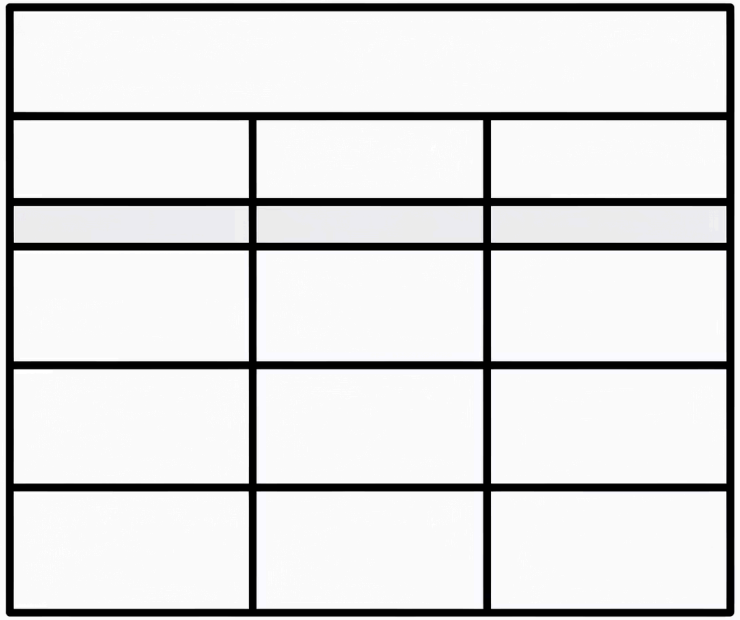


CEFET/RJ

MATERIAL COMPLEMENTAR

Introdução ao Modelo Relacional

O modelo relacional é a base fundamental para a maioria dos sistemas de gerenciamento de banco de dados modernos. Neste capítulo, exploraremos os conceitos essenciais que sustentam esta estrutura elegante e poderosa.

Eduardo Ogasawara

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

Schema e Instância de Relação

Conceitos Fundamentais

Uma relação é definida por seu **schema**, que especifica sua estrutura. O schema lista os atributos $A_1, A_2, \ldots, A_n$ que compõem a relação.

A notação formal é: $R = (A_1, A_2, \ldots, A_n)$

Exemplo: Schema instructor

instructor = (*ID, name, dept_name, salary*)

Este schema define que a relação instructor possui quatro atributos que descrevem cada instrutor no banco de dados.

Instância e Tuplas

Uma **instância de relação** $r(R)$ definida sobre o schema R é denotada por $r(R)$. Representa os valores atuais da relação em um determinado momento.

Os valores atuais são especificados por uma **tabela**, onde cada linha representa um elemento da relação.

Um elemento **t** da relação **r** é chamado de **tupla** e é representado por uma **linha** na tabela. Cada tupla contém valores específicos para todos os atributos definidos no schema.

Atributos e Domínios

Domínio de Atributos

O conjunto de valores permitidos para cada atributo é chamado de **domínio** do atributo. Por exemplo, o domínio do atributo *salary* pode ser números positivos, enquanto o domínio de *dept_name* seria strings de texto.

Atomicidade de Valores

Os valores de atributos são normalmente exigidos como **atômicos**, ou seja, indivisíveis. Isso significa que cada valor deve ser uma unidade única e não pode ser decomposto em partes menores dentro do modelo relacional.

Valor Null

O valor especial **null** é membro de todos os domínios. Indica que o valor é "desconhecido" ou não aplicável. Por exemplo, um instrutor pode não ter um endereço de email registrado ainda.

Complicações com Null

O valor null causa complicações significativas na definição de muitas operações de banco de dados. Comparações com null requerem lógica especial de três valores (verdadeiro, falso, desconhecido), e agregações devem decidir como tratar valores ausentes.

Relações São Não Ordenadas

Conceito Fundamental

Uma propriedade crucial das relações é que a **ordem das tuplas é irrelevante**. As tuplas podem ser armazenadas em qualquer ordem arbitrária sem afetar o significado da relação.

Isso contrasta com arrays ou listas em programação, onde a posição de um elemento é significativa. Em bancos de dados relacionais, a identidade de uma tupla é determinada por seus valores, não por sua posição.

Esta característica permite que o sistema de gerenciamento de banco de dados otimize o armazenamento e a recuperação de dados sem se preocupar com a preservação de uma ordem específica.

Exemplo: Relação instructor com tuplas não ordenadas

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
33456	Gold	Physics	87000

Observe que as mesmas tuplas podem aparecer em qualquer ordem na tabela. O significado dos dados permanece idêntico independentemente de como as linhas estão organizadas fisicamente.

Schema e Instância de Banco de Dados

Database Schema

O **schema do banco de dados** representa a estrutura lógica completa do banco de dados. Define todas as relações, seus atributos, tipos de dados, e restrições. É como o projeto arquitetônico de um edifício - permanece relativamente estável ao longo do tempo.

Database Instance

A **instância do banco de dados** é um snapshot dos dados no banco de dados em um determinado instante no tempo. Representa o estado atual de todos os dados armazenados. Ao contrário do schema, a instância muda frequentemente conforme dados são inseridos, atualizados ou excluídos.

Exemplo de Schema

instructor (*ID, name, dept_name, salary*)

Este schema define a estrutura da relação instructor com quatro atributos específicos.

Exemplo de Instância

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
33456	Gold	Physics	87000

Chaves em Bancos de Dados Relacionais

As chaves são fundamentais para identificar tuplas únicas e estabelecer relações entre tabelas. Vamos explorar os diferentes tipos de chaves e seus papéis.

Superkey

Seja $K \subseteq R$. **K** é uma **superkey** de *R* se os valores de *K* são suficientes para identificar uma tupla única de cada relação possível *r(R)*.

Exemplo: Na relação *instructor*, tanto {*ID*} quanto {*ID, name*} são superkeys, pois ambos podem identificar exclusivamente cada instrutor.

Candidate Key

Uma superkey *K* é uma **candidate key** se *K* é minimal - ou seja, nenhum subconjunto próprio de *K* é uma superkey.

Exemplo: {*ID*} é uma candidate key para *instructor* porque é suficiente e mínima. {*ID, name*} não é candidate key porque contém atributos desnecessários.

Primary Key

Uma das candidate keys é selecionada para ser a **primary key**. A escolha depende de fatores como simplicidade, estabilidade e significado no domínio do problema.

Questão: Qual candidate key deve ser escolhida? Geralmente, preferem-se chaves mais simples e que não mudam com frequência.

Foreign Key

Uma **foreign key** é uma restrição que garante que um valor em uma relação (relação **referenciante**) deve aparecer na primary key de outra relação (relação **referenciada**).

Exemplo: *dept_name* em *instructor* é uma foreign key que referencia a relação *department*, garantindo que cada instrutor pertença a um departamento válido.

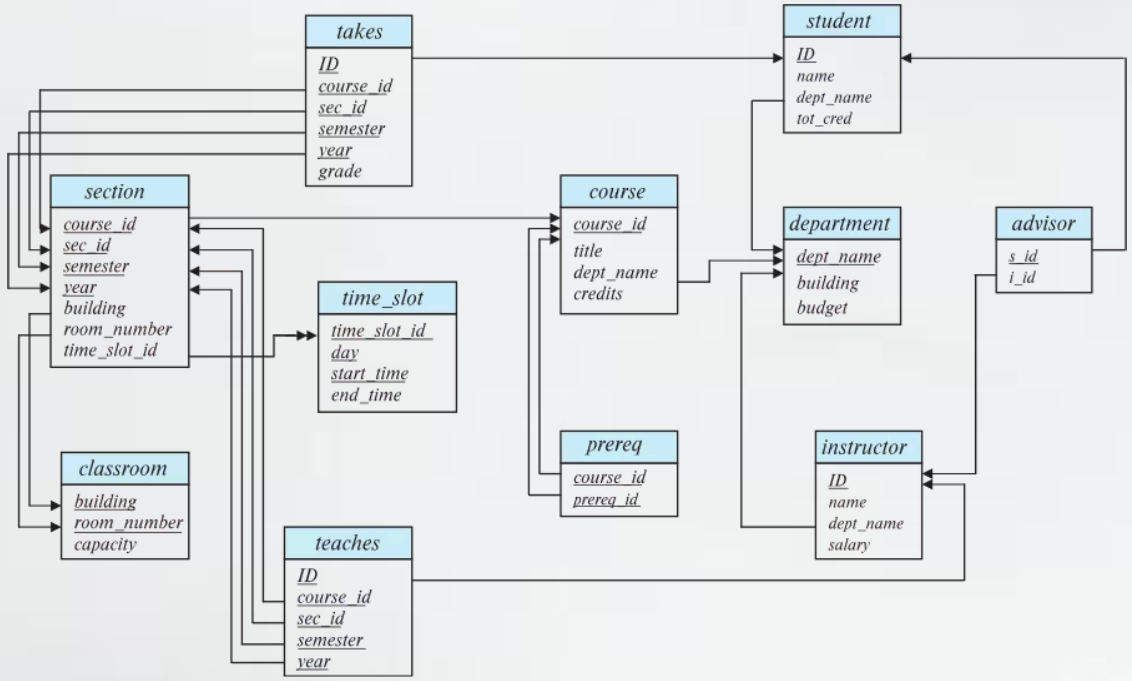


Diagrama de Esquema para Banco de Dados Universitário

O diagrama de esquema fornece uma representação visual completa das relações no banco de dados e seus relacionamentos. Cada caixa representa uma relação com seus atributos, e as setas indicam foreign keys que conectam as relações.

Componentes Principais

- **Relações**: Cada caixa representa uma tabela no banco de dados
- **Atributos**: Listados dentro de cada caixa
- **Primary Keys**: Sublinhados para identificação rápida
- **Foreign Keys**: Indicadas por setas conectando relações

Interpretação

Este diagrama mostra como diferentes entidades do sistema universitário - instrutores, departamentos, cursos, estudantes - estão interconectadas por meio de foreign keys, formando uma rede coesa de informações relacionadas.

Linguagens de Consulta Relacional

Linguagens Procedurais vs. Declarativas

Linguagens **procedurais** especificam como obter os dados (passo a passo). Linguagens **declarativas** ou **não-procedurais** especificam quais dados queremos, deixando o sistema decidir como obtê-los.

Linguagens "Puras"

Três linguagens relacionais puras formam a base teórica:

- **Álgebra Relacional** (procedural)
- **Cálculo Relacional de Tuplas**
- **Cálculo Relacional de Domínio**

Estas três linguagens são equivalentes em poder computacional - qualquer consulta expressável em uma pode ser expressa nas outras.

Foco

Concentraremos nosso estudo na **álgebra relacional**, uma linguagem procedural fundamental que fornece a base para entender operações de banco de dados.

Características da Álgebra Relacional

- Não é equivalente à Máquina de Turing
- Consiste em 6 operações básicas
- Forma a base de linguagens SQL modernas

Álgebra Relacional: Visão Geral

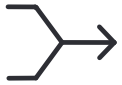
A álgebra relacional é uma linguagem procedural que consiste de um conjunto de operações que recebem uma ou duas relações como entrada e produzem uma nova relação como resultado. Esta propriedade de fechamento é fundamental para compor operações complexas.

Select (σ)

Seleciona tuplas que satisfazem um predicado específico. Filtra linhas da relação com base em uma condição.

Project (Π)

Retorna a relação com certos atributos removidos. Seleciona colunas específicas da relação.



Union ($\cup$)

Combina tuplas de duas relações, eliminando duplicatas. Requer que as relações sejam compatíveis.

Set Difference ($-$)

Retorna tuplas que estão na primeira relação mas não na segunda. Realiza subtração de conjuntos.

Cartesian Product ($\times$)

Combina cada tupla da primeira relação com cada tupla da segunda. Cria todas as combinações possíveis.

Rename (ρ)

Renomeia uma relação e/ou seus atributos. Útil para evitar ambiguidades em operações complexas.

Operação Select (σ)

Definição e Notação

A operação **select** seleciona tuplas que satisfazem um predicado específico. A notação formal é:

$\sigma_p(r)$

onde *p* é chamado de **predicado de seleção** e *r* é a relação.

> O predicado pode incluir comparações ($=, \neq, <, >, \leq, \geq$) e conectivos lógicos ($\wedge$ para AND, $\vee$ para OR, $\neg$ para NOT).

Características Importantes

- Opera sobre linhas (tuplas) da relação
- Não modifica atributos, apenas filtra tuplas
- Sempre retorna um subconjunto da relação original

Exemplo Prático

Selecionar tuplas da relação *instructor* onde o instrutor está no departamento "Physics":

Consulta:

$\sigma_{\text{dept\_name}=\text{"Physics"}}(\text{instructor})$

Resultado:

ID	name	dept_name	salary
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Observe que apenas as tuplas que satisfazem a condição especificada são incluídas no resultado.

Operação Project (Π)

1

Relação Completa

Contém todos os atributos definidos no schema

2

Aplicação do Project

$\Pi_{A_1,A_2,\ldots,A_k}(r)$

3

Relação Resultante

Apenas k colunas selecionadas permanecem

Definição

Uma operação unária que retorna sua relação argumento com certos atributos removidos. A notação é:

$\Pi_{A_1,A_2,A_3,\ldots,A_k}(r)$

onde $A_1, A_2, \ldots, A_k$ são nomes de atributos e *r* é um nome de relação.

Como Funciona

O resultado é definido como a relação de *k* colunas obtida ao **apagar as colunas que não estão listadas**.

Propriedade Importante

Linhas duplicadas são removidas do resultado, pois relações são conjuntos matemáticos e não podem conter duplicatas.

Aplicações Práticas

- Reduzir o tamanho dos resultados removendo colunas desnecessárias
- Focar apenas nos atributos relevantes para uma análise
- Preparar dados para operações subsequentes
- Melhorar desempenho ao trabalhar com menos dados

Exemplo da Operação Project

Objetivo

Eliminar o atributo *dept_name* da relação *instructor*, mantendo apenas *ID*, *name* e *salary*.

Consulta em Álgebra Relacional

$$\Pi_{\text{ID, name, salary}}(\text{instructor})$$

O Que Acontece

1. A operação examina a relação *instructor*
2. Remove a coluna *dept_name*
3. Mantém apenas as três colunas especificadas
4. Remove quaisquer linhas duplicadas (se existirem)

Esta operação é extremamente útil quando queremos trabalhar com um subconjunto dos atributos disponíveis, simplificando a visualização e análise dos dados.

Resultado da Operação

ID	name	salary
22222	Einstein	95000
12121	Wu	90000
32343	El Said	60000
45565	Katz	75000
33456	Gold	87000

Observe que a tabela resultante contém apenas as três colunas especificadas na operação project. A informação sobre departamentos foi completamente removida.

Operação Join

Passo 1: Produto Cartesiano

$\text{instructor} \times \text{teaches}$ associa cada tupla de instructor com cada tupla de teaches

1

2

Problema Identificado

A maioria das linhas resultantes contém informações sobre instrutores que NÃO ensinaram um curso específico

Passo 2: Aplicar Select

$\sigma_{\text{instructor.id} = \text{teaches.id}} (\text{instructor} \times \text{teaches})$

3

4

Resultado Final

Apenas tuplas que relacionam instrutores aos cursos que eles efetivamente ensinaram

O Desafio do Produto Cartesiano

O produto cartesiano sozinho cria **todas as combinações possíveis** entre as duas relações. Se *instructor* tem 100 tuplas e *teaches* tem 200 tuplas, o resultado teria 20.000 tuplas!

Porém, a maioria dessas combinações é sem sentido - representam associações entre instrutores e cursos que eles nunca ensinaram.

A Solução: Join

Para obter apenas as tuplas relevantes de "$\text{instructor} \times \text{teaches}$" que relacionam instrutores aos cursos que eles ensinaram, aplicamos uma condição de seleção:

$$\sigma_{\text{instructor.id}=\text{teaches.id}}(\text{instructor} \times \text{teaches})$$

Esta expressão é tão comum que chamamos de **join**, uma das operações mais importantes em bancos de dados relacionais.

Consultas Equivalentes em Álgebra Relacional

Um conceito fundamental em álgebra relacional é que **existe mais de uma maneira de escrever uma consulta**. Consultas diferentes podem ser equivalentes - produzindo o mesmo resultado em qualquer banco de dados.

Problema de Exemplo

Encontrar informações sobre cursos ensinados por instrutores do departamento de Physics com salário maior que 90.000

Consulta 1: Seleção Combinada

$$\sigma_{\text{dept\_name}=\text{"Physics"} \wedge \text{salary} > 90000}(\text{instructor})$$

Esta consulta aplica **ambas as condições simultaneamente** em uma única operação de seleção.

Consulta 2: Seleções Sequenciais

$$\sigma_{\text{dept\_name}=\text{"Physics"}}(\sigma_{\text{salary} > 90000}(\text{instructor}))$$

Esta consulta aplica **duas seleções em sequência**: primeiro filtra por salário, depois por departamento.

Equivalência

As duas consultas **não são idênticas** em estrutura, mas são **equivalentes** - produzem o mesmo resultado em qualquer banco de dados possível.

Importância da Equivalência

O sistema de gerenciamento de banco de dados pode escolher a representação mais eficiente para executar, mesmo que você escreva a consulta de forma diferente. Isso é fundamental para otimização de consultas!

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*.