L.EIC
#2.1

Bases de Dados

Resumos

1. Diagrama UML
2. Modelo relacional
3. Teoria de Design Relacional
   - SQL: Data Definition Language
4. Álgebra relacional
   - SQL: Data Modification Language
5. Tópicos avançados (views, triggers e transações)

1. UML: Modelação de dados

Classe

Associação

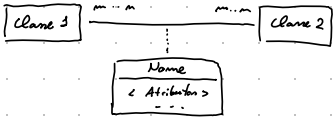
0..1 - 0..1 | Um para um

* - 0..1 | Muitas para um

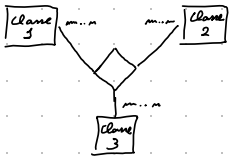
* - * | Muitas para muitas

- **Auto - associação**

- **Classe de associação**

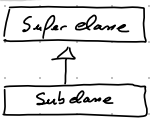


- **Associação "n-ary"**



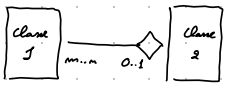
- **Associação qualificada**

Generalização



- **Propriedades:**
  - → Completa / Disjunta
  - → Exclusiva / Sobreposta

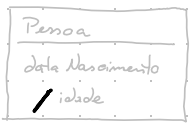
Agregação



Composição

- **Restrição**

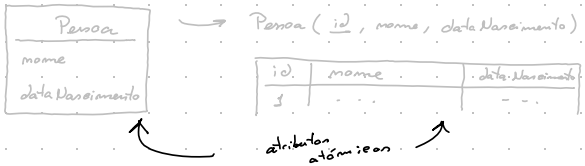
- **Elemento derivado**



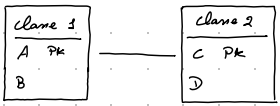
{ idade = "now()" - data Nascimento }

2. Modelo Relacional

Cada classe tem direito a uma tabela, com uma chave primária:



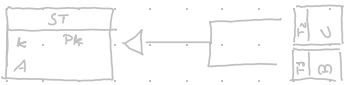
Associação



- 0..1 - 0..1 | Adicionar uma chave estrangeira (na tabela com menos tuplas)
- * - * | Nova tabela só com chaves estrangeiras na associação
- * - 0..1 | (A melhor das duas opções)

- **Auto - associação** | Nova tabela só com FKs
  | Adicionar FK (só em * - 0..1)
- **Classe de associação** | A tabela da classe contém as chaves estrangeiras
- **Associação "n - ary"** | Nova tabela só com FKs
- **Associação qualificada** | Nova tabela com FKs para o qualificador

Generalização



- **"E/R style"**

  ST (k , A)
  T1 (k → ST , B)
  T2 (k → ST , C)

  Uma relação por classe

  → Útil em generalizações sobrepostas com muitas subclasses

- **Orientado a objetos**

  ST (k , A) (se disjunto)
  T1 (k , A , B)
  T2 (k , A , C)

  Uma relação por cada subárvore

  → Útil em generalizações disjuntas com poucos atributos na superclasse

- **Uso de NULLs**

  ST (k , A , B , C)

  Uma tabela com tuplas nulas

  → Útil em generalizações sobrepostas com poucas subclasses

(*) Composições e agregações são tratadas como associações comuns.

(/) Elementos derivados são tratados como elementos comuns.

Restrições

- Not Null
- Unique
- Check (condições)
- Primary Key
- Foreign Key
- Default <valor>

3. Teoria de Design Relacional

Regras de Dependências Funcionais

- **Divisão / Combinação** | $\overline{A} \to \overline{B} \iff \overline{A} \to B_1 , \overline{A} \to B_2 , \dots$
- **Transitividade** | $\overline{A} \to \overline{B} \wedge \overline{B} \to \overline{C} \implies \overline{A} \to \overline{C}$
- **Dependências triviais** | $\overline{B} \subseteq \overline{A} \implies \overline{A} \to \overline{B}$

Fecho de atributos

1. Dividir as DFs de forma que só haja um atributo à direita
2. Procurar uma DF aplicável ao conjunto de atributos dado.
3. Adicionar o lado direito da DF ao conjunto de atributos dado.
4. Repetir **2.** e **3.** até não haver alterações

Ex:

$AB \to C$	$\{A,B\}$ → $AB \to C$
$BC \to AD$	$\{A,B,C\}$ → $BC \to AD$
$D \to E$	$\{A,B,C,D\}$ → $D \to E$
$CF \to B$	$\{A,B,C,D,E\}$
$\{A,B\}^+$?	

$\therefore \{A,B\}^+ = \{A,B,C,D,E\}$

- Algoritmo útil para **encontrar todas as superchaves**: fecho para todos os subconjuntos de atributos.

Projeção de DFs

Dado a relação R e uma projeção incompleta desta (R_1):

1. Calcular o fecho de todos os subconjuntos de R_1.
2. Apontar todas as DFs que incluam atributos de R_1.
3. Construir uma base mínima

Ex:

$R(A,B,C,D)$	$\{A\}^+ = \{A,B,C,D\}$ | $A \to CD$
$A \to B$	$\{C\}^+ = \{C,D\}$ | $C \to D$
$B \to C$	$\{D\}^+ = \{D\}$
$C \to D$	$\{A,C\}^+ = \{A,B,C,D\}$ | $AC \to D$
$R_1(A,C,D)$	$\{A,D\}^+ = \{A,B,C,D\}$ | $AD \to C$
DFs de R_1 ?	$\{C,D\}^+ = \{C,D\}$

$\therefore$ Base mínima: $A \to C$, $C \to D$

Formas Normais

1 NF	Cada atributo é atómico (exclui listas, compostos, ...)
2 NF	Atributos não-chave não podem ser dependentes de um subconjunto da chave.
3 NF	Em todas as DFs $X \to Y$: X é uma superchave ou Y é primo
BC NF	Em todas as DFs, o lado esquerdo é uma superchave
4 NF	...
5 NF	

Decomposição BCNF

| Assegura junções sem perdas
| Não garante a preservação das dependências

1. Verificar que a relação não se encontra na BCNF. Ignorar caso contrário.
2. Para cada DF $X \to Y$ que viola a BCNF, decompor duas relações com este algoritmo
   - → $R_1 = X^+$
   - → $R_2 = (X, <\text{atributos fora de } X^+>)$
3. Unir todas as relações

Ex: $R(\underline{A},\underline{B},C,D,E)$

$AB \to CD$

$D \to E$ ⚠

$A \to C$ ⚠

$B \to D$ ⚠

⚠ viola BCNF

$R_3(\underline{D},E)\ \{D \to E\}$

$R_2(D,\underline{A},\underline{B},C)\ \{AB \to CD, A \to C, B \to D\}$ ⚠

$R_3(\underline{A},C)\ \{A \to C\}$

$R_4(\underline{A},\underline{B},D)\ \{B \to D, AB \to D\}$ ⚠

$R_5(\underline{B},D)\ \{B \to D\}$

$R_6(B,A)\ \{\}$

$\therefore R_3 \cup R_3 \cup R_5 \cup R_6$

Decomposição 3NF

| Assegura junção sem perdas e preservação de dependências

1. Encontrar uma base mínimal para as DF da relação.
2. Para cada DF $X \to A$, cria uma relação $R(X, A)$.
3. Se nenhuma das relações é uma super chave, é necessário adicionar uma outra relação com a chave.

Ex: $R(A,B,C,D,E)$

$AB \to C$

$C \to B$

$A \to D$

Chaves de R: $\{A,B,E\}, \{A,C,E\}$

$R_1(A,B,C)$ $R_2(C,B)$

$R_3(A,D)$ $R_4(A,B,E)$

"Chase test"

Para junções sem perdas

1. Construir a tabela.
   - → Uma linha por relação.
   - → Uma coluna por atributo.
   - → Letras com o nº da coluna em baixo são atributos que não estão incluídos nessa relação.
2. Juntar letras em comum a partir das DFs.
3. É uma junção sem perdas se a tabela final tiver uma linha "sem números".

$S(A,B,C,D)$ $\{A \to B, B \to C, CD \to A\}$

↳ $S_1(A,D)$ $S_2(A,C)$ $S_3(B,C,D)$

A	B	C	D
a	b_1	c_1	d
a	b_2	e	d_2
a_3	b	e	d

$A \to B$ | $b_1 = b_2$

$B \to C$ | $c_1 = e$

$CD \to A$ | $a = a_3$

A	B	C	D
a	b_1	e	d
a	b	e	d_2
a	b	e	d

4. Álgebra Relacional

Seleção ■
$\sigma_{condição}$ Relação

Filtrar as linhas que não cumprem com a condição.

Projeção ■
$\pi_{colunas}$ Relação

Mostra apenas determinadas colunas da relação.

Produto cartesiano ■
Relação × Relação

Um tuplo para cada combinação de tuplas das duas relações.

Junção natural
Relação ⋈ Relação

Produto cartesiano em que é enforçado a igualdade de todos os atributos com o mesmo nome.

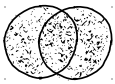
Junção "Theta"
Relação $\bowtie_{condição}$ Relação

Produto cartesiano em que apenas as tuplas que cumpram a condição é que são selecionadas.

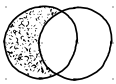
Semi-Junção
Relação ⋉ Relação

Tuplas da esquerda em que tem um tuplo da direita igual mas seus atributos de mesmo nome.

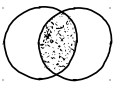
União ■
Relação ∪ Relação (mesmo esquema)



Diferença ■
Relação − Relação



Interseção
Relação ∩ Relação (mesmo esquema)



Divisão
Relação / Relação

Oposto do produto cartesiano.

Renomear ■
$\rho_{novo\_nome(novas\_colunas)}$ Relação

Renomeia nomes de atributos e relações para remover ambiguidade.

Algoritmo de divisão: R / S

1. Reordenar colunas de R de forma que as últimas sejam as de S.
2. Ordenar tuplos de R
3. Cada sub-tuplo é parte do resultado se as últimas colunas contierem S.

■ Operadores essenciais: todos os outros podem ser deduzidos a partir destes.

Em Álgebra Relacional, os duplicados são eliminados (contrariamente a SQLite)

- **Funções de agregação:**
  - cnt ()
  - sum ()
  - avg ()
  - max ()
  - min ()

5. Tópicos Avançados

Views | Tabela resultante de uma "query" sem valores armazenados.

- Também pode ser usada em queries com outras relações
- Útil para esconder dados de utilizadores (N/A em SQLite)

- **Criar**
  CREATE VIEW <nome> (<nomeAtributos>) AS <query>
- **Remover**
  DROP VIEW <nome>

Triggers

(N/A em SQ Lite)

CREATE TRIGGER <nome>
BEFORE | AFTER | INSTEAD OF <evento> ⟶ insert / delete / update
REFERENCING <variáveis> ⟶ old | new row | table
[FOR EACH ROW]
[WHEN <condição>]
<ação> ⟶ Dentro de um "Begin-End bracket"

Ex:

```
CREATE TRIGGER R
BEFORE INSERT ON College
FOR EACH ROW
WHEN exists (...)
BEGIN
   SELECT raise (ignore);
END;
```

Indexes | "IDs" automáticos em SQLite para cada tabela.

- Acelera imenso pesquisas.
- Custo de maior espaço gasto.
- Não é usado pelas utilizadores: é uma ferramenta interna do motor de queries.

- **Sintaxes SQL** (N/A em SQ Lite)
  CREATE [UNIQUE] INDEX <nome> ON T(A) / T(A, A, ...)
  DROP INDEX <nome>

Transactions | Sequência de operações executadas atomicamente

- **Comandos**
  START TRANSACTION [READ ONLY | WRITE]
  COMMIT
  ROLL BACK
- **Propriedades ACID**
  Atomicidade
  Consistência
  Isolação
  Durabilidade
- **Níveis de isolamento**

	Dirty reads	Non-repeatable reads	Phantom tuples
Read Uncommited	✓	✓	✓
Read Commited	×	✓	✓
Repeatable Read	×	×	✓
Serializable	×	×	×

↓ maior consistência ≡ menor performance