante} \bowtie_{\text{depositante.numeroConta} = \text{conta.numeroConta}} \text{conta})$$

Combina depositantes com suas respectivas contas através do número da conta.

Filtragem Adicional

```
SELECT nomeCliente, saldo
FROM depositante, conta
WHERE depositante.numeroConta =
    conta.numeroConta
 AND saldo > 1000;
```

Equivalente em Álgebra Relacional:

$$\pi_{\text{nomeCliente, saldo}}(\sigma_{\text{saldo} > 1000}(\text{depositante} \bowtie_{\text{depositante.numeroConta} = \text{conta.numeroConta}} \text{conta}))$$

Adiciona filtro para mostrar apenas contas com saldo superior a 1000.

Junção Implícita vs Explícita

Ambas as sintaxes produzem o mesmo resultado, mas a junção explícita oferece maior clareza. Veja o exemplo de seleção de clientes e saldos da agência Perryridge:

Junção Implícita

```
SELECT d.nomeCliente, c.saldo
FROM depositante d, conta c
WHERE d.numeroConta = c.numeroConta
  AND c.nomeAgencia = 'Perryridge';
```

$$\pi_{nomeCliente,\, saldo}\left(\sigma_{depositante.numeroConta = conta.numeroConta \,\wedge\, conta.nomeAgencia = 'Perryridge'}\left(depositante \times conta\right)\right)$$

Junção Explícita

```
SELECT d.nomeCliente, c.saldo
FROM depositante d
JOIN conta c
  ON d.numeroConta = c.numeroConta
WHERE c.nomeAgencia = 'Perryridge';
```

$$\pi_{nomeCliente, saldo}(\sigma_{nomeAgencia = 'Perryridge'}(depositante \bowtie_{depositante.numeroConta = conta.numeroConta} conta))$$

As expressões algébricas são **equivalentes em resultado**, mas **não equivalentes computacionalmente**. Por esse motivo, os sistemas de banco de dados realizam **transformações algébricas** para obter uma forma **mais eficiente**, como a junção direta, antes da execução da consulta.

OPERAÇÕES DE CONJUNTO

UNION: Combinando Resultados

O operador UNION combina os resultados de duas ou mais consultas, eliminando duplicatas automaticamente.

```
SELECT nomeCliente
FROM depositante
UNION
SELECT nomeCliente
FROM tomador;
```

Equivalente em Álgebra Relacional:

$$\pi_{\text{nomeCliente}}(\text{depositante}) \cup \pi_{\text{nomeCliente}}(\text{tomador})$$

Este exemplo retorna uma lista única de todos os clientes que são depositantes ou tomadores de empréstimo, sem repetições.

BOAS PRÁTICAS

Dicas para Consultas Eficientes

01

Use Aliases

Simplifique referências a tabelas com nomes curtos e descritivos.

02

Prefira JOIN Explícito

Torna o código mais legível e facilita a manutenção.

03

Especifique Colunas

Evite SELECT * em produção para melhor performance.

04

Filtre Adequadamente

Use WHERE para reduzir o volume de dados processados.

05

Ordene com Propósito

ORDER BY tem custo computacional, use quando necessário.

Estrutura Completa de uma Consulta SQL

Uma consulta SQL bem estruturada segue uma ordem lógica de cláusulas, cada uma com sua função específica:

SELECT

Define quais colunas serão retornadas

Equivalente em Álgebra Relacional: π (Projeção das colunas)

FROM

Especifica as tabelas fonte dos dados

Equivalente em Álgebra Relacional: R (Relação/Tabela)

JOIN...ON

Combina tabelas relacionadas (opcional)

Equivalente em Álgebra Relacional: $\bowtie$ (Junção)

WHERE

Filtra registros com base em condições

Equivalente em Álgebra Relacional: σ (Seleção)

ORDER BY

Organiza os resultados finais

Equivalente em Álgebra Relacional: Não há um operador direto; a álgebra relacional trata conjuntos não ordenados.

RESUMO

Conceitos-Chave do SQL

Linguagem Declarativa

Foco no "o quê" obter, não no "como" obter. SQL abstrai a complexidade do acesso aos dados.

DDL e DML

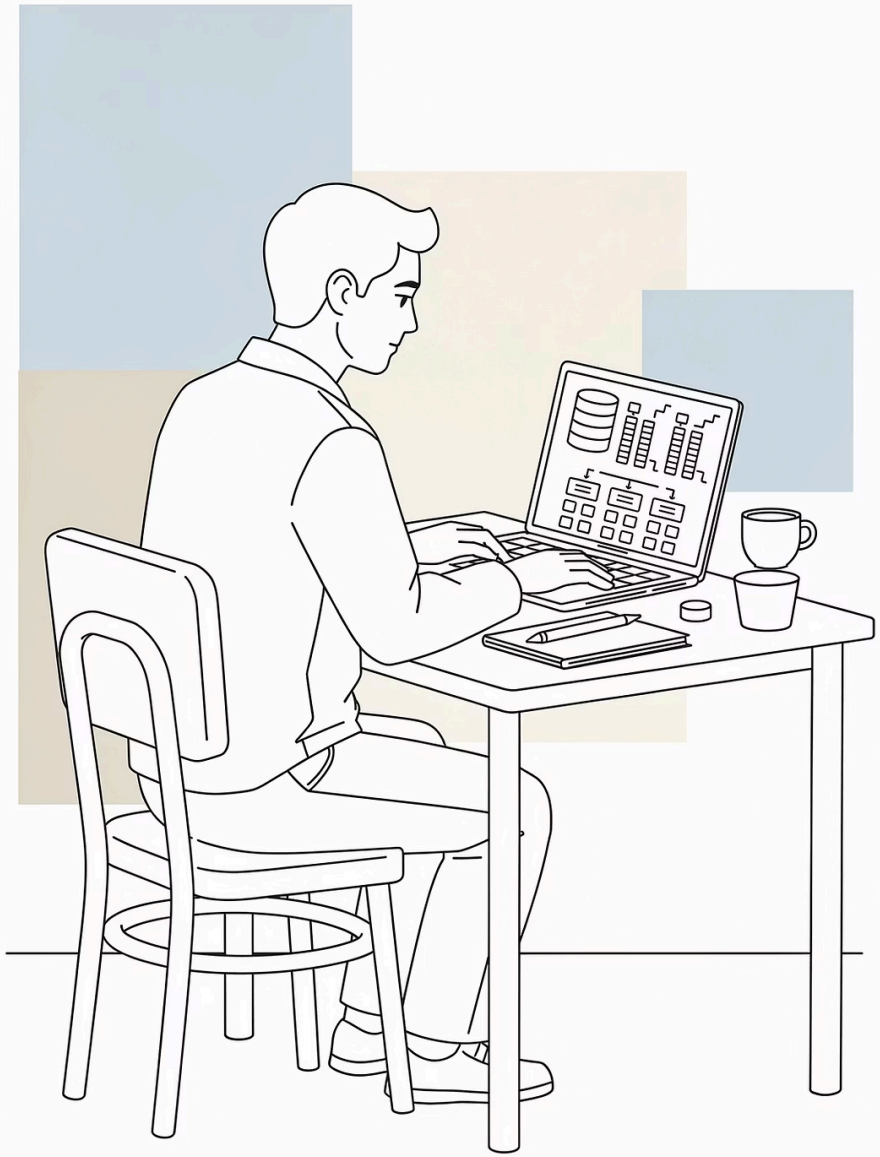
Dois subconjuntos principais: definição de estruturas (DDL) e manipulação de dados (DML).

Filtragem e Projeção

WHERE filtra linhas, SELECT escolhe colunas - controle total sobre os resultados.

Junções

Combinam dados de múltiplas tabelas através de relacionamentos, essencial para bancos relacionais.



Próximos Passos

Agora que você domina os fundamentos do SQL, está pronto para explorar conceitos mais avançados e aplicar esse conhecimento em projetos reais.

1

Pratique Regularmente

Execute consultas em bancos de dados reais para consolidar o aprendizado.

2

Explore Funções Agregadas

COUNT, SUM, AVG, MAX, MIN e GROUP BY para análises mais complexas.

3

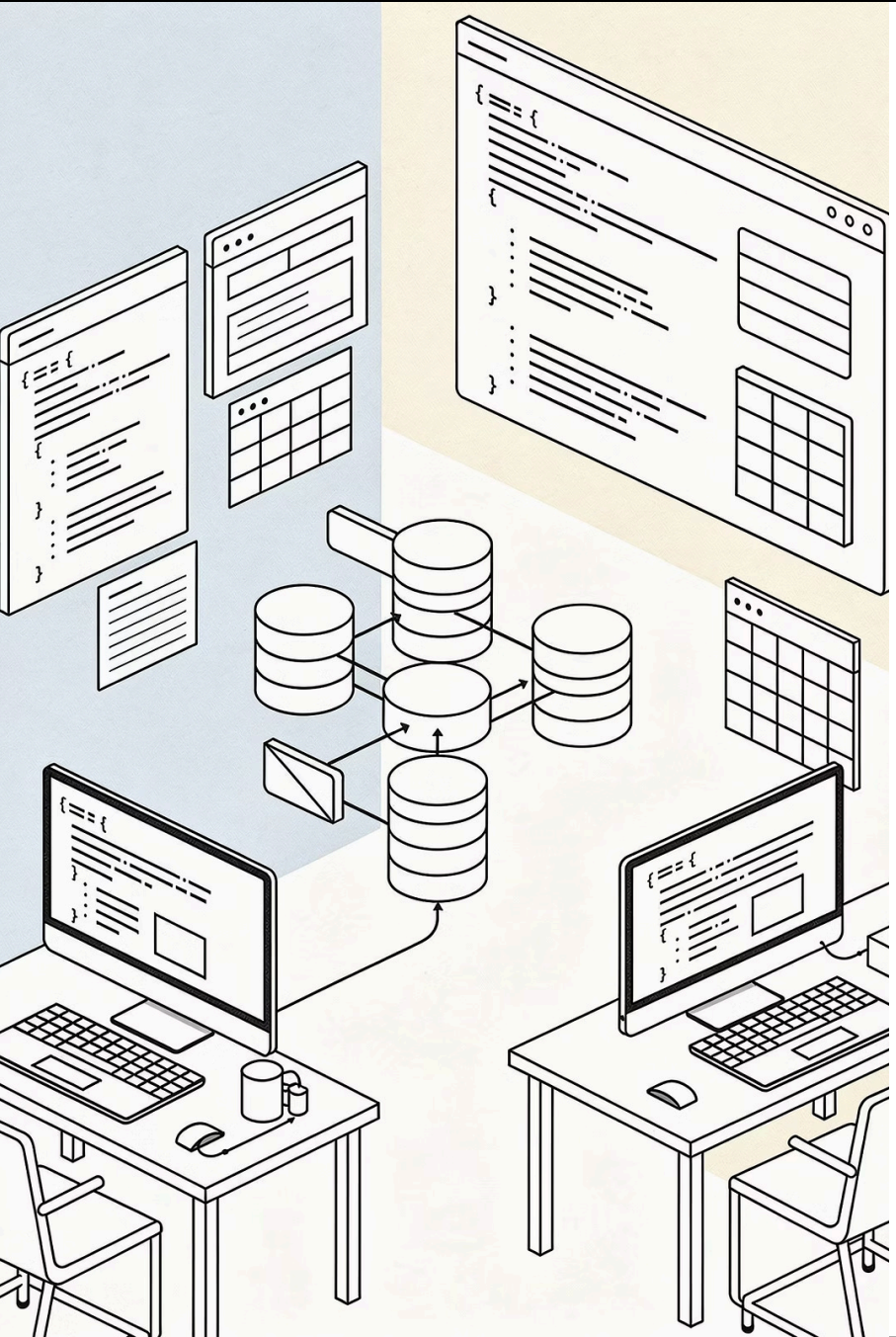
Aprenda Subconsultas

Consultas aninhadas para resolver problemas mais sofisticados.

4

Estude Otimização

Índices, planos de execução e técnicas para melhorar performance.



Operações Avançadas em SQL

Domine consultas complexas, funções agregadas e operações de conjunto para análise de dados eficiente em bancos de dados relacionais.

FUNÇÕES AGREGADAS

Análise de Dados com Agregações

Funções agregadas permitem realizar cálculos sobre conjuntos de dados, transformando múltiplas linhas em resultados únicos e significativos.

SUM

Calcula a soma total de valores numéricos em uma coluna.

```
SELECT SUM(saldo)
FROM conta;
```

Álgebra Relacional: $\Gamma_{SUM(saldo)}(conta)$

COUNT

Conta o número total de registros ou valores não nulos.

```
SELECT COUNT(*)
FROM conta;
```

Álgebra Relacional: $\Gamma_{COUNT(*)}(conta)$

AVG

Calcula a média aritmética de valores numéricos.

```
SELECT AVG(saldo)
FROM conta;
```

Álgebra Relacional: $\Gamma_{AVG(saldo)}(conta)$

Agrupamento de Dados com GROUP BY

O GROUP BY organiza registros em grupos baseados em valores de colunas específicas, permitindo aplicar funções agregadas a cada grupo individualmente.

```
SELECT nomeAgencia, AVG(saldo)
FROM conta
GROUP BY nomeAgencia;
```

Álgebra Relacional: ${}_{nomeAgencia}\Gamma_{AVG(saldo)}(conta)$

Esta consulta calcula o saldo médio para cada agência bancária, agrupando todas as contas por agência e aplicando a função AVG a cada grupo.

FILTRAGEM AVANÇADA

HAVING: Filtragem Pós-Agrupamento

Diferença entre WHERE e HAVING

WHERE filtra registros **antes** do agrupamento, enquanto HAVING filtra grupos **após** a agregação.

Exemplo Prático

```
SELECT nomeAgencia, AVG(saldo)
FROM conta
GROUP BY nomeAgencia
HAVING AVG(saldo) > 1000;
```

Equivalente em Álgebra Relacional:

$$\pi_{nomeAgencia,\ \mathrm{AVG}(saldo)} \left(\sigma_{\mathrm{AVG}(saldo) > 1000} \left(\Gamma^{\mathrm{AVG}(saldo)}_{nomeAgencia}(conta) \right) \right)$$

Retorna apenas agências com saldo médio superior a 1000.

TIPOS DE SUBCONSULTAS

Consultas Não Correlacionadas

Características

- Executa independentemente da consulta externa
- Executada apenas uma vez
- Mais eficiente em termos de desempenho
- Resultado é reutilizado para todas as linhas

Exemplo: Contas Acima da Média

```
SELECT numeroConta
FROM conta
WHERE saldo > (
    SELECT AVG(saldo)
    FROM conta
);
```

Álgebra Relacional: $\pi_{numeroConta}\left(\sigma_{saldo>\Gamma_{AVG(saldo)}(conta)}(conta)\right)$

Identifica contas com saldo superior à média geral.

✓ EXISTS

Operador EXISTS

O operador EXISTS verifica se uma subconsulta retorna algum resultado. Retorna TRUE se a subconsulta encontrar pelo menos uma linha correspondente.

Clientes com Empréstimos

```
SELECT nomeCliente
FROM cliente c
WHERE EXISTS (
    SELECT *
    FROM tomador t
    WHERE t.nomeCliente = c.nomeCliente
);
```

Retorna clientes que possuem empréstimos ativos no banco.

Vantagem de Performance

EXISTS para de procurar assim que encontra a primeira correspondência, tornando-o mais eficiente que COUNT(*) > 0 em grandes conjuntos de dados.

Equivalente em Álgebra Relacional: $\pi_{nomeCliente}(Cliente \ltimes_{nomeCliente} Tomador)$

NOT EXISTS

Operador NOT EXISTS

NOT EXISTS é o inverso de EXISTS, retornando TRUE quando a subconsulta NÃO encontra nenhuma linha correspondente. Ideal para identificar ausências ou exclusões.

Clientes Sem Empréstimos

Identifica clientes que não possuem empréstimos registrados no sistema.

Análise de Risco

Útil para segmentar clientes por perfil de crédito e comportamento financeiro.

```
SELECT nomeCliente
FROM cliente c
WHERE NOT EXISTS (
   SELECT *
   FROM emprestimo e
   JOIN tomador t ON e.numeroEmprestimo = t.numeroEmprestimo
   WHERE t.nomeCliente = c.nomeCliente
);
```

Equivalente em Álgebra Relacional: $\pi_{nomeCliente}(Cliente) - \pi_{nomeCliente}(Cliente \bowtie (Tomador \bowtie_{Tomador.numeroEmprestimo=Emprestimo.numeroEmprestimo} Emprestimo))$

Esta consulta combina NOT EXISTS com JOIN para verificar a ausência de empréstimos associados a cada cliente.

Referências

Elmasri & Navathe

Fundamentals of Database Systems

Pearson, 2016

Referência abrangente sobre fundamentos de sistemas de bancos de dados, cobrindo aspectos teóricos e práticos.

Korth, Sudarshan & Silberschatz

Database System Concepts

McGraw-Hill, 2019

Texto fundamental que serviu como base para a maioria dos exemplos apresentados nesta apresentação.

Özsu & Valduriez

Principles of Distributed Database Systems

Springer Nature, 2019

Obra especializada em sistemas de bancos de dados distribuídos, essencial para compreensão avançada.

Nota: Os conceitos e exemplos apresentados baseiam-se principalmente na literatura clássica de sistemas de bancos de dados, em especial *Database System Concepts* e *Fundamentals of Database Systems*.