

# O Modelo Relacional

Feliz Ribeiro Gouveia  
fribeiro@ufp.edu.pt  
UFP

# Modelo Relacional de dados

- Um modelo refere-se a três aspectos fundamentais dos dados:
  - a sua estrutura
  - a sua integridade
  - a sua manipulação
- O Modelo Relacional (Ted Codd, IBM, 1969) admite **relações** como única estrutura

# Relações

- Uma relação é definida pela sua estrutura (esquema)
- O esquema de uma relação é o conjunto do nome dos seus atributos (colunas)
- Todos os tuplos de uma relação têm todos os atributos do esquema
- Uma base de dados define-se como um conjunto de esquemas

# Relações: exemplo

**aluno** {numero, nome, apelido, morada}

- Os valores possíveis de uma relação são a sua extensão:
  - {1000, Luis, Sousa, R. Direita}
  - {1010, Ana, Ribeiro, R. de Cima}
- Se for necessário, escreve-se:
  - {numero:1000, nome:Luis, apelido:Sousa, morada:R. Direita}
- Na prática, torna-se mais simples

# Propriedades de um sistema relacional

- Só existem tabelas para representar os dados
  - Logicamente, não fisicamente
- Os valores dos atributos são todos explícitos
  - Não existem apontadores
- Todos os valores são atômicos (escalares)
  - Não há grupos de valores (Grupos de Repetição)

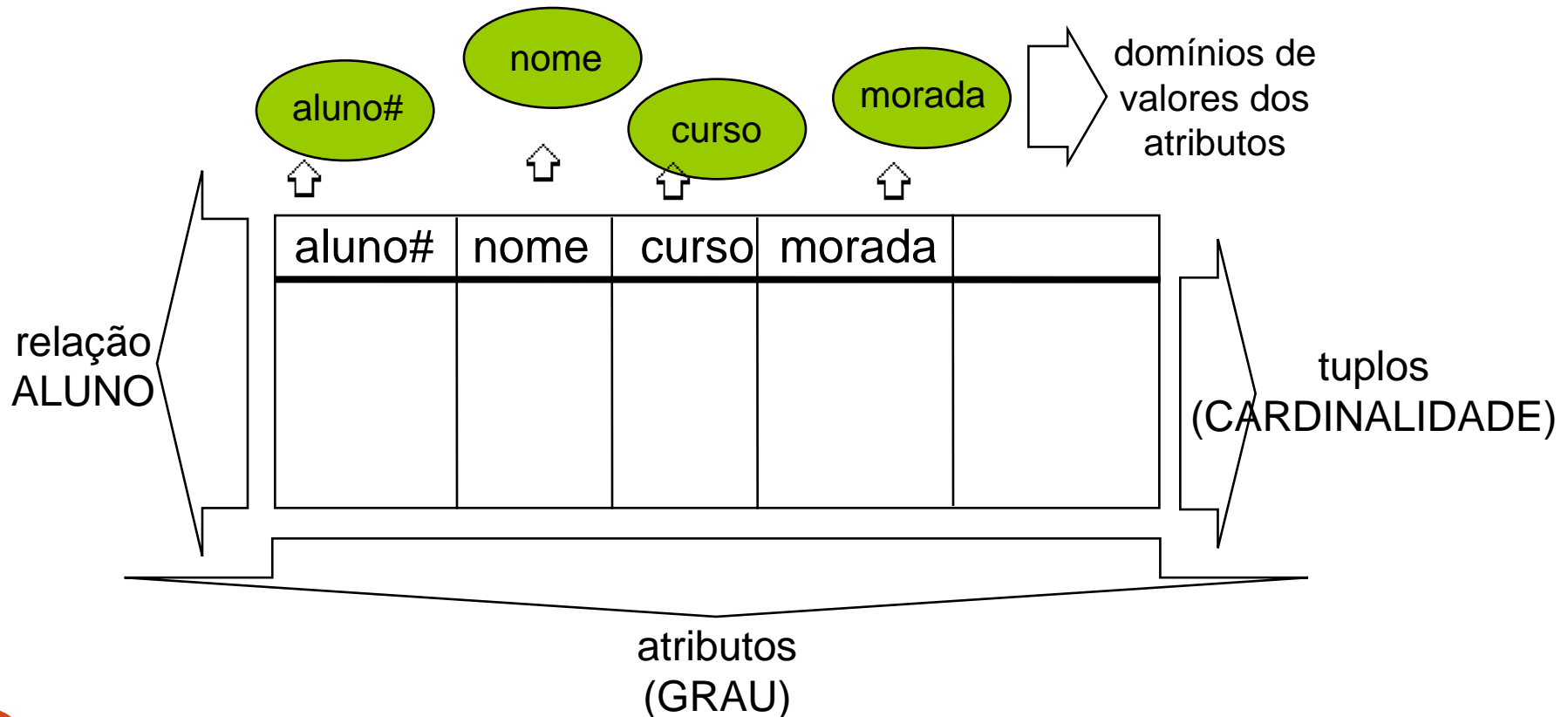
# Mais propriedades

- A ordem das linhas não interessa
  - o acesso não é por posição
- A ordem das colunas não interessa
  - todas as colunas têm um nome diferente
- Não podem existir linhas duplicadas
  - é sempre possível distingui-las pelos valores de algumas das células

# Terminologia relacional

- Relação => Tabela
- Tuplo => registo, linha
- Atributo => campo, propriedade, coluna
- Domínio: conjunto de valores admissíveis para os atributos de uma relação

# Terminologia (cont)





# Regras de integridade

- Do domínio: uma coluna só pode assumir valores do seu domínio
- Chaves: existe um conjunto de atributos único para cada tuplo: superchaves
  - Chave candidata: superchave que não pode ser reduzida
  - Chave primária: uma das chaves candidatas
  - Chave estrangeira: conjunto que é chave noutra relação
- De Entidade: a chave primária nunca é nula

# Regras de integridade

- Gerais: dependem da aplicação
  - Podem ser definidas na base de dados
  - Exemplo: custo > 0, quantidade > 0
- As restrições gerais e de domínio podem ser implementadas com CHECK e CREATE ASSERTION
  - CHECK efetua verificações numa tabela
  - ASSERTION pode envolver várias tabelas
    - Mas não é implementada em quase nenhum SGBD...

# Regras de integridade

- As restrições são verificadas de cada vez que uma linha é inserida ou alterada
- Por vezes é útil diferir a verificação (problema do “ovo e da galinha”: inserimos numa tabela o valor que devia existir noutra, e como não existe ainda não podemos inserir)
- Ao criar uma restrição podemos definir se queremos verificação diferida
  - a verificação de NOT NULL e CHECK não pode ser diferida

# Regras de integridade

- Podemos especificar ao criar a restrição:
  - deferrable (not deferrable): significa que a verificação da restrição pode ser diferida para o fim da transação, usando SET CONSTRAINTS dentro de uma transação
  - initially deferred (initially immediate): significa que a verificação da restrição é diferida para o fim da transação
- Uma transação começa por BEGIN e termina por COMMIT (confirmar) ou ROLLBACK (anular)

# Chaves

numero	nome	apelido	morada	
1000	Luis			
1010	Ana			
1020	Paulo			

Chave primária:  
{numero}

# Exemplo

- CREATE TABLE aluno (id serial PRIMARY KEY, nome text, apelido text, data\_nasc date, genero char(1) CHECK (genero = 'F' OR genero = 'M'));
- Definem-se duas restrições nesta relação:
  - PRIMARY KEY: chave primária
  - CHECK: restrição de domínio
- Outro tipo de restrições
  - NOT NULL: coluna que não admite valores nulos
  - UNIQUE: coluna que não admite valores repetidos
  - REFERENCES : coluna é chave estrangeira

# Exemplo (1)

- CREATE TABLE matricula (aluno\_id int REFERENCES aluno(id), disciplina\_id int REFERENCES disciplina(id), PRIMARY KEY(aluno\_id, disciplina\_id);
- CREATE ASSERTION tamanho\_turma CHECK (**NOT EXISTS** (SELECT COUNT(\*) FROM turma GROUP BY disc\_id HAVING COUNT(\*) > 25));
  - Sempre que se altera TURMA , verifica que o número de inscritos não ultrapassa os 25
  - A maioria dos SGBD não implementa asserções

## Exemplo 2

- `CREATE DOMAIN semestre AS INTEGER  
CHECK (VALUE = 1 OR VALUE = 2);`
- Este novo domínio pode ser usado:
  - `CREATE TABLE disciplina (id serial, semestre  
SEMESTRE,.....)`



# Triggers

- A verificação de restrições também pode ser feita com um TRIGGER
- Um *trigger* tem duas partes:
  - Uma função, definida com CREATE FUNCTION
  - A ligação desta função a uma tabela, feita com CREATE TRIGGER
- Um *trigger* pode disparar uma vez para cada linha (row level), ou uma vez para o comando (statement level)
- A sintaxe de CREATE TRIGGER é rica, com muitas possibilidades

# Triggers (1)

- Um trigger pode disparar ANTES ou DEPOIS de um dado evento (INSERT, DELETE, UPDATE)
- A função tem acesso a NEW (nova linha, após o evento) e a OLD (antiga linha, antes do evento ser executado) → apenas em “row level” triggers
- NEW pode ser modificada se a condição for BEFORE
- Podem existir vários triggers para a mesma tabela e evento

# Triggers (2)

- A função de um trigger que dispare ANTES para cada linha pode devolver NULL (a ação não é executada), ou a linha (NEW) no caso do evento ser INSERT ou UPDATE
- Se o trigger dispara DEPOIS, a função deve devolver NULL
- Um trigger que propaga informação para outras tabelas (ex. log) deve disparar DEPOIS para garantir que todos os ANTES já dispararam

# Triggers: definir a função

```
CREATE FUNCTION VerificaCurso()  
    RETURNS TRIGGER AS  
    $$BEGIN IF NOT (SELECT curso_id from  
disciplina WHERE id = NEW.disciplina_id) = (SELECT  
curso_id FROM matricula WHERE  
aluno_id=NEW.aluno_id)  
        THEN RAISE EXCEPTION 'Erro:  
disciplina de outro curso.';  
    END IF; RETURN NEW;  
END$$  
LANGUAGE plpgsql;
```

# Triggers: associar à tabela e evento

- CREATE TRIGGER InscricaoCorrecta BEFORE  
UPDATE OR INSERT ON inscrito  
FOR EACH ROW  
EXECUTE PROCEDURE VerificaCurso();
- Este *trigger* dispara para cada linha inserida ou modificada
- Se fosse definido para o comando (statement level) dispararia uma única vez, mesmo que o comando resultasse em nenhuma linha inserida ou modificada

# Triggers

```
CREATE FUNCTION log_mudanca_curso()  
  RETURNS trigger AS  
$$BEGIN  
  IF NEW.curso_id <> OLD.curso_id THEN  
    INSERT INTO mudancas_curso(aluno_id,  
      curso_id, data) VALUES (OLD.aluno_id,  
      NEW.curso_id ,now());  
  END IF;  
  RETURN NULL; -- é um trigger AFTER  
END$$ LANGUAGE plpgsql
```

# Triggers

```
CREATE TRIGGER regista_mudancas  
  AFTER UPDATE  
  ON matricula  
  FOR EACH ROW  
  EXECUTE PROCEDURE log_mudanca_curso();
```

# Vistas

- Permite apresentar ao utilizador final a informação de que este necessita, evitando expor a relação, ou as relações, de base
- É o resultado da execução de uma consulta
- Evita expor no nível conceptual a informação do nível lógico
- As vistas podem ser definidas a partir de uma ou mais relações
  - E devem ser atualizadas sempre que essas relações são atualizadas
  - Não são, por isso, materializadas (chamam-se tabelas virtuais)



# Vistas (exemplo)

- `CREATE MATERIALIZED VIEW` permite materializar em memória a vista; para se verem os resultados mais recentes deve-se pedir para atualizar a vista
- `CREATE VIEW` opcionais `AS SELECT * FROM disciplinas WHERE tipo = 'opcional'`;
- A vista permite esconder detalhes das relações de base
- Nem todos os SGBD permitem atualizar vistas

# Álgebra Relacional

- Linguagem procedimental que usa expressões algébricas
  - Diz-se o “que se quer”, e não “como se consegue”
- Na prática o Modelo Relacional é diferente da Álgebra Relacional
- O Modelo Relacional não é baseado em conjuntos (admite duplicados)

# Propriedade de fecho

- Os operandos da álgebra são as relações ou variáveis que representam relações, e os operadores permitem manipulações comuns nas relações da base de dados
- A álgebra relacional forma um “sistema fechado”: o resultado de um operador (sendo sempre uma tabela) pode constituir a entrada de outros operadores:
  - $\text{operador}_1(\dots(\text{operador}_n(\text{expressão}))\dots)$

# Operadores relacionais

- SELECT/RESTRICT: seleciona linhas de uma relação
- PROJECT: seleciona colunas de uma relação
- JOIN: junta duas relações com base em valores comuns de atributos comuns (na sua forma mais utilizada, mas há outras)


# SELECT: exemplo

- `select Vinho where Preço > 100`
- $\sigma_{Preço > 100}(Vinho)$

**Vinho**

MARCA	PREÇO	ANO	
A	20	1992	
C	50	1992	
D	120	1972	
E	200	1970	

*select também  
se chama **restrict***



MARCA	PREÇO	ANO	
D	120	1972	
E	200	1970	

# PROJECT: exemplo

- project *Vinho* over *Preço*, *Ano*
- $\pi_{\text{preço,ano}}(\textit{Vinho})$

**Vinho**

MARCA	PREÇO	ANO	
A	20	1992	
C	50	1992	
D	120	1972	
E	200	1970	



PREÇO	ANO
20	1992
50	1992
120	1972
200	1970

# JOIN: exemplo

- join *Vinho* and *Região* over *Zona*
- $Vinho \bowtie Região$

Vinho

MARCA	PREÇO	ANO	ZONA
A	20	1992	Z3
C	50	1992	Z1
D	120	1972	Z3
E	200	1970	Z2

Região

ZONA	NOME
Z1	Douro
Z2	V.Real



únicas linhas com valor  
comum de ZONA

MARCA	PREÇO	ANO	ZONA	NOME
C	50	1992	Z1	Douro
E	200	1970	Z2	V.Real

# Mais exemplos

- $\pi_{\text{preço,ano,zona.nome}}(\sigma_{\text{Preço} > 100}(\text{Vinho}) \bowtie \text{Região})$
- Qual o resultado desta consulta?



# JOIN: características

- É um operador importante pois permite fazer relacionamentos entre tabelas
- Pode ser uma operação custosa em tempo, tem de percorrer todas as tabelas envolvidas
- A sua utilização deve ser optimizada, pois é uma operação efectuada frequentemente

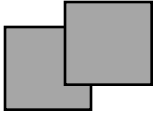
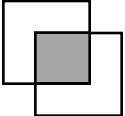
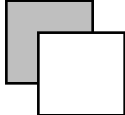
# JOIN: tipos

- A junção natural compara colunas com o mesmo nome (não existe em SQL)
- Usa-se a junção *teta*: Vinho  $\bowtie_{\text{marca}}$  Região
- Equi-junção: a função teta é =
- Junção  $\theta$ : INNER JOIN
- Junção externa (direita, esquerda, completa): LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN

# JOIN: casos particulares

- Se R e S tiverem o mesmo esquema, o resultado de  $R \bowtie S$  é a intersecção
- Se R e S não tiverem colunas comuns, a sua junção degenera num produto

# Operadores sobre conjuntos

- UNION 
- INTERSECT 
- MINUS 

Operam sobre relações que devem ser compatíveis:  
ter o mesmo número de atributos e estes serem do  
mesmo tipo e terem o mesmo nome

# Produto cartesiano

- Pouco utilizado na prática
- Dadas duas tabelas R e S o seu produto cartesiano é a combinação de todos os tuplos de R com todos os tuplos de S. A cardinalidade do resultado é o produto das cardinalidades de R e de S, e o grau é a soma dos seus graus (se houver colunas com nomes iguais, mudam-se os nomes)

# SELECT genérico

```
SELECT [ALL | DISTINCT] select_expr, ...  
  FROM table_references  
  [WHERE where_definition]  
  [GROUP BY {col_name | expr | position} [ASC | DESC]]  
  [HAVING where_definition]  
  [ORDER BY {col_name | expr | position} [ASC | DESC] , ...]  
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]  
  [FOR UPDATE | LOCK IN SHARE MODE]]
```

# GROUP BY

- Agrupamentos:
  - max, min
  - count
  - avg
  - sum
  - ....

# Resultado de um SELECT

- Lê todas as combinações possíveis de tuplos das tabelas no FROM
- Elimina as que não respeitam o WHERE
- Agrupa de acordo com GROUP BY
- Elimina grupos de acordo com HAVING