

Bases de Dados



Universidade do Porto

Faculdade de Engenharia

FEUP

1

Diagrama de Classes UML

Observação: adaptado de slides desenvolvidos pelos Profs. da FEUP :Ademar Aguiar, Gabriel David e João Pascoal Faria.

UML

2

Índice

- O que é a UML?
- Modelos e diagramas
- Diagrama de classes
 - Objectivos
 - Objectos
 - Classes
 - Atributos
 - Operações
 - Associações
- Diagrama de classes
 - Agregações
 - Composições
 - Generalizações
 - Restrições
 - Elementos derivados
- Mapeamento para o Modelo Relacional

UML

3

O que é a UML?

- UML = *Unified Modeling Language*
- UML é uma linguagem (notação com semântica associada) para
 - visualizar
 - especificar
 - construir
 - documentaros artefactos de um sistema com uma componente intensiva de software (*software intensive system*)
- UML não é uma metodologia
 - não diz quem deve fazer o quê, quando e como
 - UML pode ser usado segundo diferentes metodologias, tais como RUP (*Rational Unified Process*), FDD (*Feature Driven Development*), etc.
- UML não é uma linguagem de programação

UML

4

Modelos e Diagramas

- Um modelo é uma representação em **pequena escala**, numa perspectiva particular, de um sistema existente ou a criar
 - Atitude de **abstracção** (omissão de detalhes) fundamental na construção de um modelo
 - Modelos são a linguagem por excelência do projectista (*designer*)
 - Modelos são veículos para comunicação com vários interessados (*stakeholders*)
 - Modelos permitem raciocinar acerca do sistema real, sem o chegar a construir
- Ao longo do ciclo de vida de um sistema são construídos vários modelos, sucessivamente refinados e enriquecidos

UML

5

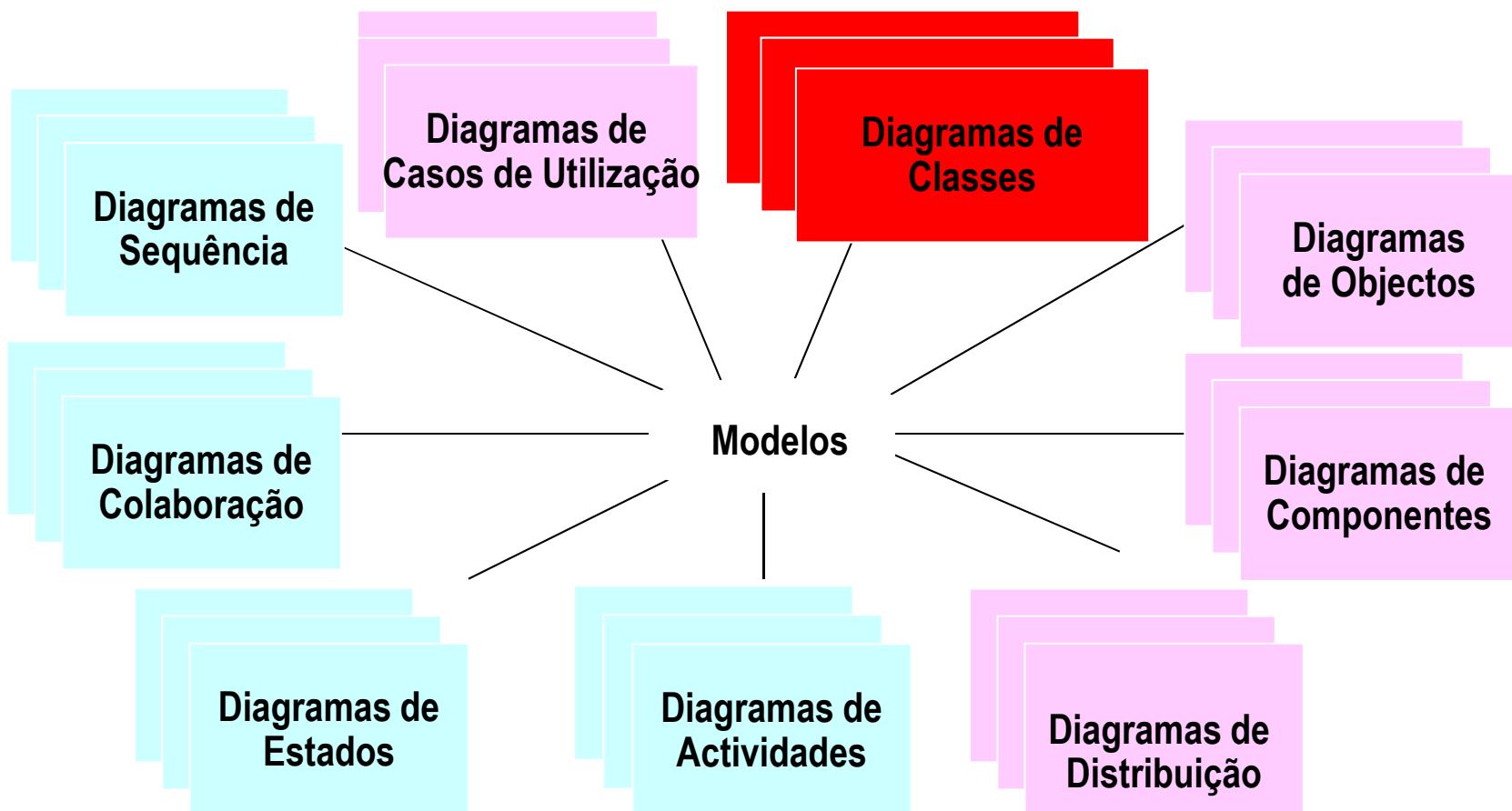
Modelos e Diagramas

- Um modelo é constituído por um conjunto de **diagramas** (desenhos) consistentes entre si, acompanhados de descrições textuais dos **elementos** que aparecem nos vários diagramas
 - Um **diagrama** é uma vista sobre um modelo
 - O mesmo **elemento** (exemplo: classe) pode aparecer em vários diagramas de um modelo
- No UML, há nove diagramas *standard*
 - Diagramas de visão estática: casos de utilização (*use case*), classes, objectos, componentes, distribuição (*deployment*)
 - Diagramas de visão dinâmica: sequência, colaboração, estados (*statechart*), actividades

UML

6

Modelos e Diagramas



UML: Diagrama de Classes

7

Objectivos

- Um diagrama de classes serve para modelar o vocabulário de um sistema, do ponto de vista do utilizador/problema ou do implementador/solução
 - Ponto de vista do utilizador/problema – na fase de captura e análise de requisitos, em paralelo com a identificação dos casos de utilização
 - Vocabulário do implementador/solução – na fase de projecto (*design*)
- Construído e refinado ao longo das várias fases do desenvolvimento do software, por analistas, projectistas (*designers*) e implementadores
- Também serve para:
 - Especificar colaborações (no âmbito de um caso de utilização ou mecanismo)
 - **Especificando esquemas lógicos de bases de dados**
 - Especificar vistas (estrutura de dados de formulários, relatórios, etc.)
- Modelos de objectos de domínio, negócio, análise e *design*

UML: Diagrama de Classes

8

Objectos

Um **objecto** é:

- algo com fronteiras bem definidas,
- relevante para o problema em causa,
- com estado,
 - modelado por valores de atributos (tamanho, forma, peso, etc.) e por ligações que num dado momento tem com outros objectos
- com comportamento
 - um objecto exibe comportamentos invocáveis (por resposta a chamadas de operações) ou reactivos (por resposta a eventos)

UML: Diagrama de Classes

9

Objectos

- e identidade
 - no espaço: é possível distinguir dois objectos mesmo que tenham o mesmo estado
 - exemplo: podemos distinguir duas folhas de papel A4, mesmo que tenham os mesmos valores dos atributos
 - no tempo: é possível saber que se trata do mesmo objecto mesmo que o seu estado mude
 - exemplo: se pintarmos um folha de papel A4 de amarelo, continua a ser a mesma folha de papel

UML: Diagrama de Classes

10

Objectos

Objectos do mundo real e objectos computacionais:

- No desenvolvimento de software orientado por objectos, procura-se imitar no computador o mundo real visto como um conjunto de objectos que interagem entre si
- Muitos objectos computacionais são imagens de objectos do mundo real
- Dependendo do contexto (análise ou projecto) podemos estar a falar em objectos do mundo real, em objectos computacionais ou nas duas coisas em simultâneo
- Exemplos de objectos do mundo real:
 - o Sr. João
 - a aula de ES no dia 11/10/2000 às 11 horas
- Exemplos de objectos computacionais:
 - o registo que descreve o Sr. João (imagem de objecto do mundo real)
 - uma árvore de pesquisa binária (objecto puramente computacional)

UML: Diagrama de Classes

11

Classes

- No desenvolvimento de software OO, não nos interessam tanto os objectos individuais mas sim as classes de objectos
- Uma **classe** é um descritor de um conjunto de objectos que partilham as mesmas propriedades (semântica, atributos, operações e relações)
 - Trata-se de uma noção de classe em **compreensão**, no sentido de **tipo de objecto**, por oposição a uma noção de classe em **extensão**, como conjunto de objectos do mesmo tipo
- Um objecto de uma classe é uma **instância** da classe
- A **extensão** de uma classe é o conjunto de instâncias da classe
- Em Matemática, uma classe é um conjunto de “objectos” com uma propriedade em comum, podendo ser definida indiferentemente em compreensão ou em extensão

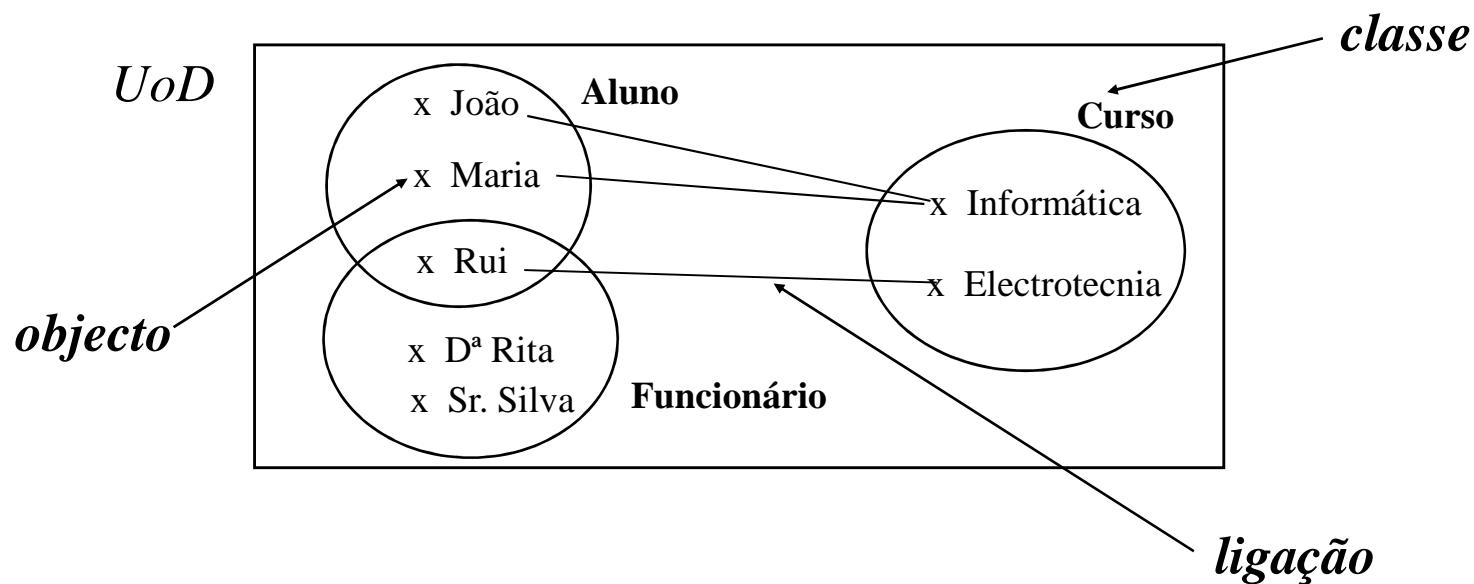
$$C = \{x \in \mathbb{N} : x \bmod 3 = 2\} = \{2, 5, 8, 11, 14, \dots\}$$

UML: Diagrama de Classes

12

Classes

- O conjunto de todos os objectos num determinado contexto forma um universo (*UoD - Universe of Discourse*)
- As extensões das classes são subconjuntos desse universo



UML: Diagrama de Classes

13

Classes

- Classes podem representar:
 - **Coisas concretas:** Pessoa, Turma, Carro, Imóvel, Factura, Livro
 - **Papéis que coisas concretas assumem:** Aluno, Professor, Piloto
 - **Eventos:** Curso, Aula, Acidente
 - **Tipos de dados:** Data, Intervalo de Tempo, Número Complexo, Vector
- Decomposição orientada por objectos : começa por identificar os tipos de objectos (classes) presentes num sistema
 - contrapor com decomposição funcional ou algorítmica
 - tipos de objectos são mais estáveis do que as funções, logo a decomposição orientada por objectos leva a arquitecturas mais estáveis

UML: Diagrama de Classes

14

Classes

- Em UML, uma classe é representada por um rectângulo com o nome da classe



- Habitualmente escreve-se o nome da classe no singular (nome de uma instância), com a 1^a letra em maiúscula
- Para se precisar o significado pretendido para uma classe, deve-se explicar o que é (e não é ...) uma instância da classe
 - Exemplo: “Um aluno é uma pessoa que está inscrita num curso ministrado numa escola. Uma pessoa que esteve no passado inscrita num curso, mas não está presentemente inscrita em nenhum curso, não é um aluno.”
 - Em geral, o nome da classe não é suficiente para se compreender o significado da classe

UML: Diagrama de Classes

15

Atributos

Atributos de instância:

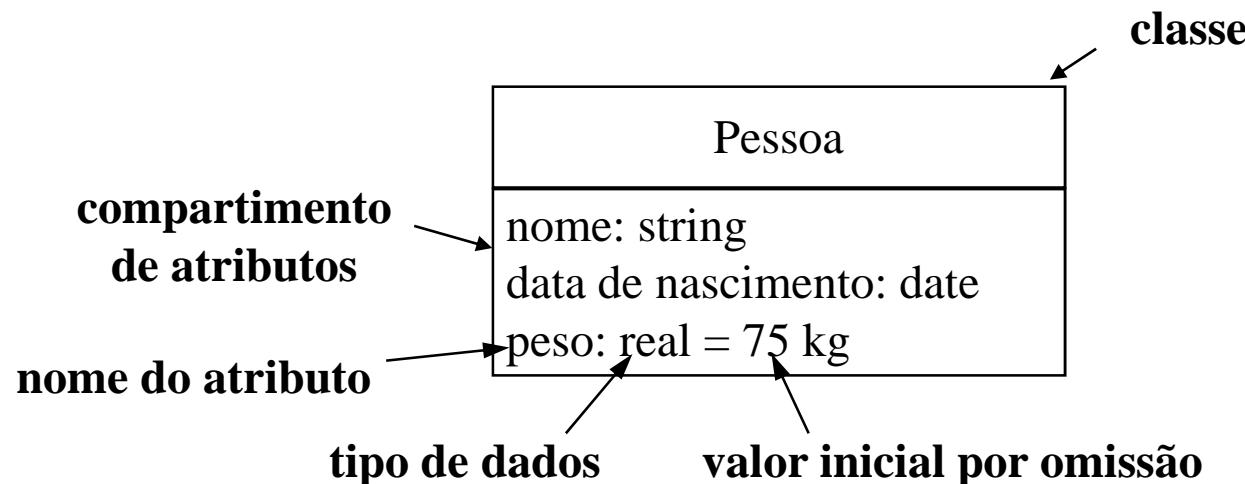
- O estado de um objecto é dados por valores de atributos (e por ligações que tem com outros objectos)
- Todos os objectos de uma classe são caracterizados pelos mesmos atributos (ou variáveis de instância)
 - o mesmo atributo pode ter valores diferentes de objecto para objecto
- **Atributos** são definidos ao nível da classe, enquanto que os **valores dos atributos** são definidos ao nível do objecto
- Exemplos:
 - Uma pessoa (classe) tem os atributos nome, data de nascimento e peso
 - João (objecto) é uma pessoa com nome “João Silva”, data de nascimento “18/3/1973” e peso “68 Kg”

UML: Diagrama de Classes

16

Atributos

- Atributos são listados num compartimento de atributos (opcional) a seguir ao compartimento com o nome da classe
- Uma classe não deve ter dois atributos com o mesmo nome
- Os nomes dos tipos não estão pré-definidos em UML, podendo-se usar os da linguagem de implementação alvo



UML: Diagrama de Classes

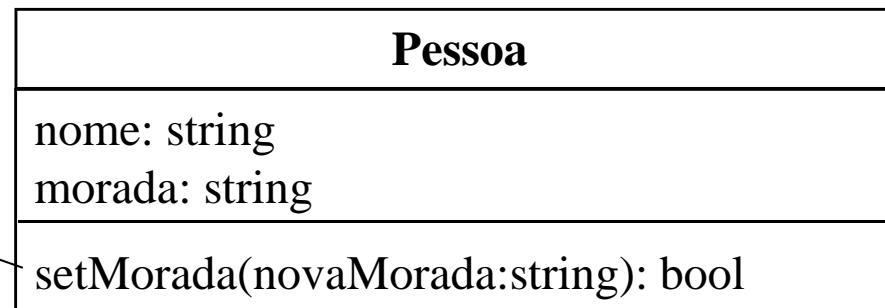
17

Operações

Operações de instância:

- Comportamento invocável de objectos é modelado por operações
 - uma operação é algo que se pode pedir para fazer a um objecto de uma classe
- Objectos da mesma classe têm as mesmas operações
- Operações são definidas ao nível da classe, enquanto que a invocação de uma operação é definida ao nível do objecto
- Princípio do encapsulamento: acesso e alteração do estado interno do objecto (valores de atributos e ligações) controlado por operações
- Nas classes que representam objectos do mundo real é mais comum definir responsabilidades em vez de operações

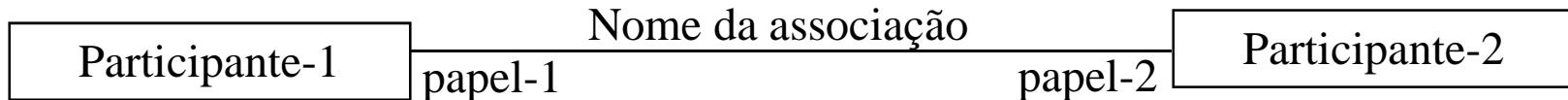
compartimento
de operações



UML: Diagrama de Classes

18

Associações



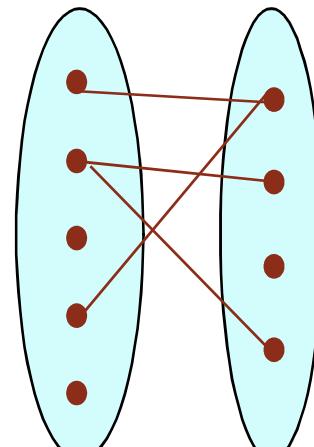
- Uma **associação** é uma relação entre objectos das classes participantes (um objecto de cada classe em cada ligação)
- Não gera novos objectos
- Matematicamente, uma associação binária é uma relação binária, i.e., um subconjunto do produto cartesiano das extensões das classes participantes
- Assim como um objecto é uma instância duma classe, uma **ligação** é uma instância de uma associação
- Pode haver mais do que uma associação (com nomes diferentes) entre o mesmo par de classes
- Papéis nos extremos da associação podem ter indicação de visibilidade (pública, privada, etc.)

UML: Diagrama de Classes

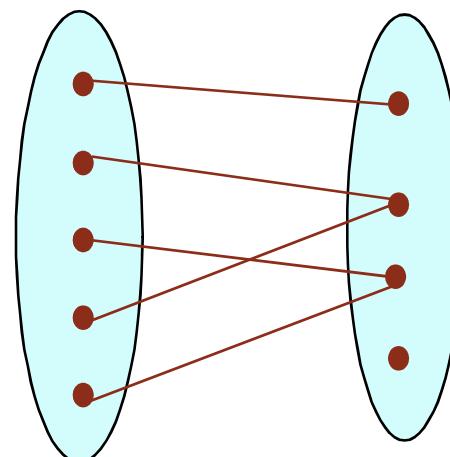
19

Multiplicidade das associações

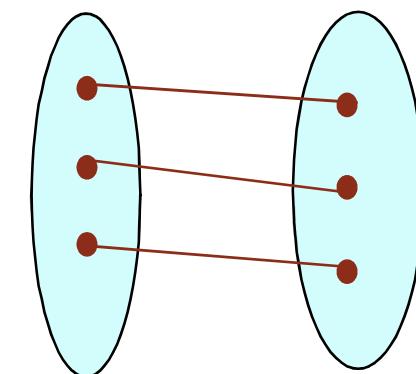
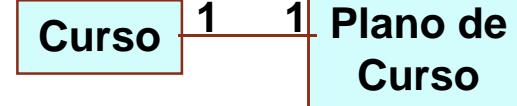
Muitos-para-Muitos



Muitos-para-1



1-para-1



(sem restrições)

UML: Diagrama de Classes

20

Multiplicidade das associações

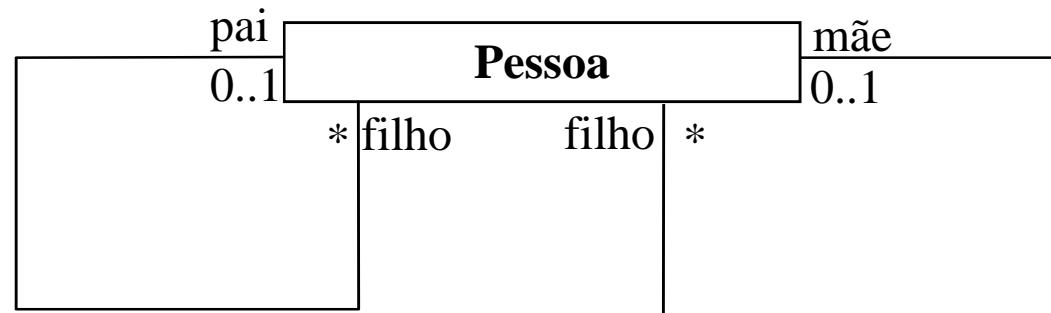
- | | |
|---------|-------------------------|
| 1 | - exatamente um |
| 0..1 | - zero ou um (zero a 1) |
| * | - zero ou mais |
| 0..* | - zero ou mais |
| 1..* | - um ou mais |
| 1, 3..5 | - um ou três a 5 |

UML: Diagrama de Classes

21

Associações reflexivas

- Pode-se associar uma classe com ela própria (em papéis diferentes)



UML: Diagrama de Classes

22

Nomes de associações

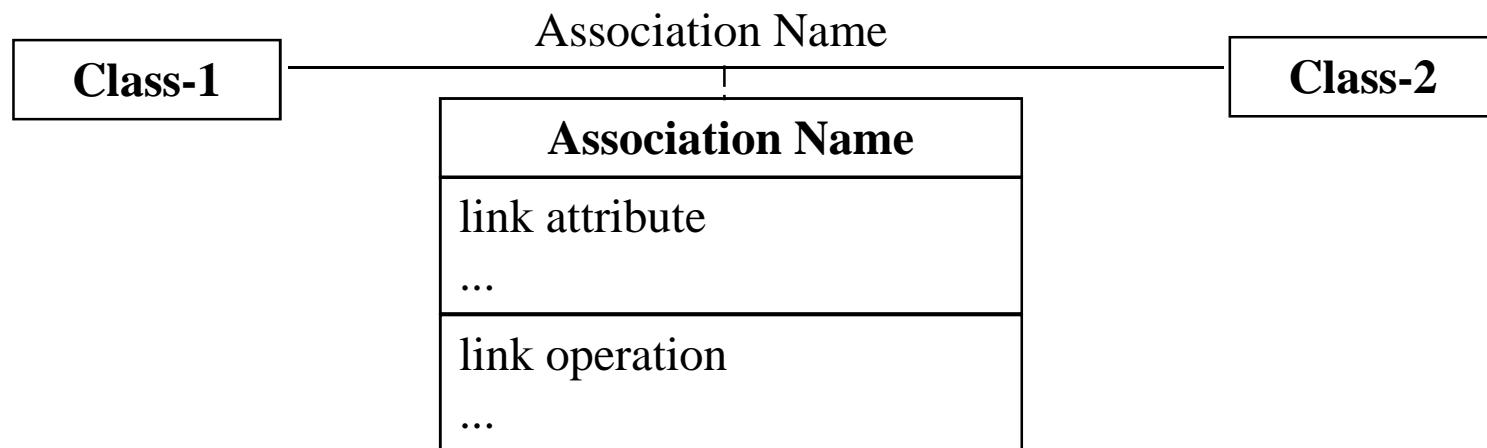
- A indicação do nome é opcional
- O nome é indicado no meio da linha que une as classes participantes
- Pode-se indicar o sentido em que se lê o nome da associação



UML: Diagrama de Classes

23

Classes de associação



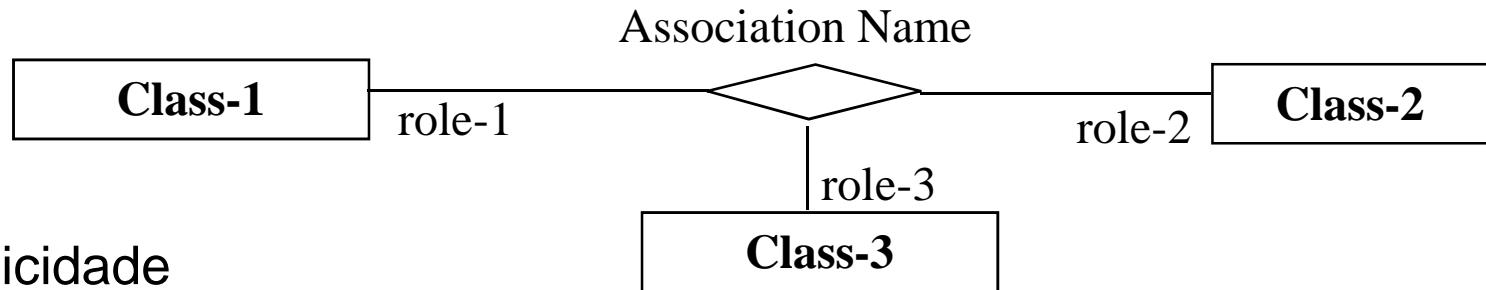
- reúne as propriedades de associação e classe
- o nome pode ser colocado num sítio ou outro, conforme interessa realçar a natureza de associação ou de classe, mas a semântica é a mesma
- o nome também pode ser colocado nos dois sítios
- não é possível repetir combinações de objectos das classes participantes na associação

UML: Diagrama de Classes

24

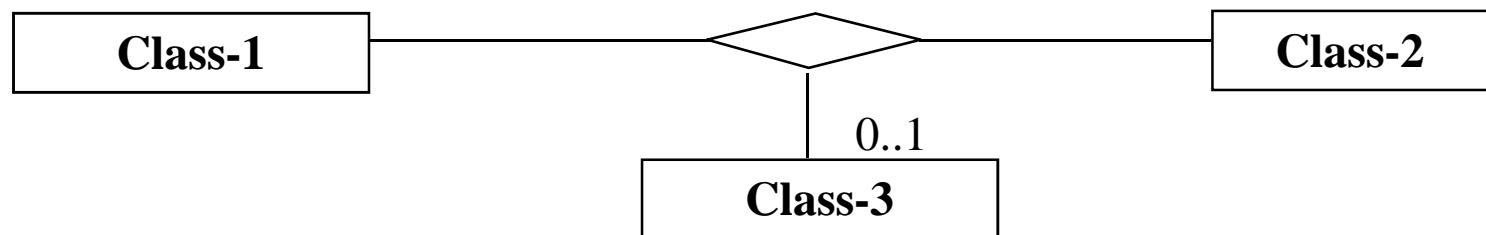
Associações n-árias

- Notação



- Multiplicidade

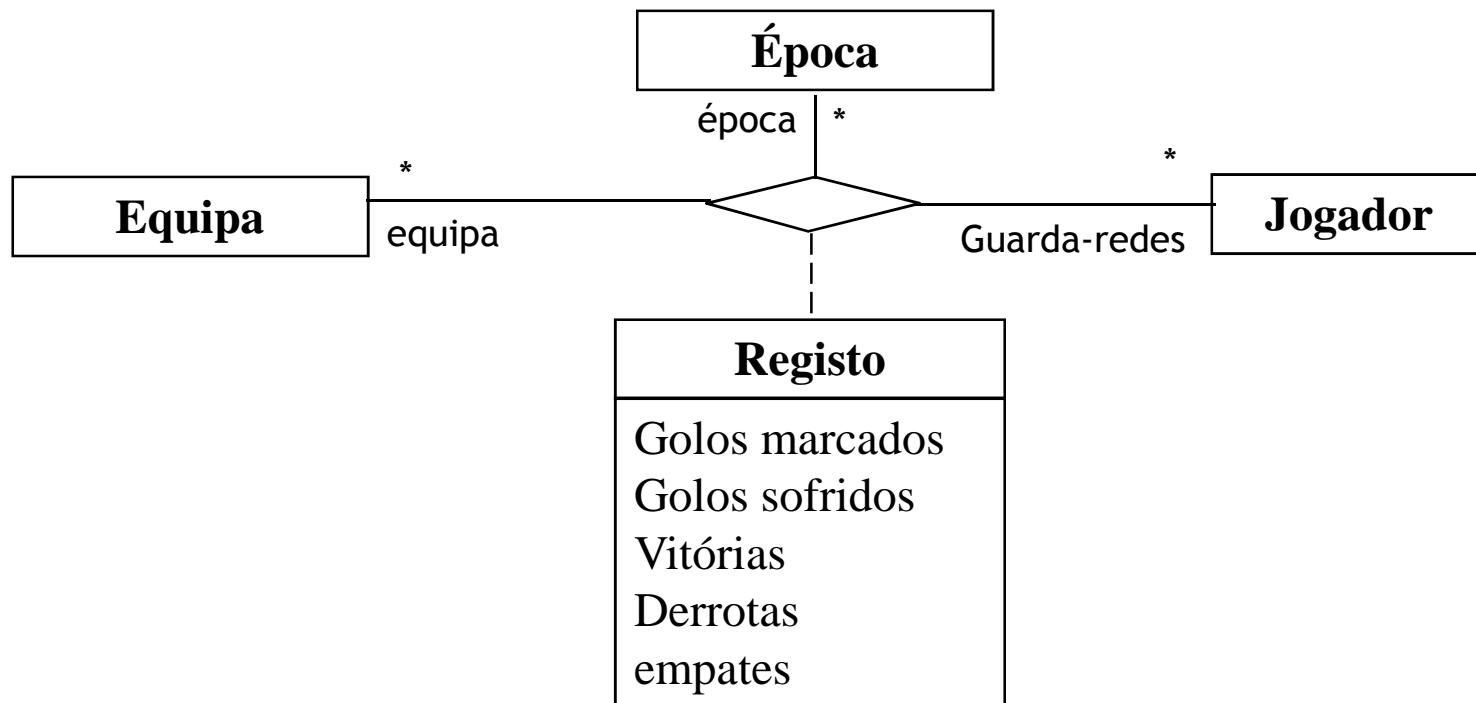
a cada par de objectos das restantes classes (1 e 2), correspondem 0 ou 1 objectos da classe 3



UML: Diagrama de Classes

25

Associações n-árias

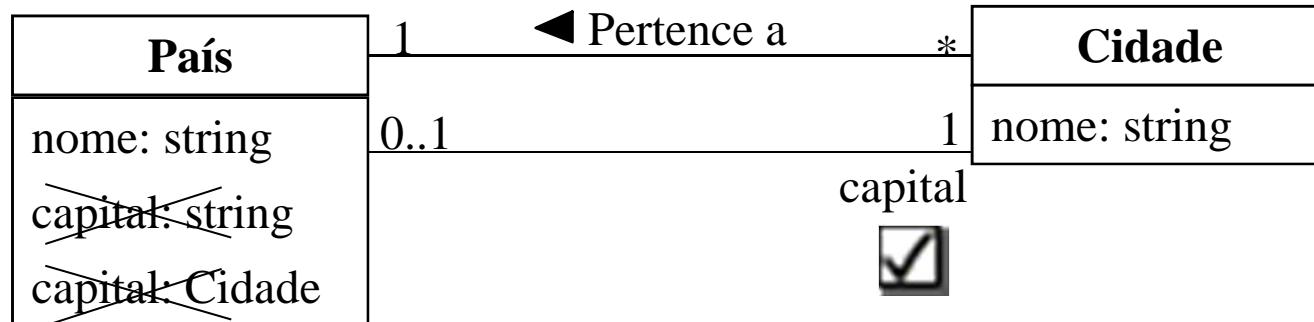


UML: Diagrama de Classes

26

Associação vs. atributo

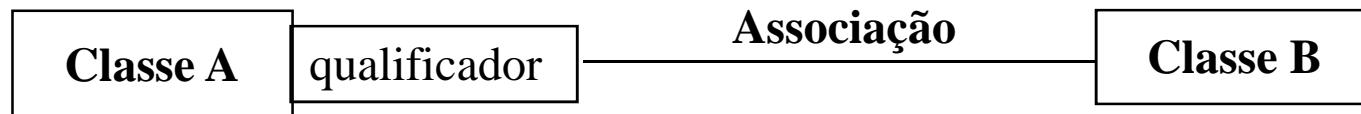
- Uma propriedade que designa um objecto de uma classe presente no modelo, deve ser modelada como uma associação e não como um atributo
- Exemplo:



UML: Diagrama de Classes

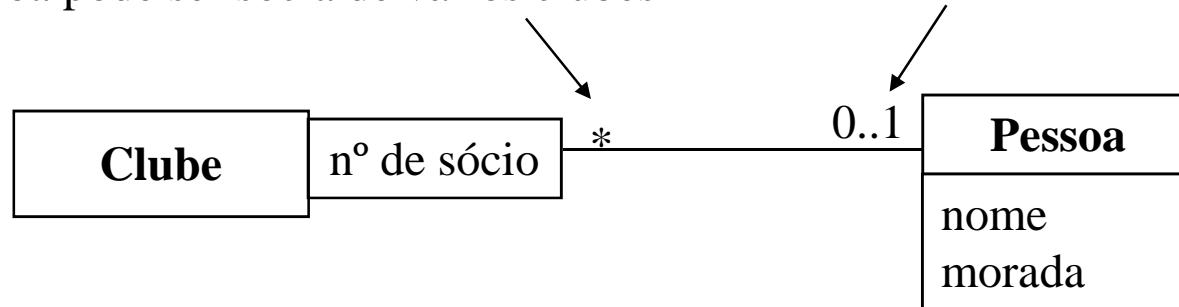
27

Associações qualificadas



- Qualificador: lista de um ou mais atributos de uma associação utilizados para navegar de A para B
- "Chave de acesso" a B (acesso a um objecto ou conjunto de objectos) a partir de um objecto de A

uma Pessoa pode ser sócia de vários clubes

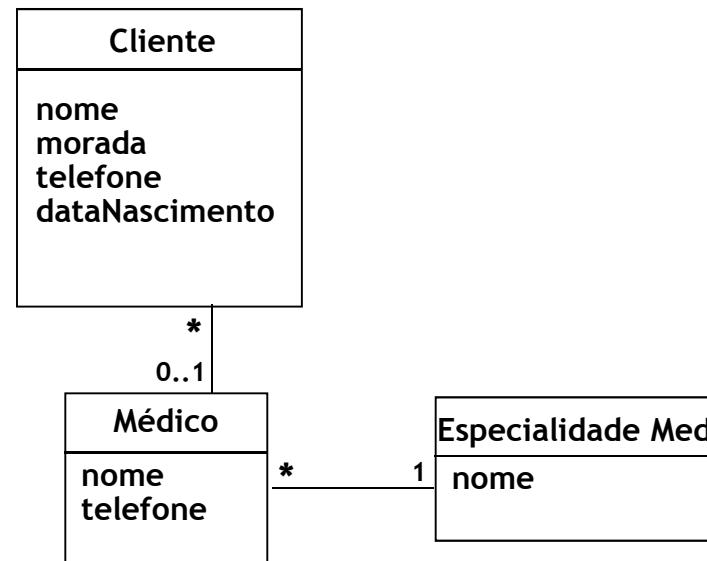


Exercício

28

O gabinete de psicologia HiperEgo pretende contratar o desenvolvimento de um sistema de informação que permita organizar a informação sobre os seus clientes e técnicos.

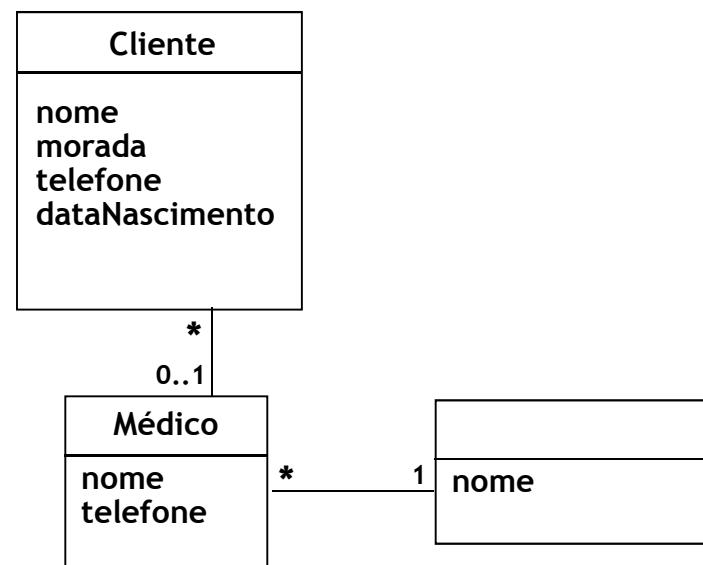
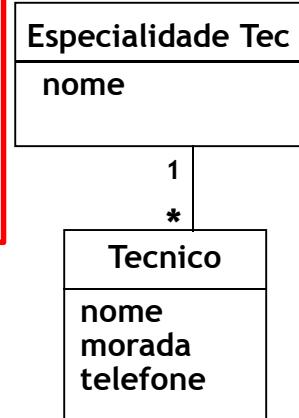
Sobre os clientes é necessário saber o nome, a morada, o telefone e a data de nascimento. É importante ainda saber se o apoio prestado pelo gabinete foi ou não receitado por um médico e, no caso afirmativo, é necessário saber o nome do médico, a sua especialidade e o número de telefone.



Exercício

29

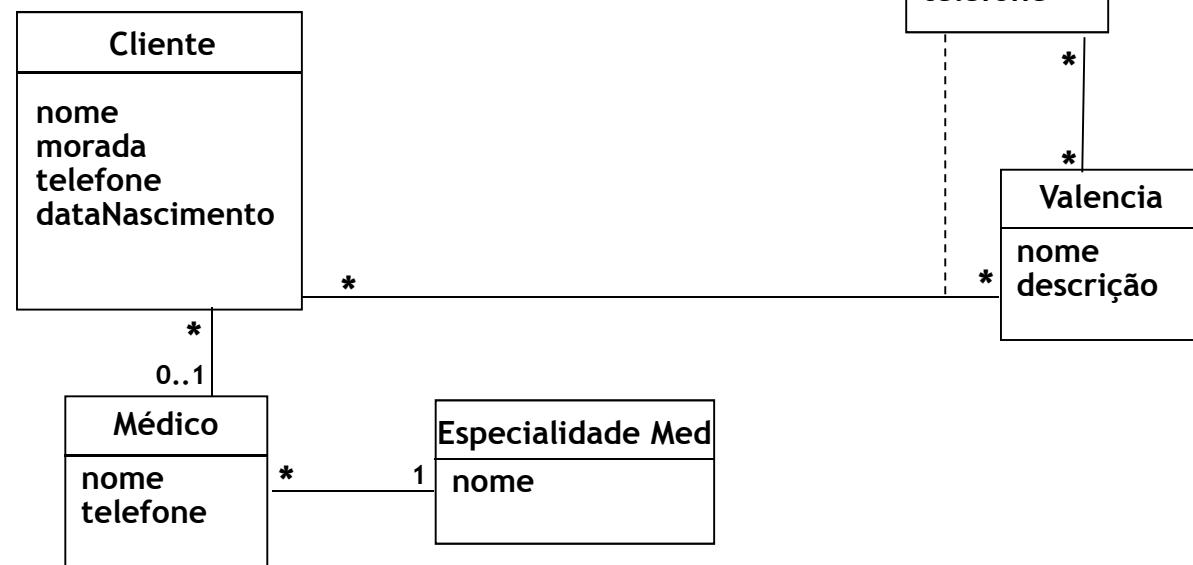
O gabinete conta com serviços de diferentes técnicos. Sobre cada um é necessário saber o nome, a morada, o telefone e a especialidade (psicólogo, terapeuta, professor especializado, etc.).



Exercício

30

O gabinete abrange diferentes valências (consulta psicológica, apoio psicopedagógico, orientação vocacional, terapia da fala, etc.), que podem ser realizados por diferentes técnicos. Cada valência é caracterizada por um nome e uma descrição. Os clientes podem ser acompanhados em diferentes valências, sendo nomeado um técnico responsável para cada cliente em cada valência.

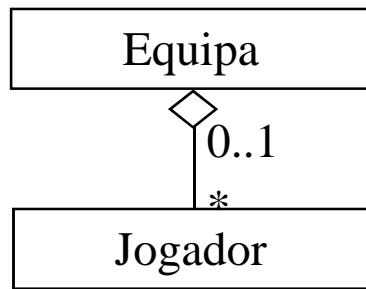


UML: Diagrama de Classes

31

Agregações

- Associação com o significado contém (é constituído por) / faz parte de (*part of*)
- Relação de inclusão nas instâncias das classes
- Hierarquias de objectos
- Exemplo:



- Uma equipa **contém** 0 ou mais jogadores
- Um jogador **faz parte de** uma equipa (num dado momento), mas também pode estar desempregado

UML: Diagrama de Classes

32

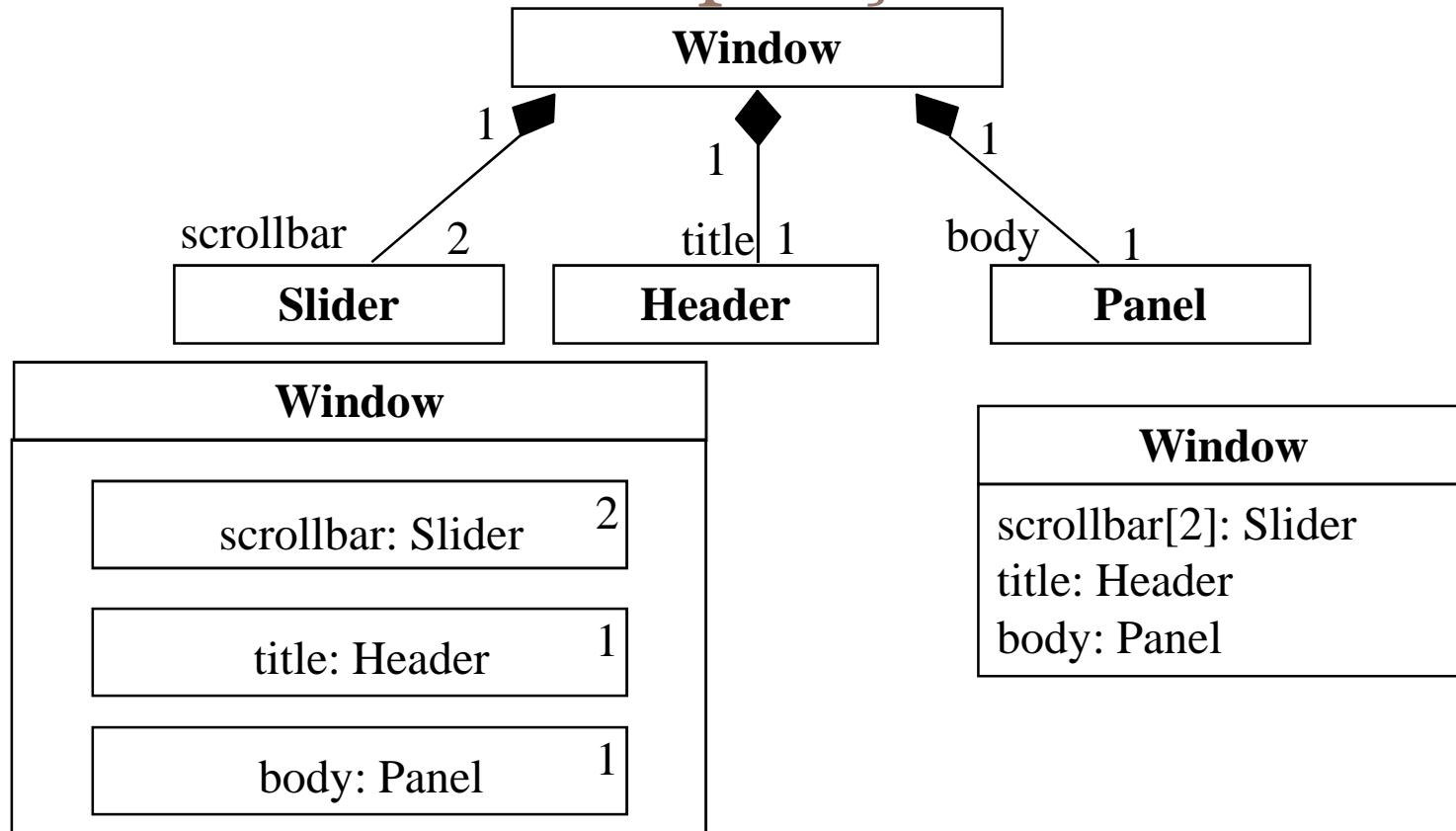
Composições

- Forma mais forte de agregação aplicável quando:
 - existe um forte grau de pertença das partes ao todo
 - cada parte só pode fazer parte de um todo (i.e., a multiplicidade do lado do todo não excede 1)
 - o topo e as partes têm tempo de vida coincidente, ou, pelo menos, as partes nascem e morrem dentro de um todo
 - a eliminação do todo propaga-se para as partes, em cascata
- Notação: losango cheio (◆) ou notação encaixada
- Membros-objecto em C++

UML: Diagrama de Classes

33

Composições

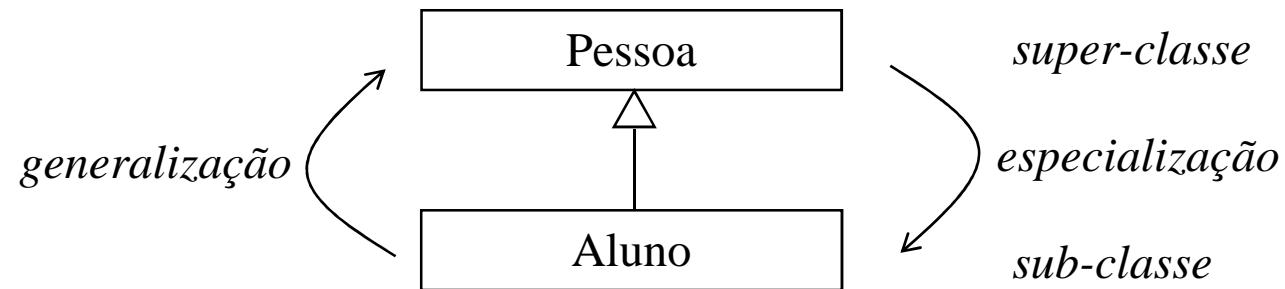


(sub-objectos no compartimento dos atributos)

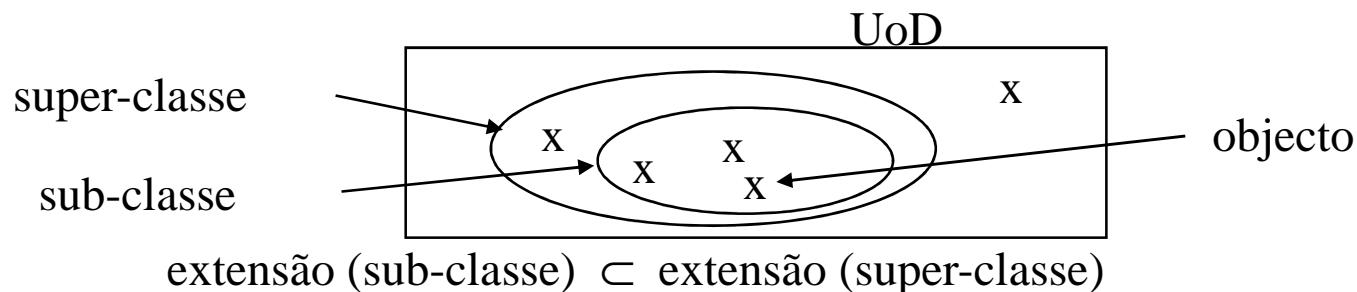
UML: Diagrama de Classes

34

Generalizações



- Relação semântica “is a” (“é um” / “é uma”) : um aluno é uma pessoa
- Relação de inclusão nas extensões das classes:



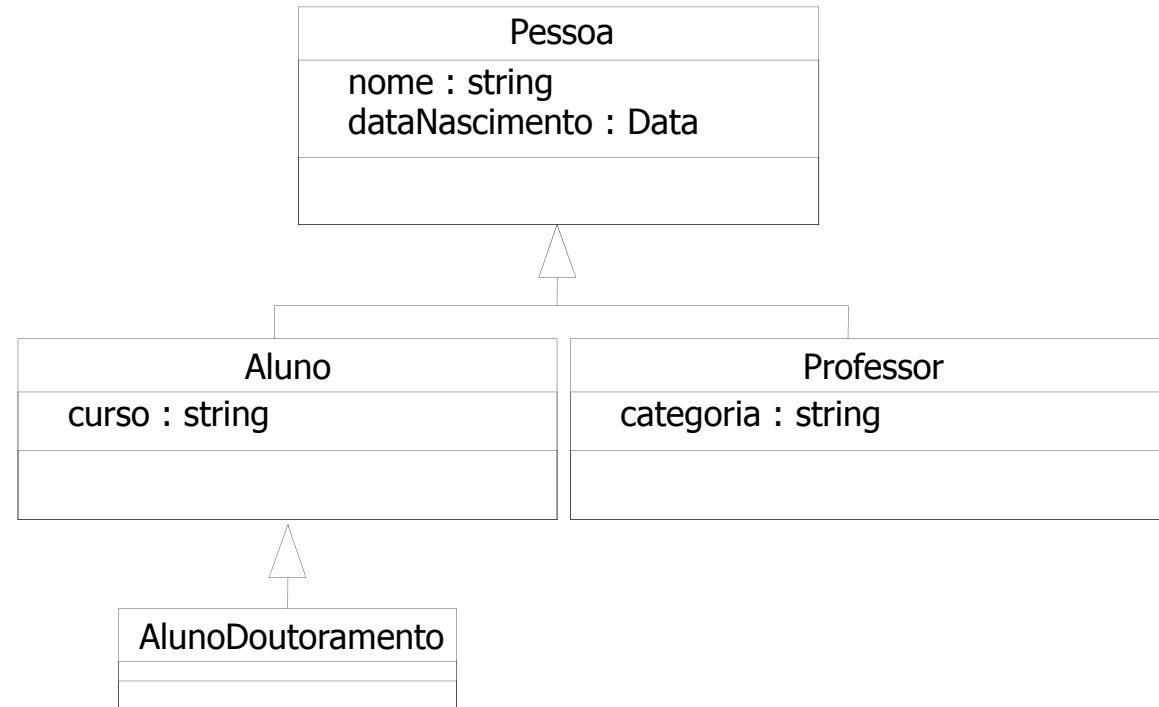
- Relação de herança nas propriedades: A sub-classe herda as propriedades (atributos, operações e relações) da super-classe, podendo acrescentar outras

UML: Diagrama de Classes

35

Generalizações: hierarquias de classes

- Em geral, pode-se ter uma hierarquia de classes relacionadas por herança / generalização
 - em cada classe da hierarquia colocam-se as propriedades que são comuns a todas as suas subclasses
- ⇒ **evita-se redundância, promove-se reutilização!**



UML: Diagrama de Classes

36

Generalizações: hierarquias de classes

Notações alternativas:

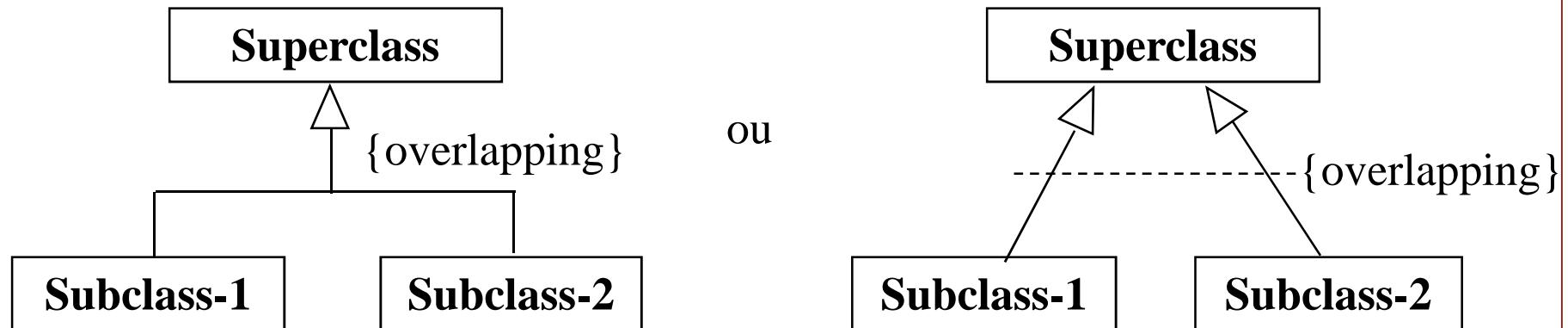


UML: Diagrama de Classes

37

Generalizações: subclasses sobrepostas

- caso em que um objecto da superclasse pode pertencer simultaneamente a mais do que uma subclasses
- indicado por restrição {overlapping}
- o contrário é {disjoint} (situação por omissão?)

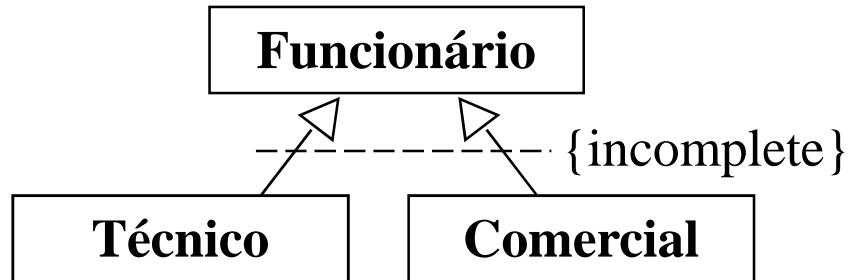


UML: Diagrama de Classes

38

Generalizações: subclasses incompletas

- caso em que um objecto da superclasse pode não pertencer a nenhuma das subclasses
- indicado por restrição {incomplete}
- o contrário é {complete} (situação por omissão?)

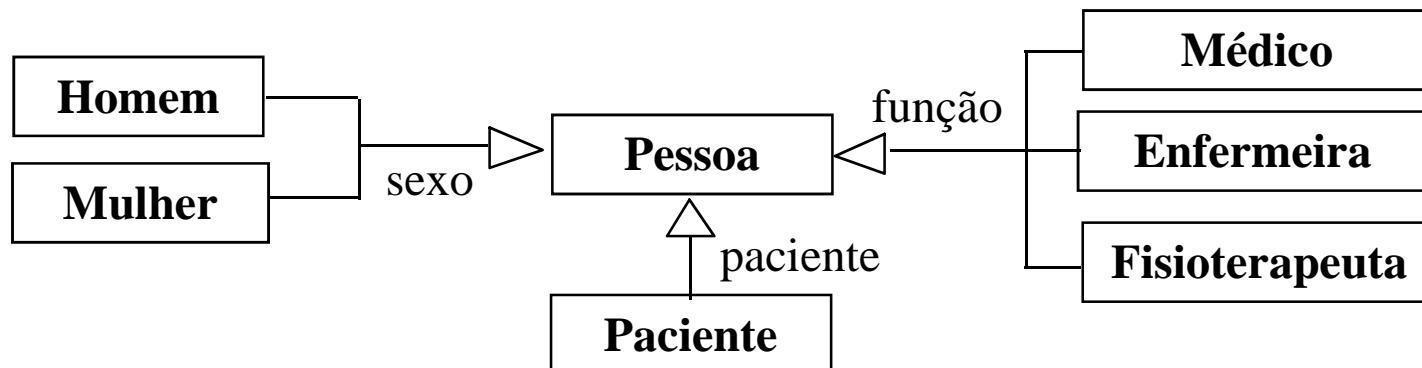


UML: Diagrama de Classes

39

Generalizações: classificação múltipla

- caso em que um objecto pode pertencer num dado momento a várias classes, sem que exista uma subclasse que represente a intersecção dessas classes (com herança múltipla)
- geralmente não suportado pelas linguagens de programação OO (pode então ser simulada por agregação de papéis)



exemplo de combinação legal: {Mulher, Paciente, Enfermeira}

UML: Diagrama de Classes

40

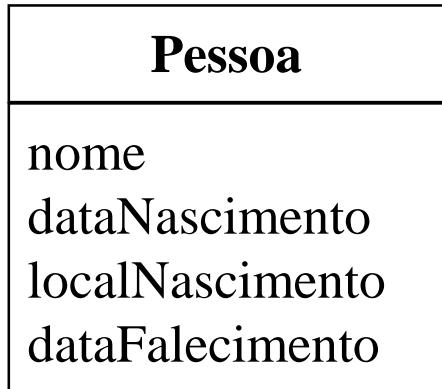
Restrições

- Uma restrição especifica uma condição que tem de se verificar no estado do sistema (objectos e ligações)
- Uma restrição é indicada por uma expressão ou texto entre chavetas ou por uma nota posicionada junto aos elementos a que diz respeito, ou a eles ligada por linhas a traço interrompido (sem setas, para não confundir com relação de dependência)
- Podem ser formalizadas em UML com a OCL - "Object Constraint Language"
- Também podem ser formalizadas (como invariantes) numa linguagem de especificação formal como VDM++

UML: Diagrama de Classes

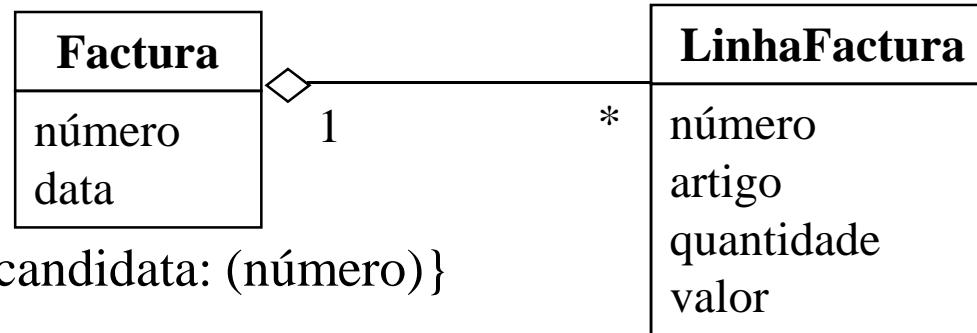
41

Restrições: em classes



{chave candidata: (nome, dataNascimento, localNascimento)}

{dataFalecimento > dataNascimento}



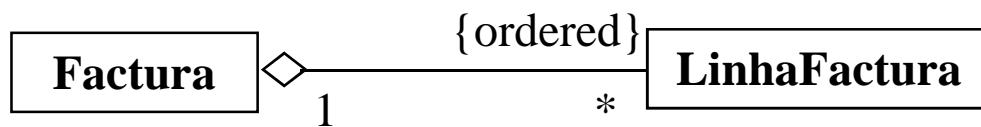
{chave candidata: (número)}

{chave candidata: (factura, número)}

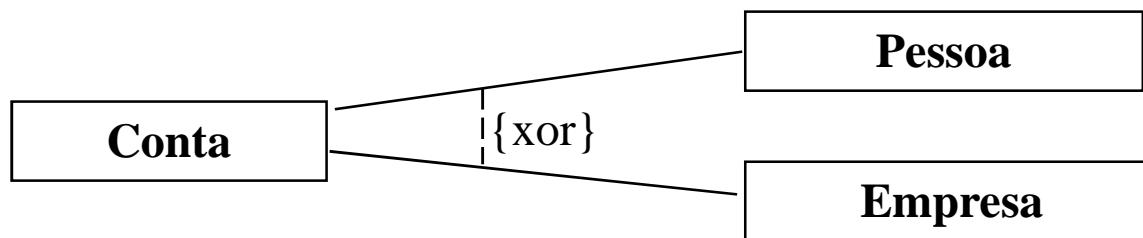
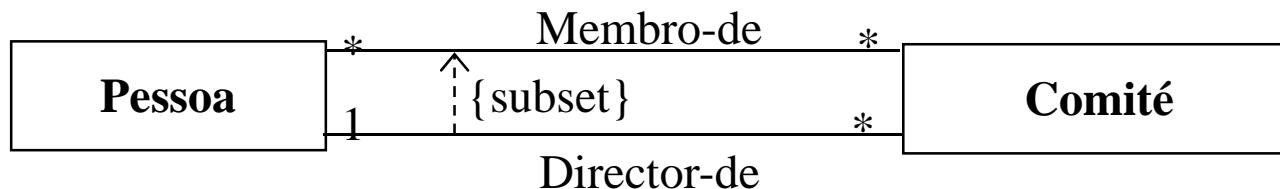
UML: Diagrama de Classes

42

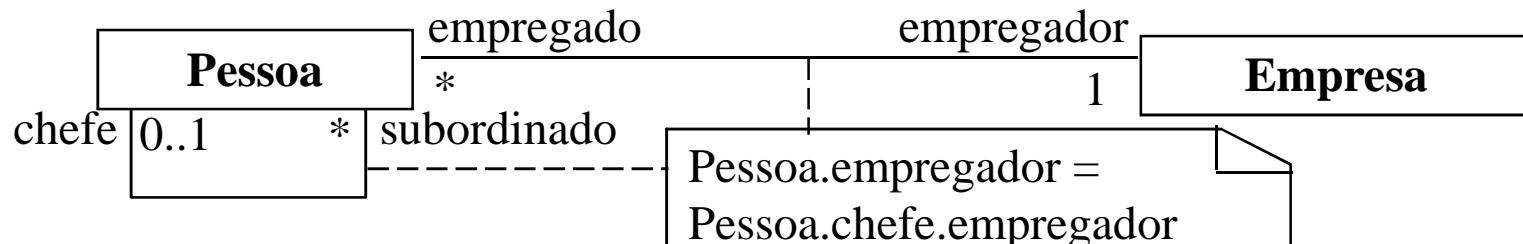
Restrições: em associações



uma factura é constituída por um conjunto ordenado de 0 ou mais linhas



associações mutuamente exclusivas



UML: Diagrama de Classes

43

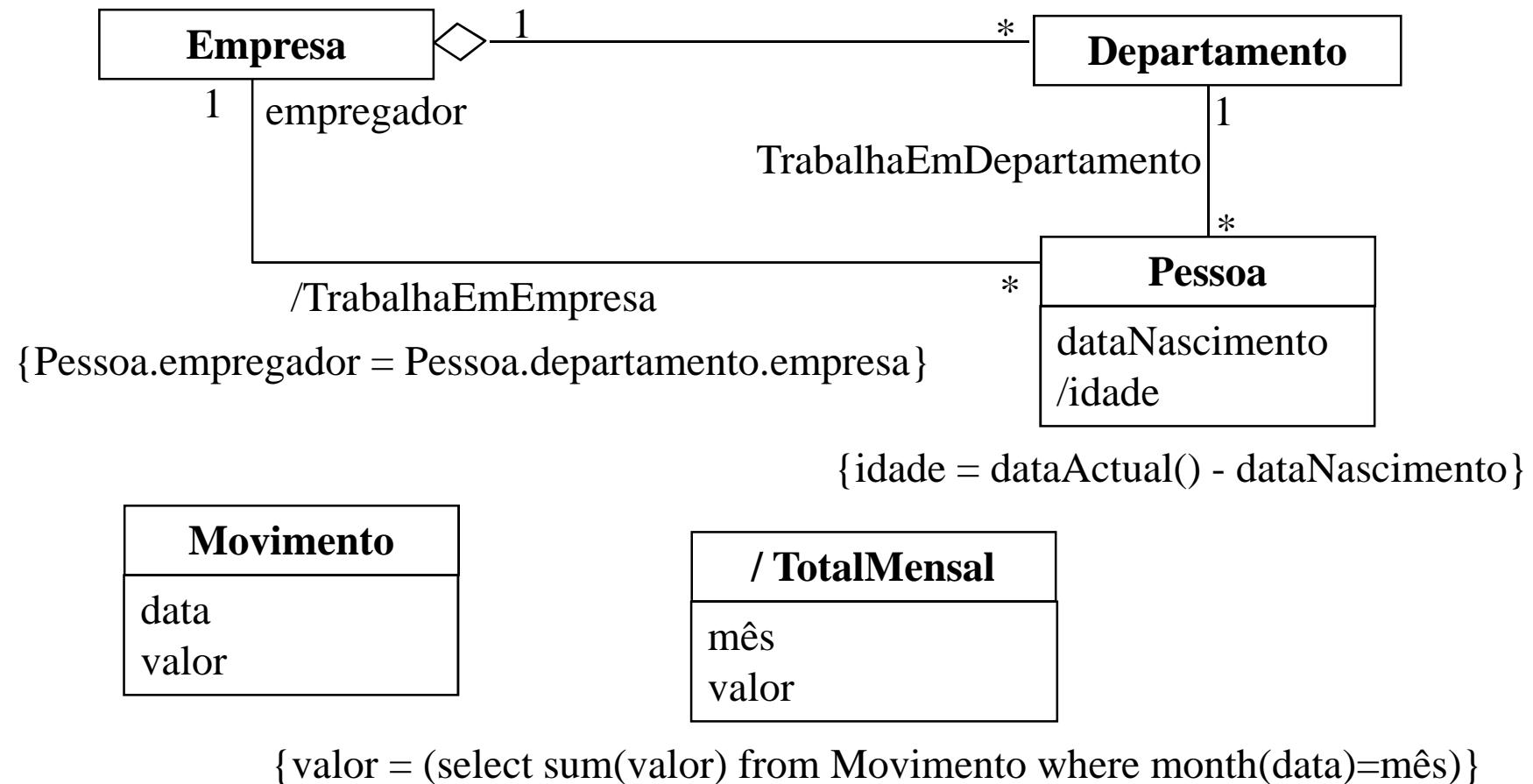
Elementos derivados

- Elemento derivado (atributo, associação ou classe): elemento calculado em função de outros elementos do modelo
- Notação: barra “/” antes do nome do elemento derivado
- Um elemento derivado tem normalmente associada uma restrição que o relaciona com os outros elementos

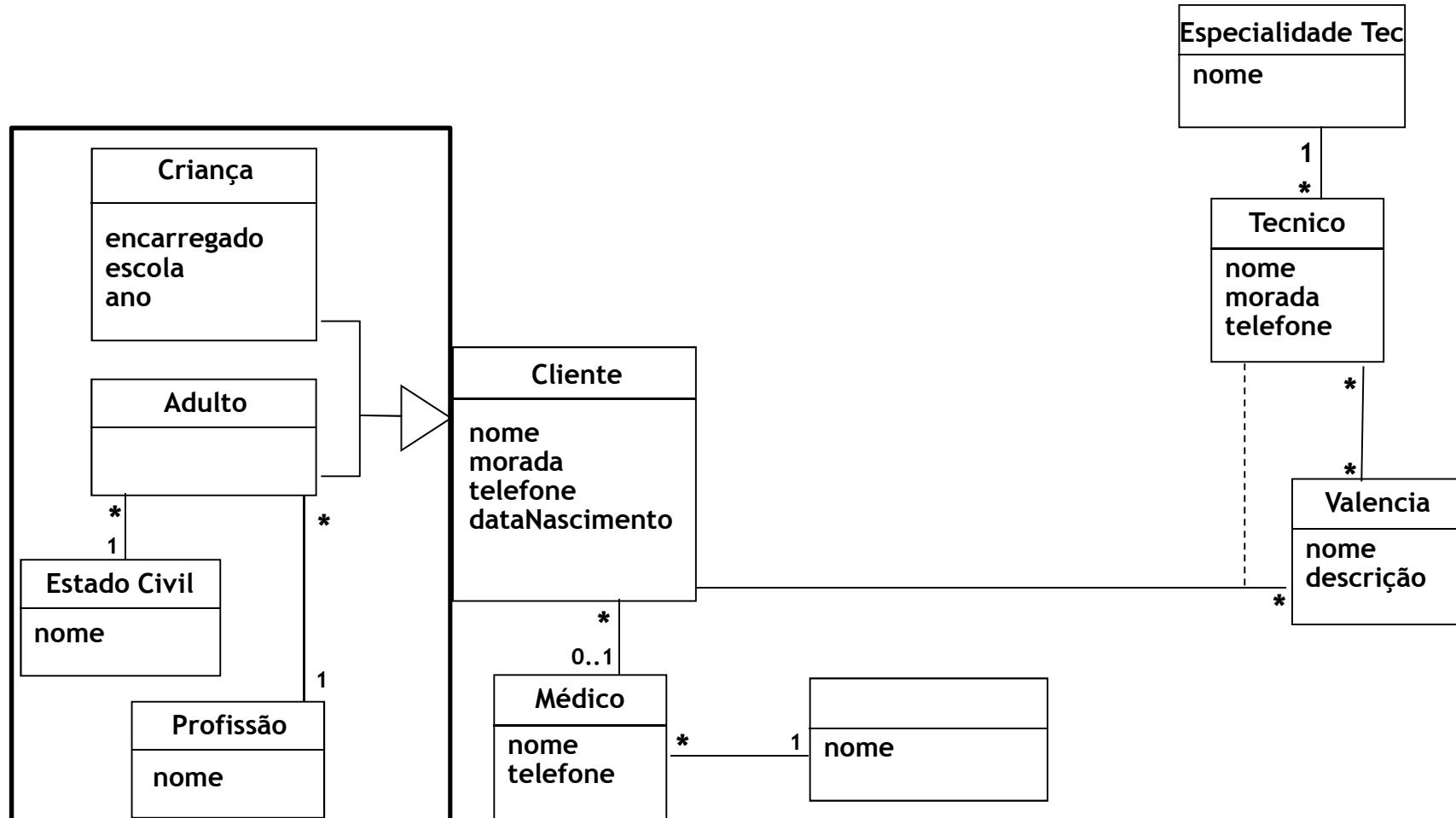
UML: Diagrama de Classes

44

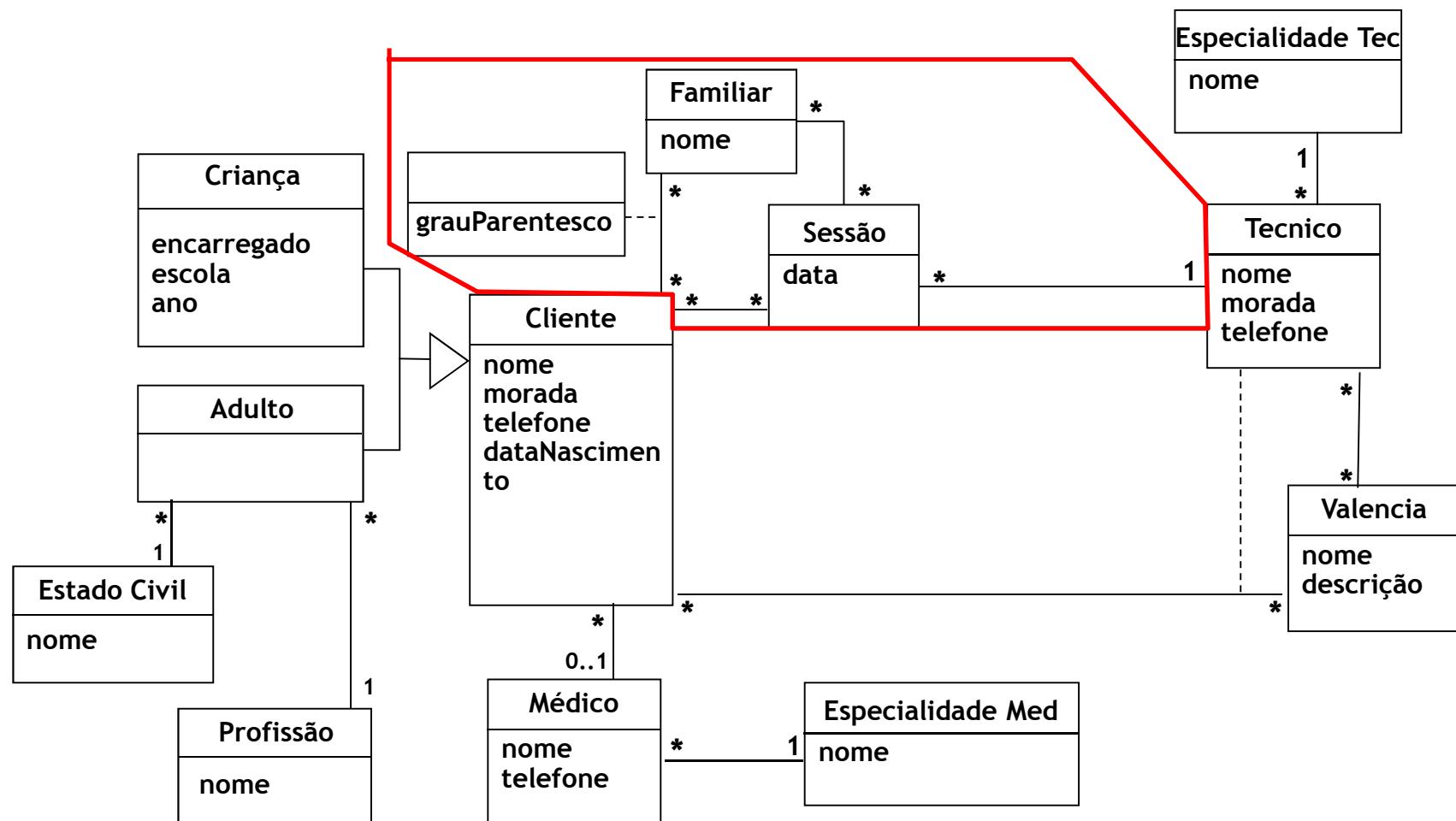
Elementos derivados



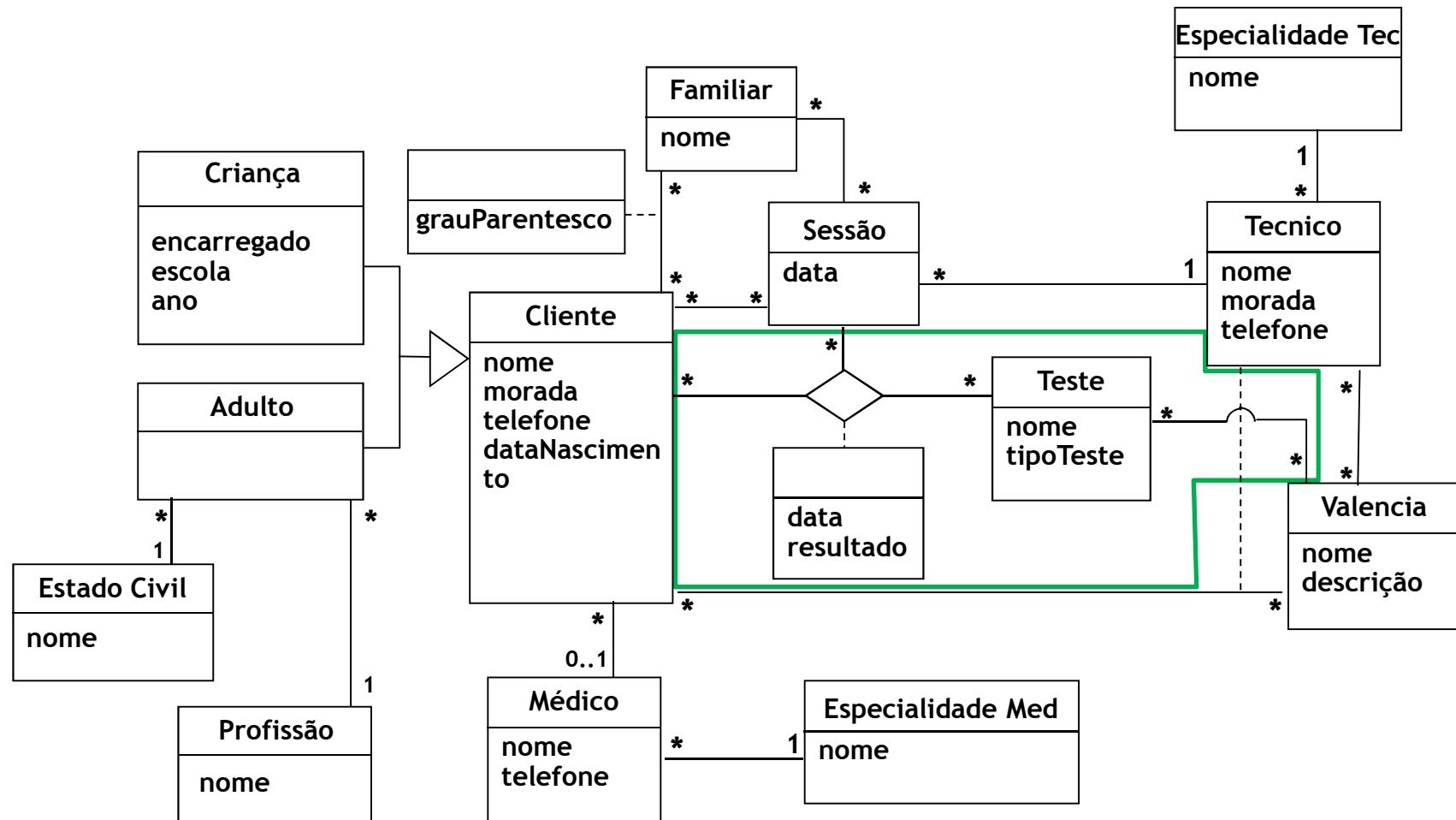
Os clientes do gabinete são maioritariamente crianças, mas um pequeno número são adultos. Sobre as crianças é necessário saber quem é o encarregado de educação, o nome da escola que frequentam e o ano em que estão. Sobre os adultos é necessário saber a profissão e o estado civil.



As actividades de gabinete são realizadas em sessões que podem ser em grupo ou individuais. Sobre cada sessão é importante saber a data em que se realizou e os participantes. Em cada sessão, além de um técnico do gabinete, podem participar vários clientes e também alguns familiares. Sobre estes é necessário saber o seu nome e o grau de parentesco com o cliente.



Durante as sessões os clientes podem efectuar diversos testes (psicotécnicos, WISC, COPS, etc.), sendo importante saber para cada um o nome, o tipo de teste e as valências com que está relacionado. De cada vez que um cliente realiza um teste é importante guardar a data e o resultado do teste.



UML: Diagrama de Classes

48

Mapeamento para MR: classes

- Cada classe é mapeada para uma relação (tabela)
 - a tabela terá os mesmos atributos que a classe respectiva, mais o atributo referente ao identificador de objecto
 - decide-se para cada atributo da tabela se poderá ou não ter valores nulos
 - atribui-se a cada atributo um domínio, pré-definido ou definido pelo utilizador

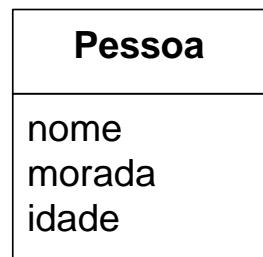


Tabela	Atributo	Nulos?	Domínio
Pessoa	pessoa-ID	N	ID
	nome	N	Nome
	morada	Y	Endereço
	idade	Y	Idade
	Chave candidata:	pessoa-ID	
	Chave primária:	pessoa-ID	
	Atributos mais acedidos:	(pessoa-ID) (nome)	

UML: Diagrama de Classes

49

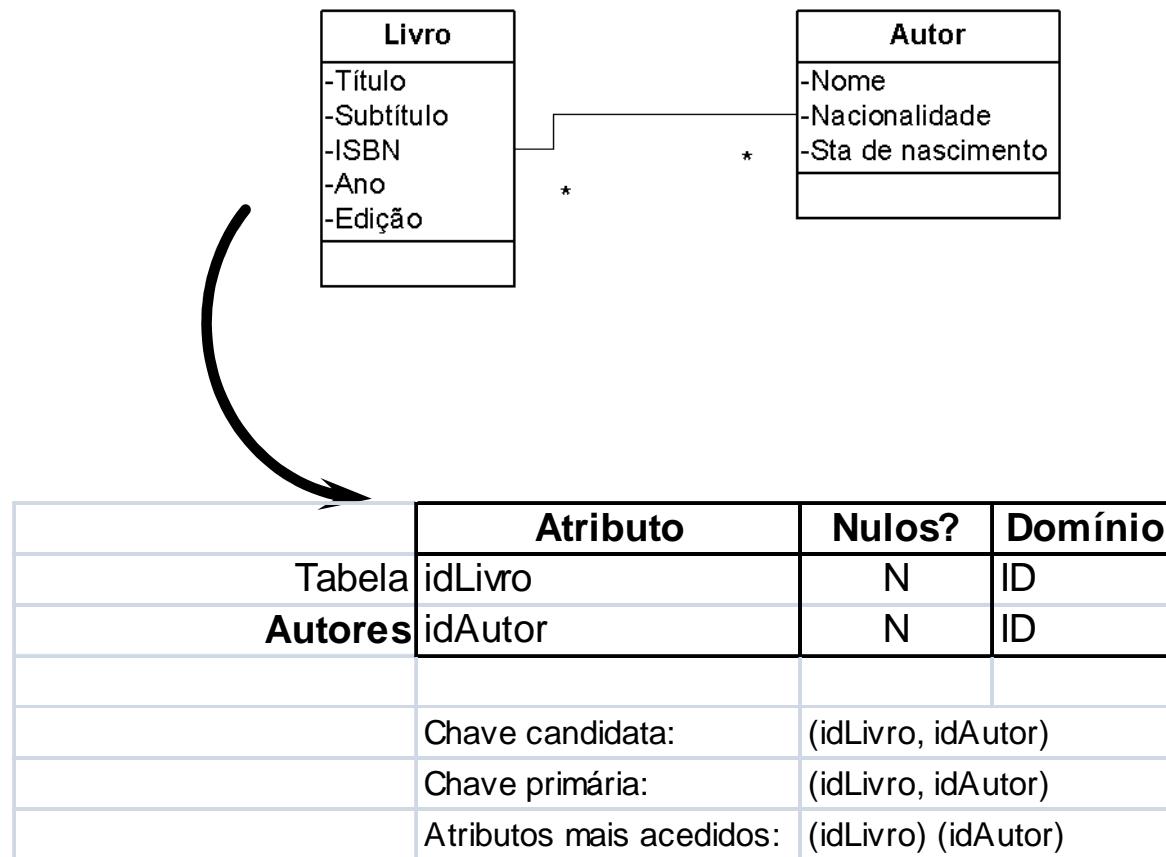
Mapeamento para MR: associações muitos-para-muitos

- Em geral, uma associação pode ou não ser mapeada para uma tabela, dependendo:
 - do tipo e multiplicidade da associação
 - das preferências do projectista em termos de extensibilidade, número de tabelas e compromissos de desempenho
- Uma associação muitos-para-muitos é sempre mapeada para uma tabela distinta
 - a chave primária será constituída pela concatenação das chaves primárias de cada uma das tabelas envolvidas na associação

UML: Diagrama de Classes

50

Mapeamento para MR: associações muitos-para-muitos



UML: Diagrama de Classes

51

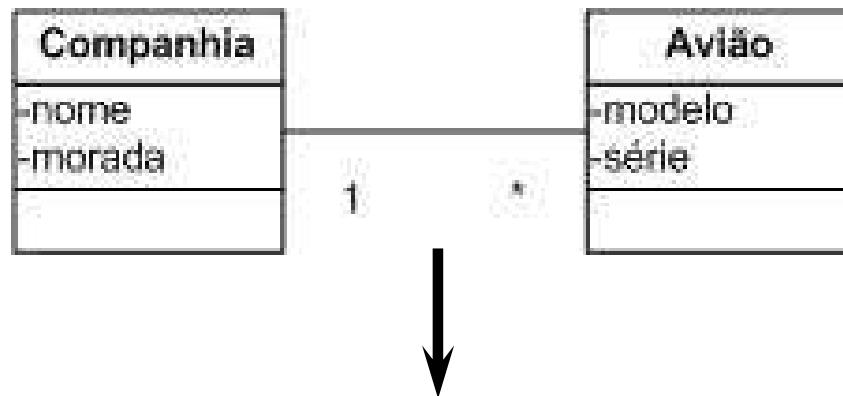
Mapeamento para MR: associações um-para-muitos

- Uma associação um-para-muitos pode ser mapeada de 2 formas
 - criação de uma tabela distinta para a associação
 - adicionar uma chave-externa na tabela relativa a muitos
- Vantagens de não criar uma tabela distinta
 - menos tabelas no esquema final
 - maior desempenho devido a um menor número de tabelas a navegar
- Desvantagens de não criar uma tabela distinta
 - menos rigor em termos de projecto do esquema
 - extensibilidade reduzida

UML: Diagrama de Classes

52

Mapeamento para MR: associações um-para-muitos

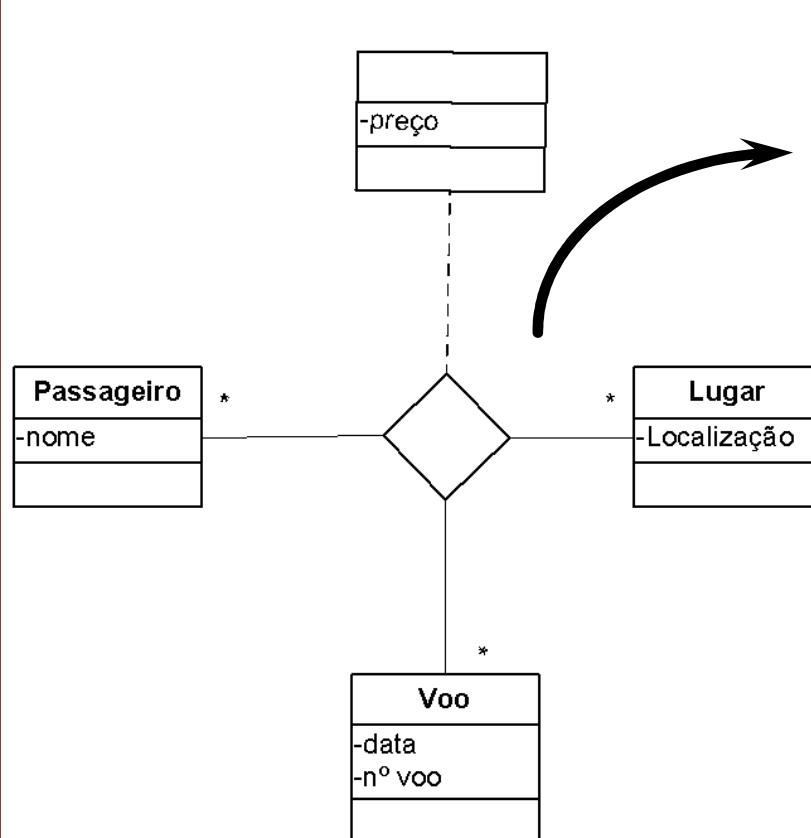


Atributo	Nulos?	Domínio
avião-ID	N	ID
modelo	Y	Text
no_serie	Y	Text
companhia-ID	Y	ID
Chave candidata:	(avião-ID)	
Chave primária:	(avião-ID)	
Atributos mais acedidos	(avião-ID) (no_serie) (companhia-ID)	

UML: Diagrama de Classes

53

Mapeamento para MR: associações ternárias



	Atributo	Nulos?	Domínio
Tabela	passageiro-ID	N	ID
Preço	voo-ID	N	ID
	lugar-ID	N	ID
	preço	Y	Montante
	Chave candidata:	(passageiro-ID, voo-ID, lugar-ID)	
	Chave primária:	(passageiro-ID, voo-ID, lugar-ID)	
	Atributos mais acedidos	(passageiro-ID) (voo-ID) (lugar-ID)	

E como será o mapeamento de associações ternárias com outras multiplicidades?

UML: Diagrama de Classes

54

Mapeamento para MR: generalizações (1)

1. Tanto a superclasse como cada uma das subclasses são mapeadas para tabelas distintas
 - abordagem logicamente correcta e extensível
 - envolve várias tabelas, pelo que a navegação da superclasse para as subclasses pode tornar-se lenta

UML: Diagrama de Classes

55

Mapeamento para MR: generalizações (1)

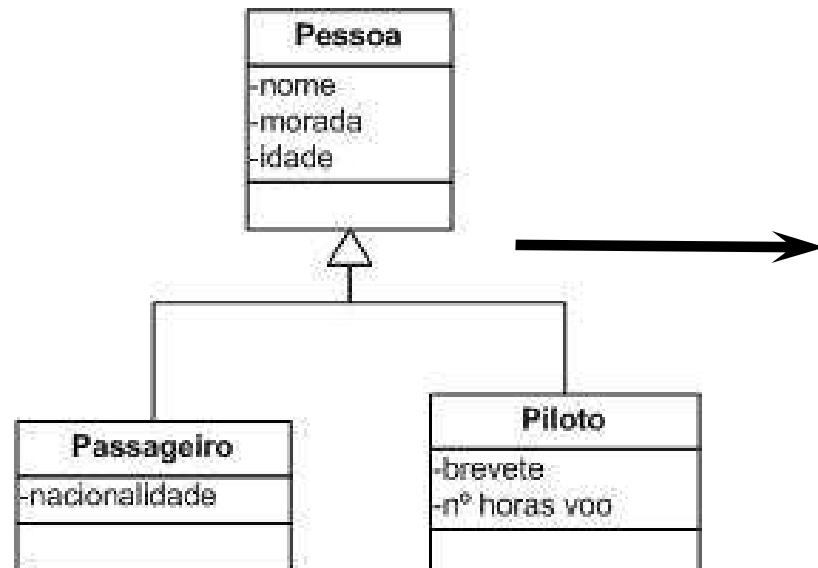


Tabela	Atributo	Nulos?	Domínio
Pessoa	pessoa-ID	N	ID
	nome	N	Nome
	morada	Y	Endereço
	idade	Y	Idade
	tipo-de-pessoa	N	Tipo-Pessoa
	Chave candidata:	(pessoa-ID)	
	Chave primária:	(pessoa-ID)	
	Atributos mais acedidos	(pessoa-ID) (nome)	
Tabela	Atributo	Nulos?	Domínio
Passageiro	pessoa-ID	N	ID
	nacionalidade	Y	País
	Chave candidata:	(pessoa-ID)	
	Chave primária:	(pessoa-ID)	
	Atributos mais acedidos	(pessoa-ID)	
Tabela	Atributo	Nulos?	Domínio
Piloto	pessoa-ID	N	ID
	brevete	Y	Text
	horas_de_voo	Y	Inteiro
	Chave candidata:	(pessoa-ID)	
	Chave primária:	(pessoa-ID)	
	Atributos mais acedidos	(pessoa-ID)	

UML: Diagrama de Classes

56

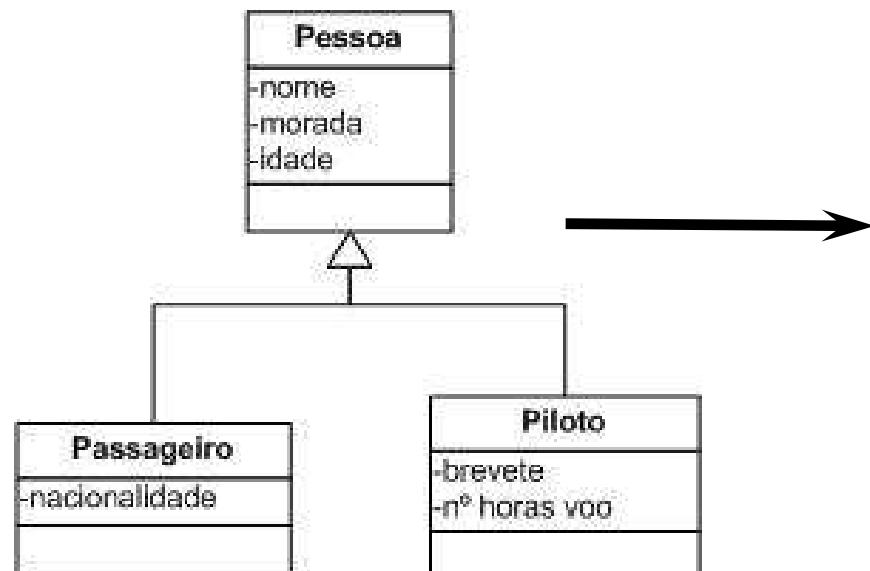
Mapeamento para MR: generalizações (2)

- 2. Eliminação da navegação da superclasse-para-subclasse
 - elimina a tabela da superclasse e replica todos os atributos dela em cada subclasse
 - ideal, quando a superclasse possui poucos atributos e as subclasses muitos atributos
 - não se pode garantir a unicidade dos valores dos atributos da superclasse

UML: Diagrama de Classes

57

Mapeamento para MR: generalizações (2)



Atributo	Nulos?	Domínio
pessoa-ID	N	ID
nome	N	Nome
morada	Y	Endereço
idade	Y	Idade
nacionalidade	Y	País

Chave candidata: (pessoa-ID)
Chave primária: (pessoa-ID)
Atributos mais acedidos: (pessoa-ID)

Atributo	Nulos?	Domínio
pessoa-ID	N	ID
nome	N	Nome
morada	Y	Endereço
idade	Y	Idade
brevete	Y	Text
horas_de_voo	Y	Inteiro

Chave candidata: (pessoa-ID)
Chave primária: (pessoa-ID)
Atributos mais acedidos: (pessoa-ID)

UML: Diagrama de Classes

58

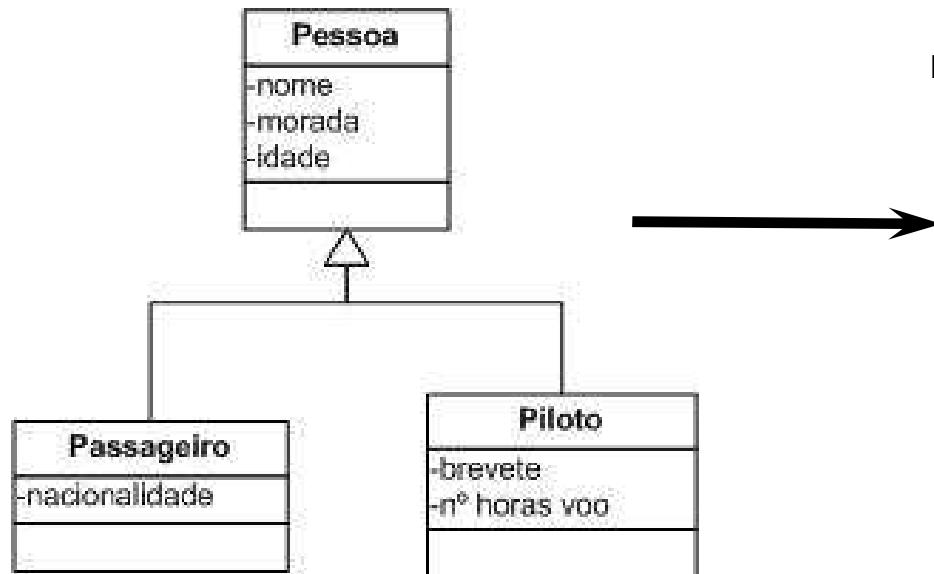
Mapeamento para MR: generalizações (3)

- 3. Uma tabela para a superclasse
 - criação de apenas uma tabela para a superclasse
 - todos os atributos das subclasses são acrescentados à tabela da superclasse
 - cada subclass utiliza apenas os atributos que lhe pertencem, deixando os restantes com valores nulos
 - viola a terceira forma normal
 - abordagem interessante quando em presença de poucas subclasses (2 ou 3)

UML: Diagrama de Classes

59

Mapeamento para MR: generalizações (3)



Atributo	Nulos?	Domínio
pessoa-ID	N	ID
nome	N	Nome
morada	Y	Endereço
idade	Y	Idade
tipo-de-pessoa	N	Tipo-Pessoa
nacionalidade	Y	País
brevete	Y	Text
horas_de_voo	Y	Inteiro

Chave candidata: (pessoa-ID)

Chave primária: (pessoa-ID)

Atributos mais acedidos: (pessoa-ID)

UML: Diagrama de Classes

60

Mapeamento para MR

- O mapeamento para BD Relacionais não é único
 - existem três formas de mapear uma generalização
 - é necessário adicionar detalhes que não existem no modelo de objectos, tais como, chaves primárias e chaves candidatas, se um atributo pode ter valores nulos ou não
 - obriga à atribuição de um domínio a cada atributo
 - e obriga a listar os atributos mais frequentemente acedidos

UML: Diagrama de Classes

61

Mapeamento para MR

- Resumindo
 - criar um atributo *id* correspondente a cada classe
 - classe implementada por *relação*
 - associação muitos-para-muitos ou ternária implementada por *relação*
 - associação muitos-para-um implementada por *atributo extra* (*id* do lado 1) na relação do lado muitos (*)
 - traduzir hierarquias de uma das três formas apresentadas

Exercício (continuação)

62

Mapeamento para MR

- Converter o diagrama de classes obtido anteriormente para o modelo relacional.

Exercício (continuação)

63

Mapeamento para MR

- Cliente(idCliente, nome, morada, telefone, dataNasc, idMedico->Medico, tipoCliente)
- Crianca(idCrianca->Cliente.idCliente, encarregado, escola, ano)
- Adulto(idAdulto->Cliente.idCliente, idEstadoCivil->EstadoCivil, idProfissao->Profissao)
- EstadoCivil(idEstadoCivil, nome)
- Profissao(idProfissao, nome)
- Medico(idMedico, nome, telefone, idEspecialidadeMed->EspecialidadeMed)
- EspecialidadeMed(idEspecialidadeMed, nome)
- Familiar(idFamiliar, nome)
- Sessao(idSessao, data, idTecnico->Tecnico)
- Teste(idTeste, nome, tipoTeste)
- Parentesco(idFamiliar->Familiar, idCliente->Cliente, grauParentesco)
- ParticipacoesFam(idFamiliar->Familiar, idSessao->Sessao)
- ParticipacoesCli(idCliente->Cliente, idSessao->Sessao)
- ResultadosTeste(idCliente->Cliente, idSessao->Sessao, idTeste->Teste, data, resultado)
- Tecnico(idTecnico, nome, morada, telefone, idEspecialidadeTec->EspecialidadeTec)
- EspecialidadeTec(idEspecialidadeTec, nome)
- Valencia(idValencia, nome, descricao)
- ValenciaTeste(idValencia->Valencia, idTeste->Teste)
- ValenciaCliente(idValencia->Valencia, idCliente->Cliente, idTecnico->Tecnico)
- ValenciaTecnico(idValencia->Valencia, idTecnico->Tecnico)

Bases de Dados



Universidade do Porto

Faculdade de Engenharia

FEUP

1

- INTRODUÇÃO
- MODELOS CONCEPTUAIS
 - Diagrama de Classes UML
 - Modelo Entidade-Associação (E-A)
- MODELO RELACIONAL
- LINGUAGEM DE DEFINIÇÃO DE DADOS (LDD)
- INTERROGAÇÃO DE DADOS
 - Álgebra relacional
 - Linguagem de Manipulação de Dados (LMD)

Observação: parcialmente baseado em slides desenvolvidos pelo Prof. Jeffrey D. Ullman

Índice

2

- Relações e atributos
- Dependências funcionais
- Chaves de relações
- Inferência de dependências funcionais
- Formas normais

Relações e atributos

3

O modelo relacional caracteriza-se por um conjunto de relações cada uma delas identificada por um nome e um conjunto de atributos

Exemplo

Valencia(idValencia, nome, descricao) - Relação Valencia com os atributos idValencia, nome e descricao.

Dependências funcionais

4

- Os atributos das relações relacionam-se entre si com base em regras a que se chamam **dependências funcionais**
- $X \rightarrow Y$ é uma **dependência funcional** sobre a relação R dizendo que quaisquer dois tuplos de R que sejam iguais em todos os atributos de X , têm também de ser iguais em todos os atributos de Y .
 - Diz-se “ $X \rightarrow Y$ verifica-se em R .”
 - **Notação:** ..., X, Y, Z representam conjuntos de atributos; A, B, C, \dots representam atributos isolados.

Dependências funcionais

5

Separação dos lados direitos

- $X \rightarrow A_1A_2\dots A_n$ verifica-se em R exactamente quando cada um dos $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$ se verifica em R .
- Exemplo: $A \rightarrow BC$ é equivalente a $A \rightarrow B$ e $A \rightarrow C$.
- Não há regras de separação para os lados esquerdos.
- Habitualmente expressam-se as DF's pondo um só atributo do lado direito.

Dependências funcionais

6

Exemplo

**Encomendas(idEncomenda, dataEncomenda,
idProduto, nomeProduto, quantidade,
precoUnit, idCliente, nomeCliente)**

DF's que se podem assumir:

- $\text{idEncomenda} \rightarrow \text{dataEncomenda}$
- $\text{idEncomenda} \rightarrow \text{idCliente}$
- $\text{idProduto} \rightarrow \text{nomeProduto}$
- $\text{idProduto} \rightarrow \text{precoUnit}$
- $\text{idCliente} \rightarrow \text{nomeCliente}$
- $\text{idEncomenda}, \text{idProduto} \rightarrow \text{quantidade}$

Chaves de relações

7

- K é uma *super-chave* da relação R se K determina funcionalmente todos os atributos de R .
- K é uma *chave* (*também se pode designar por chave candidata*) de R se K é super-chave, e nenhum subconjunto próprio de K é super-chave.
 - ✖ Um subconjunto diz-se *próprio* se estiver contido mas não for igual ao conjunto K

Chaves de relações

8

Super-chave: exemplo

Encomendas(idEncomenda, dataEncomenda,
idProduto, nomeProduto, quantidade,
precoUnit, idCliente, nomeCliente)

{idEncomenda, idProduto, quantidade} é uma **super-chave** porque estes atributos determinam todos os outros atributos.

- idEncomenda -> dataEncomenda idCliente nomeCliente
- idProduto -> nomeProduto precoUnit

Chaves de relações

9

Chave: exemplo

- $\{\text{idEncomenda}, \text{idProduto}, \text{quantidade}\}$ não é uma **chave** porque existe um seu subconjunto próprio $\{\text{idEncomenda}, \text{idProduto}\}$ que é uma **super-chave**.
- Não há mais chaves para a relação Encomendas, mas há muitas super-chaves.
 - Qualquer super-conjunto de $\{\text{idEncomenda}, \text{idProduto}\}$.
 - ✖ Um super-conjunto do conjunto S é um conjunto que contém S, i.e., possui todos os elementos de S e possivelmente mais alguns.

Chaves de relações

10

Chave primária

Designa-se por **chave primária** uma e uma só chave, de entre as chaves dessa relação.

- O conceito de chave primária não é importante no modelo relacional. No entanto, os SGBD costumam exigir a definição de uma chave primária. Por isso, é costume assinalar a chave primária de uma relação no modelo relacional (atributos sublinhados).
 - SGBD = Sistema Gestor de Bases de Dados (Ex.: MySQL, Oracle, SQLite, ...)

Encomendas(idEncomenda, dataEncomenda, idCliente)

Chaves de relações

11

Chave externa

Designa-se por **chave externa** o conjunto de atributos de uma relação que é chave de uma outra relação.

Encomenda(idEncomenda, dataEncomenda, idCliente -> Cliente)

Cliente(idCliente, nomeCliente)

Quants(idEncomenda -> Encomenda, idProduto -> Produto, quantidade)

Produto(idProduto, nomeProduto)

Inferência de DFs

12

- São-nos dadas as DFs $X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots, X_n \rightarrow A_n$, e queremos saber se uma DF $Y \rightarrow B$ é garantida numa relação que satisfaça as DFs dadas.
 - Exemplo das encomendas: Se $idEncomenda \rightarrow idCliente$ e $idCliente \rightarrow nomeCliente$ se verificam, seguramente $idEncomenda \rightarrow nomeCliente$ também se verifica, mesmo que tal não seja dito de forma explícita.

Inferência de DFs

13

Teste de inferência

1. Para testar se $Y \rightarrow B$, começar por assumir que dois registos têm todos os atributos de Y iguais.
 2. Usar as DFs dadas para inferir se esses registos também têm de coincidir noutros atributos.
 - Se B é um desses atributos, então $Y \rightarrow B$ é verdade.
 - Caso contrário, os dois registos, com algumas igualdades forçadas, formam uma relação de dois registos provando que $Y \rightarrow B$ não provem das DFs dadas.
- $\leftarrow Y \rightarrow$
- 0000000...0
- 00000??...?

Inferência de DFs

14

Teste de inferência

Uma forma fácil de testar se $Y \rightarrow B$ provem das DFs dadas é através do cálculo do *fecho* de Y (denota-se por Y^+).

- **Base:** $Y^+ = Y$.
- **Indução:** Procurar por lados esquerdos (X) das DFs que sejam sub-conjuntos do actual Y^+ . Se a DF é $X \rightarrow A$, acrescentar A a Y^+ .
- **Conclusão:** Se no final B pertencer a Y^+ , então a DF $Y \rightarrow B$ provem das DFs dadas sendo desnecessário acrescentá-la às existentes.

Inferência de DFs

15

Teste de inferência: indução de DFs para subconjuntos de atributos da relação original

Ideia base

1. Começar com as DFs dadas e procurar todas as DFs *não triviais* deduzíveis a partir das DFs dadas.
 - Não trivial = lado direito não está contido no lado esquerdo.
2. Utilizar apenas as DFs que envolvem só atributos do subconjunto pretendido.

Inferência de DFs

16

Teste de inferência: indução

Simples, Algoritmo Exponencial

1. Para cada conjunto de atributos X , calcular X^+ .
2. Acrescentar $X \rightarrow A$ para todos os A em $X^+ - X$.
3. Apagar $XY \rightarrow A$ sempre que se descobrir $X \rightarrow A$.
 - ◆ Porque $XY \rightarrow A$ provém de $X \rightarrow A$ **em qualquer projeção**.
4. Finalmente, usar só DFs que envolvam atributos projetados.

Inferência de DFs

17

Teste de inferência: exemplo

- ABC com as DFs $A \rightarrow B$ e $B \rightarrow C$.
- Objectivo: determinar as DFs para o subconjunto $\{A,C\}$
- Iterações:
 - $A^+ = ABC$; logo $A \rightarrow B$, $A \rightarrow C$.
 - ✖ Não é necessário calcular AB^+ ou AC^+ .
 - $B^+ = BC$; logo $B \rightarrow C$.
 - ✖ Não é necessário calcular BC^+ .
 - $C^+ = C$; não assegura nenhuma DF não trivial.
- DFs resultantes: $A \rightarrow B$, $A \rightarrow C$ e $B \rightarrow C$.
- Nota: não se procurou AB^+ nem AC^+ porque pertencem a A^+ . Nem BC^+ (porque ...), CA^+ ($=AC^+$), ou CB^+ ($=BC^+$).

Inferência de DFs

18

Teste de inferência: exemplo

- Projectar exemplo anterior em AC
- Projecção em $AC : A \rightarrow C$.
 - Só as DFs que envolvem subconjuntos de $\{A,C\}$.

Formas normais

19

Exemplo

Pretende-se armazenar a informação relativa a uma época do campeonato de Fórmula 1.

De cada marca participante no campeonato pretende-se armazenar o seu nome, país de origem, nº actual de pontos no campeonato de marcas e quais os carros inscritos. De cada carro interessa saber o seu peso, potência e velocidade máxima.

Relativamente aos pilotos participantes é necessário conhecer o seu nome, morada, idade, nacionalidade e nº actual de pontos no campeonato de pilotos. Um piloto só pode conduzir um carro ao longo da época, embora um determinado carro possa ser conduzido por mais de um piloto. Esta situação, embora não muito frequente, pode surgir, por exemplo, devido ao afastamento de um piloto ferido num acidente.

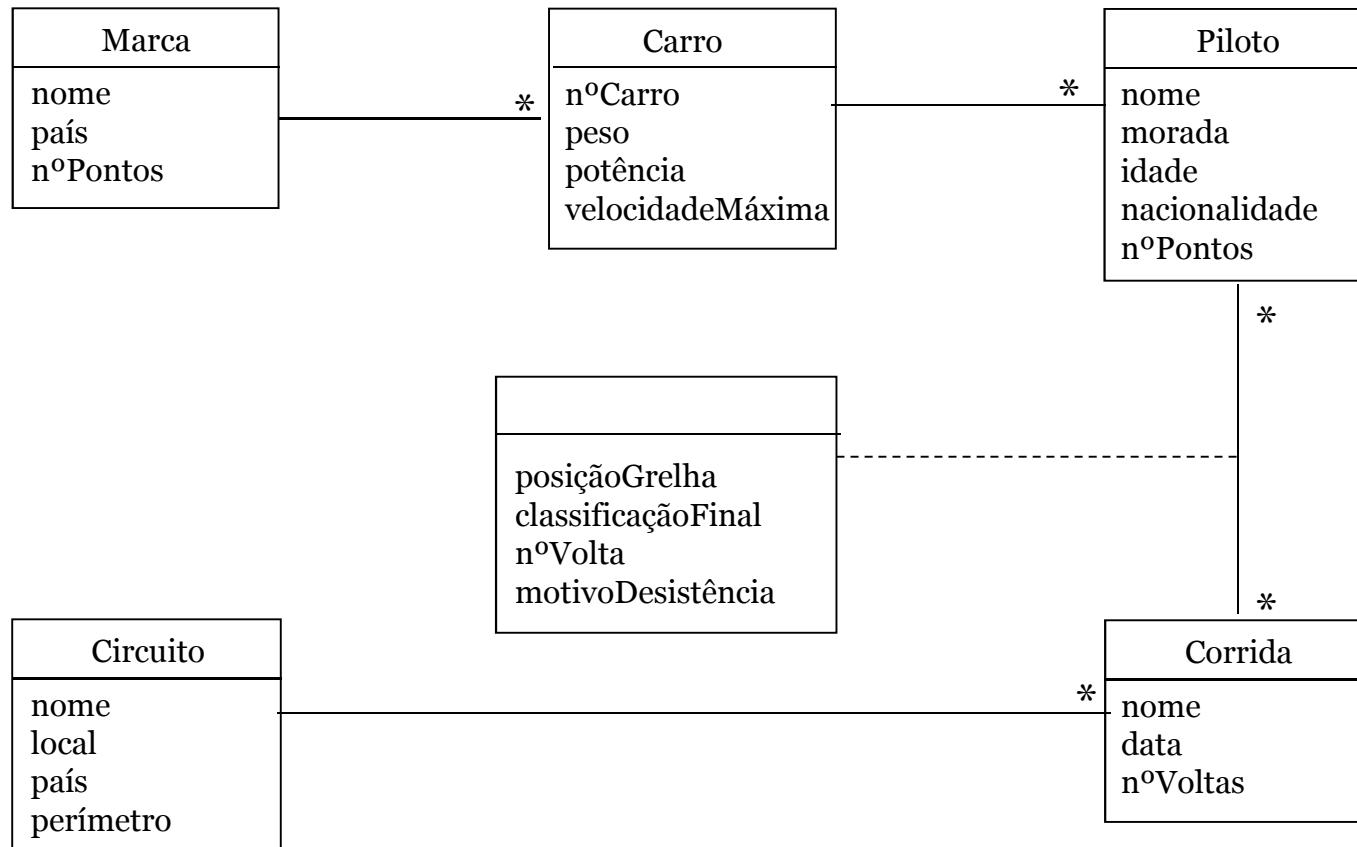
Cada época é constituída por um conjunto de corridas que se realizam em circuitos e em datas definidas no início da época. Para uma determinada corrida pode, ao longo da época e por razões várias, ser alterado o circuito onde esta se realiza. Em situações excepcionais pode acontecer também a realização de duas corridas no mesmo circuito. De cada circuito pretende-se saber o nome, local, país, nº de voltas e perímetro.

No que diz respeito à realização de uma corrida interessa saber quais os pilotos que participaram, as posições que ocuparam na grelha de partida e a classificação final. Relativamente à corrida interessa também saber quais os pilotos que desistiram, em que volta ocorreu e qual o motivo da desistência.

Formas normais

20

Exemplo: Diagrama de classes UML



Formas normais

21

1^a FN

O domínio de cada atributo contém apenas valores atômicos, e o valor de cada atributo contém apenas um único valor do domínio.

Exemplo

Classificação(nome_marca, país_origem, pontuação_marca, num_carro, peso_carro, potência_carro, vel_max, nome_piloto, morada_piloto, idade_piloto, nacionalidade_piloto, pontuação_piloto, nome_circuito, local_circuito, país_circuito, num_voltas_circuito, perímetro, nome_corrida, data, posição_grelha, classificação, motivo_desistência, num_voltas_realizadas)

Formas normais

22

2^a FN

Está na 1^a FN e nenhum atributo não primo da relação é funcionalmente dependente de um subconjunto próprio de uma chave candidata

- Atributo primo = pertence a alguma chave candidata da relação.

Exemplo

Piloto(nome_piloto, morada_piloto, idade_piloto, nacionalidade_piloto, pontuação_piloto, nome_marca, país_origem, pontuação_marca, num_carro, peso_carro, potência_carro, vel_max)

Corrida(nome_corrida, data, nome_circuito, local_circuito, país_circuito, num_voltas_circuito, perímetro)

Classificação(nome_piloto->Piloto, nome_corrida->Corrida, posição_grelha, classificação, motivo_desistência, num_voltas_realizadas)

Formas normais

23

3^a FN

Está na 2^a FN e todos os atributos não primos da relação dependem funcionalmente de todas as chaves candidatas da relação de forma não transitiva

Exemplo

Piloto(nome_piloto, morada_piloto, idade_piloto, nacionalidade_piloto, pontuação_piloto, num_carro->Carro)

Carro(num_carro, peso_carro, potência_carro, vel_max, nome_marca->Marca)

Marca(nome_marca, país_origem, pontuação_marca)

Corrida(nome_corrida, data, nome_circuito->Circuito, num_voltas_circuito)

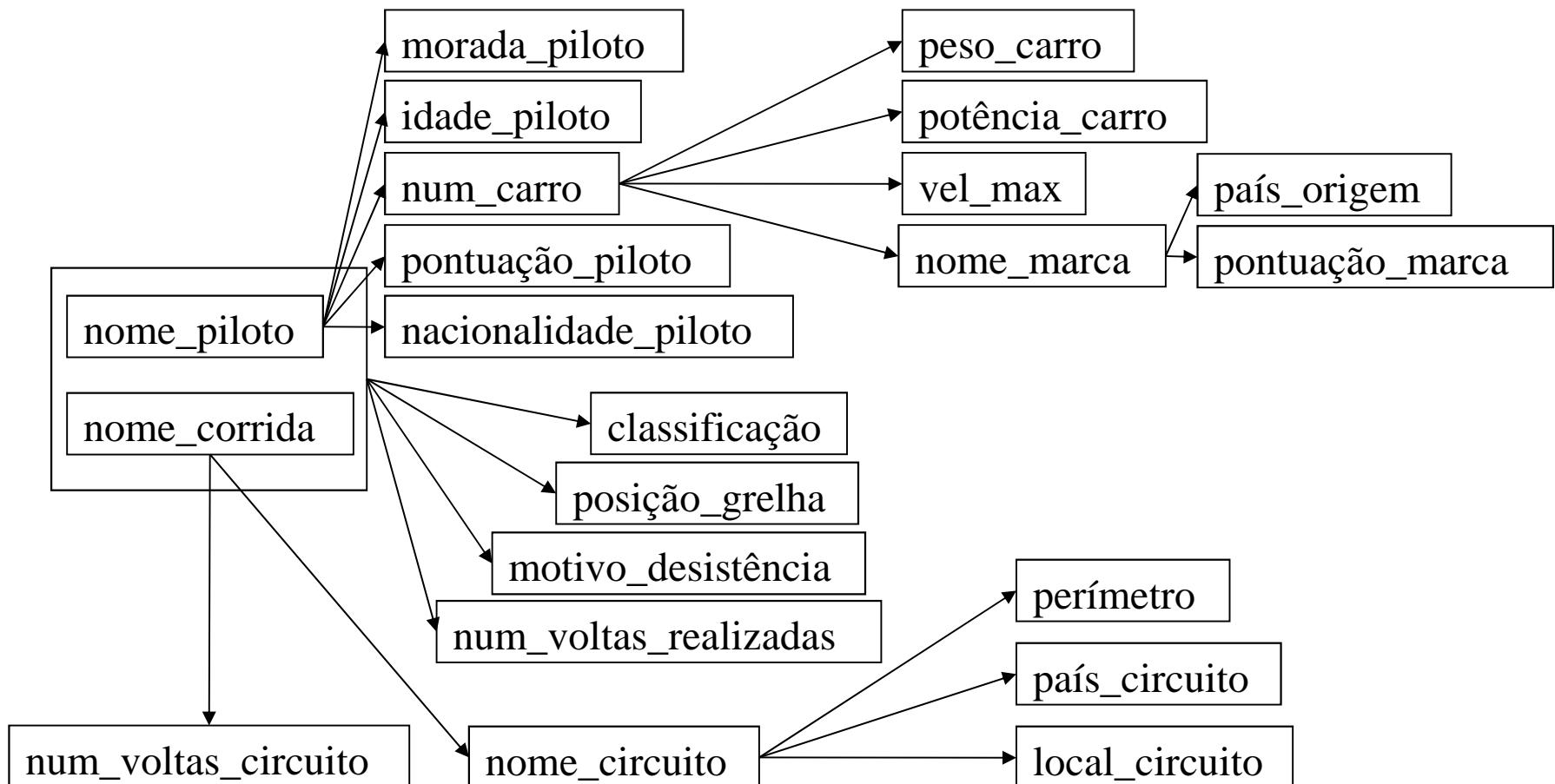
Circuito(nome_circuito, local_circuito, país_circuito, perímetro)

Classificação(nome_piloto->Piloto, nome_corrida->Corrida, posição_grelha, classificação, motivo_desistência, num_voltas_realizadas)

Formas normais

24

Diagrama de dependências funcionais: exemplo



Formas normais

25

Normalização vs. espaço ocupado pela BD

Sabendo que existem 11 equipas, 22 carros, 28 pilotos, 16 corridas e 16 circuitos e ainda que em cada corrida participam exactamente 22 pilotos, **estime o espaço ocupado pelas tabelas nas 1^a, 2^a e 3^a formas normais.**

Admita que o espaço ocupado por cada atributo é de 4 bytes para os atributos numéricos e para as datas e de 30 bytes para os atributos texto.

1^a FN: **Classificação**(nome_marca, país_origem, pontuação_marca, num_carro, peso_carro, potência_carro, vel_max, nome_piloto, morada_piloto, idade_piloto, nacionalidade_piloto, pontuação_piloto, nome_circuito, local_circuito, país_circuito, num_voltas_circuito, perímetro, nome_corrida, data, posição_grelha, classificação, motivo_desistência, num_voltas_realizadas)

2^a FN: **Piloto**(nome_piloto, morada_piloto, idade_piloto, nacionalidade_piloto, pontuação_piloto, nome_marca, país_origem, pontuação_marca, num_carro, peso_carro, potência_carro, vel_max)

Corrida(nome_corrida, data, nome_circuito, local_circuito, país_circuito, num_voltas_circuito, perímetro)

Classificação(nome_piloto->**Piloto**, nome_corrida->**Corrida**, posição_grelha, classificação, motivo_desistência, num_voltas_realizadas)

3^a FN: **Piloto**(nome_piloto, morada_piloto, idade_piloto, nacionalidade_piloto, pontuação_piloto, num_carro->**Carro**)

Carro(num_carro, peso_carro, potência_carro, vel_max, nome_marca->**Marca**)

Marca(nome_marca, país_origem, pontuação_marca)

Corrida(nome_corrida, data, nome_circuito->**Circuito**, num_voltas_circuito)

Circuito(nome_circuito, local_circuito, país_circuito, perímetro)

Classificação(nome_piloto->**Piloto**, nome_corrida->**Corrida**, posição_grelha, classificação, motivo_desistência, num_voltas_realizadas)

Formas normais

27

Normalização vs. espaço ocupado pela BD

1^a Forma Normal

EspaçoOcupado =

$$\begin{aligned} & 22 \times 16 \times (30+30+4+4+4+4+4+30+30+4+30+4+30+30+4+4+30+4+4+4+30+4) \\ & = 22 \times 16 \times 352 = \mathbf{123\,904\ bytes} \end{aligned}$$

2^a Forma Normal

$$\begin{aligned} \text{EspaçoOcupado} &= 28 \times (30+30+4+30+4+30+30+4+4+4+4+4) \\ &\quad + 16 \times (30+4+30+30+30+4+4) + 22 \times 16 \times (30+30+4+4+30+4) \\ &= 28 \times 178 + 16 \times 132 + 22 \times 16 \times 102 = \mathbf{52\,968\ bytes} \end{aligned}$$

3^a Forma Normal

$$\begin{aligned} \text{EspaçoOcupado} &= 28 \times (30+30+4+30+4+4) + 22 \times (4+4+4+4+30) + 11 \times (30+30+4) \\ &\quad + 16 \times (30+4+4+30) + 16 \times (30+30+30+4) + 22 \times 16 \times (30+30+4+4+30+4) \\ &= 28 \times 102 + 22 \times 46 + 11 \times 64 + 16 \times 68 + 16 \times 94 + 22 \times 16 \times 102 = \mathbf{43\,068\ bytes} \end{aligned}$$

Nota: os cálculos apresentados não são adequados para estimar o espaço efectivamente ocupado pelos dados. Realizam-se para fins exclusivamente pedagógicos.

Formas normais

28

Normalização vs. facilidade e rapidez de consulta

O Comando *SELECT* da Linguagem SQL (a forma mais simples):

```
SELECT lista_de_campos  
      FROM lista_de_tabelas  
      [WHERE lista_de_condições];
```

- Os [] indicam que é opcional

EXEMPLO: Seleccionar o nome de todos os pilotos alemães

```
SELECT nome_piloto  
      FROM piloto  
      WHERE nacionalidade_piloto="Alemanha";
```

Formas normais

29

Normalização vs. facilidade e rapidez de consulta

Usando a linguagem SQL diga quais as marcas que já obtiveram o 1º lugar da grelha de partida utilizando, para tal, as tabelas na 1ª, 2ª e 3ª Forma Normal.

1ª Forma Normal

```
SELECT nome_marca FROM Classificação  
WHERE posição_grelha=1;
```

2ª Forma Normal

```
SELECT nome_marca FROM Classificação, Piloto  
WHERE posição_grelha=1  
AND Classificação.nome_piloto = Piloto.nome_piloto;
```

3ª Forma Normal

```
SELECT nome_marca FROM Classificação, Piloto, Carro  
WHERE posição_grelha=1 AND Classificação.nome_piloto = Piloto.nome_piloto  
AND Piloto.num_carro = Carro.num_carro;
```

Formas normais

30

Exemplo

Cervejas(nome, empr, moradaEmpr)

DFs: nome->empr, empr->moradaEmpr

- A única chave é {nome}. Porquê?
- Em que forma normal se encontra esta relação?

Formas normais

31

Normalização

Actualmente não se desenham modelos relacionais recorrendo a este processo de normalização.

Porquê?

- Porque fazendo o mapeamento do modelo conceptual (por ex.: diagrama de classes UML ou modelo E-A) para o relacional, as relações já estão, tipicamente, na 3^a FN.
- Mas é importante verificar se o modelo relacional que resulta do mapeamento do modelo conceptual se encontra na forma normal pretendida.

Formas normais

32

Normalização

Muito importante:

Uma relação na 2^a FN também está na 1^a FN.

Uma relação na 3^a FN também está na 2^a FN.

Mas garantir a 3^a FN pode não ser suficiente ...

Formas normais

33

Normalização

Nível de refina- mento do MR		
	6 ^a FN (6NF)	C.J. Date, H. Darwen e N. Lorentzos (2002)
	FNCD (DKNF)	Ronald Fagin (1981)
	5 ^a FN (5NF)	Ronald Fagin (1979)
	4 ^a FN (4NF)	Ronald Fagin (1977)
	FNBC (BCNF)	Raymond F. Boyce and E.F. Codd (1974)
	3 ^a FN (3NF)	E.F. Codd (1971)
	2 ^a FN (2NF)	E.F. Codd (1971)
	1 ^a FN (1NF)	E.F. Codd (1970), C.J. Date (2003)

Formas normais

34

Normalização

Garantindo que uma base de dados se encontra na forma normal de Boyce-Codd (FNBC) e na 3^a FN, cobre-se a maior parte dos problemas que se encontram na prática.

As formas normais da 4^a em diante não têm tanto interesse prático como a FNBC sendo, habitualmente, leccionadas em tópicos mais avançados sobre Bases de Dados.

Formas normais

35

Forma normal de Boyce-Codd

- Dizemos que uma relação está na *FNBC* (Forma Normal de *Boyce-Codd*) quando: sempre que existe alguma DF não trivial de R em que $X \rightarrow Y$, X é uma super-chave.
 - Lembrar: *não trivial* significa que Y não está contido em X .
 - Lembrar: uma *super-chave* é qualquer super conjunto de uma chave (não necessariamente um super-conjunto próprio).

Formas normais

36

Exemplo

CodPostais(localidade, ruaEnº, codPostal)

DFs: localidade ruaEnº -> codPostal, codPostal -> localidade

- Quais as chaves candidatas de *CodPostais*?
- Em que forma normal se encontra esta relação?
- O que devo alterar para garantir a forma normal de Boyce-Codd?

Formas normais

37

Exemplo

Clientes(nome, morada, cervQGosta, empr, cervFav)

DFs: nome->morada cervFav; cervQGosta->empr.

- A única chave é {nome, cervQGosta}.
- Em cada DF, o lado esquerdo *não* é uma super-chave.
- Qualquer uma destas DFs mostra que *Clientes* não está na FNBC.
- *Clientes* também não está na 3^a forma normal! Porque ...
- Nem na 2^a forma normal! Porque ...
- Mas está na 1^a FN! Porque ...

Formas normais

38

Forma normal de Boyce-Codd: decomposição

- Dados de entrada: relação R com as DFs F .
- Procurar por entre as DFs dadas uma DF $X \rightarrow Y$ que viole a FNBC.
- Calcular X^+ .
 - Se der todos os atributos é porque X é uma super-chave logo, essa DF não viola a FNBC, ou seja, qualquer coisa correu mal ...
- Substituir R pelas seguintes relações:
 1. $R_1 = X^+$.
 2. $R_2 = R - (X^+ - X)$.
- **Projectar** as DFs F dadas em duas novas relações.

Formas normais

39

Forma normal de Boyce-Codd: exemplo

CodPostais(localidade, ruaEnº, codPostal)

DFs: $\text{localidade } \text{ruaEn}^0 \rightarrow \text{codPostal}$, $\text{codPostal} \rightarrow \text{localidade}$

- Verificar se há violação da FNBC em,
 $\text{localidade } \text{ruaEn}^0 \rightarrow \text{codPostal}$
 $\text{codPostal} \rightarrow \text{localidade}$.
- Calcular o fecho do lado esquerdo:
 $\{\text{codPostal}\}^+ = \{\text{localidade}, \text{codPostal}\}$.
- Relações decompostas:
 1. $R_1 = X^+$: **CodPostais1(codPostal, localidade)**
 2. $R_2 = R - (X^+ - X)$: **CodPostais2(codPostal, ruaEnº)**

Formas normais

40

Forma normal de Boyce-Codd: exemplo

- Ainda não acabou; precisamos de verificar se CodPostais1 e CodPostais2 estão na FNBC.
- Para **CodPostais1(codPostal, localidade)**, a única DF relevante é codPostal->localidade.
 - Assim, {codPostal} é a única chave e CodPostais1 está na FNBC.
- Para **CodPostais2(codPostal, ruaEnº)**, não há dependências funcionais.
 - Assim, {codPostal, ruaEnº} é a única chave e CodPostais2 está na FNBC.

Formas normais

41

Problema encontrado com a decomposição na FNBC

- A DF localidade ruaEn^o -> codPostal deixou de ser garantida.
- Para que uma DF seja assegurada, todos os seus atributos têm de estar numa mesma relação.

Formas normais

42

Garantir a FNBC pode não ser suficiente: exemplo

FNBC:

ruaEnº	codPostal
Av. da liberdade, 265	1250-144
Av. da liberdade, 265	1250-145

localidade	codPostal
Lisboa	1250-144
Lisboa	1250-145

DF: codPostal -> localidade

Juntando registos com ruaEnº iguais dá:

ruaEnº	localidade	codPostal
Av. da liberdade, 265	Lisboa	1250-144
Av. da liberdade, 265	Lisboa	1250-145

Apesar de nenhuma DF ser violada nas relações decompostas em FNBC, a DF ruaEnº localidade -> codPostal é violada pela base de dados como um todo.

Formas normais

43

Garantir a FNBC pode não ser suficiente

- Existem conjuntos de DFs que criam problemas na decomposição.
- $AB \rightarrow C$ e $C \rightarrow B$.
 - Exemplo: A = rua e nº, B = cidade, C = código postal.
- Existem duas chaves, $\{A,B\}$ e $\{A,C\}$.
- $C \rightarrow B$ é uma violação da FNBC, como tal temos de a decompor em AC , BC .

Formas normais

44

Garantir a FNBC pode não ser suficiente

- O problema é que se usarmos as relações AC e BC , não podemos garantir a DF $AB \rightarrow C$ pela verificação das DFs nas relações decompostas.
- Exemplo com $A = \text{ruaEn}^0$, $B = \text{cidade}$ e $C = \text{codPostal}$ tal como apresentado anteriormente.

Formas normais

45

Propriedades desejáveis da decomposição

- A decomposição deve ter duas propriedades importantes:
 1. *Junção sem perdas*: deverá ser possível projectar em relações decompostas, e depois reconstruir a original.
 2. *Preservação das dependências*: deverá ser possível verificar nas relações projectadas se todas as DF são satisfeitas.

Formas normais

46

Propriedades desejáveis da decomposição

- Podemos garantir *Junção sem perdas* com a decomposição FNBC.
- Podemos assegurar tanto *Junção sem perdas* como *Preservação das dependências* com a decomposição 3FN (tema a abordar já de seguida).
- Mas nunca conseguimos garantir *Junção sem perdas* e *Preservação das dependências* com a decomposição FNBC.
 - ruaEnº-localidade-codPostal é um exemplo.

Formas normais

47

Garantir a FNBC quando não há preservação das dependências

CodPostais(localidade, ruaEnº, codPostal)

DFs: localidade ruaEnº -> codPostal,
codPostal -> localidade

Neste exemplo, a decomposição na FNBC não garante a **preservação das dependências**: DF localidade ruaEnº -> codPostal

Nestes casos, a única maneira de ultrapassar o problema é redefinindo os atributos e, por inerência, as DFs.

CodPostais(localidade, ruaEnº, CP1, CP2)

DFs: CP1 -> localidade,
CP1, ruaEnº -> CP2

Exemplo: codPostal=4200-465;
Dá origem a: CP1=4200; CP2=465.

Exercício: Faça a decomposição desta relação na FNBC e verá que agora é possível decompor na FNBC garantindo a **preservação das dependências**.

Formas normais

48

Teste da caça: para aferir das junções sem perdas

- Exemplo:

Decomposição de $R(A,B,C,D)$ em $S_1(A,D)$, $S_2(A,C)$ e $S_3(B,C,D)$.

DFs: $A \rightarrow B$; $A \rightarrow C$; $CD \rightarrow A$.

- Como verificar se esta decomposição permite a *Junção sem perdas*?

Formas normais

49

Teste da caça: para aferir das junções sem perdas

Quadro inicial:

Põe-se uma linha por cada relação decomposta existente.

Coloca-se uma letra designativa do atributo por cada atributo na relação decomposta.

Coloca-se uma letra designativa do atributo com um posfixo designativo da linha por cada atributo que não esteja na relação decomposta.

Exemplo:

A	B	C	D	
a	b1	c1	d	→ S1(A,D)
a	b2	c	d2	→ S2(A,C)
a3	b	c	d	→ S3(B,C,D)

Formas normais

50

Teste da caça: para aferir das junções sem perdas

Iterações:

Sempre que o lado esquerdo de uma DF seja comum em duas linhas do quadro, substituem-se os atributos que aparecem no lado direito da DF preferencialmente por valores sem posfixo de forma a que as duas linhas com lados esquerdos iguais nessa DF fiquem também com lados direitos iguais

Exemplo: A->B; A->C; CD->A

A	B	C	D	
a	b1	c	d	→ A->C
a	b2	c	d2	
a	b	c	d	→ CD->A

Formas normais

51

Teste da caça: para aferir das junções sem perdas

Conclusão:

Se se conseguir obter uma linha sem posfixos significa que a decomposição garante junções sem perdas.

Caso todas as linhas tenham pelo menos um atributo com posfixo então a decomposição não garante que a junção seja feita sem perdas.

Exemplo: A->B; A->C; CD->A

A	B	C	D
a	b1	c	d
a	b1	c	d2
a	b	c	d

Uma linha com todos os atributos sem posfixo => junção sem perdas

Formas normais

52

Decomposição 3FN

- Podemos sempre construir uma decomposição em relações 3FN sem perdas nas junções e com preservação das dependências.
- Requere *base mínima* para as DFs:
 1. Os lados direitos têm um só atributo.
 2. Nenhuma DF pode ser removida.
 3. Nenhum atributo pode ser removido dos lados esquerdos.

Formas normais

53

Decomposição 3FN: construção da base mínima

1. DFs com lados direitos de um só atributo.
2. Repetidamente tentar remover uma DF e ver se as DFs restantes são equivalentes à original.
3. Repetidamente tentar remover um atributo do lado esquerdo e ver se as DFs resultantes são equivalentes à original.

Formas normais

54

Decomposição 3FN: síntese

- Uma relação por cada lado esquerdo diferente nas DFs na base mínima.
 - Cada relação é a união dos lados direito e esquerdo das DFs com lados esquerdos iguais.
- Se nenhuma das relações criadas tiver uma super-chave, acrescentar uma relação com uma chave da relação R original.

Formas normais

55

Decomposição 3FN: exemplo

- Relação $R = ABCD$.
- DFs $A \rightarrow B$ e $A \rightarrow C$.
- Decomposição: $R_1 = AB$ e $R_2 = AC$ a partir das DFs, mais $R_3 = AD$ para a chave.

Formas normais

56

Decomposição 3FN

Garante:

- Preservação das dependências funcionais;
- Junção sem perdas;
- 3FN.

Formas normais

57

Exemplos

1. Faça a decomposição para garantia da FNBC do exemplo:

Cientes(nome, morada, cervQGosta, empr, cervFav)

DFs: nome->morada cervFav; cervQGosta->empr.

2. Faça a decomposição 3FN para o mesmo exemplo.

Formas normais

58

Decomposição FNBC: exemplo

Cientes(nome, morada, cervQGosta, empr, cervFav)

$F = \text{nome} \rightarrow \text{morada}; \text{nome} \rightarrow \text{cervFav};$
 $\text{cervQGosta} \rightarrow \text{empr}.$

- Verificar se há violação da FNBC em, nome \rightarrow morada.
- Calcular o fecho do lado esquerdo: $\{\text{nome}\}^+ = \{\text{nome}, \text{morada}, \text{cervFav}\}$.
- Relações decompostas:
 1. **Cientes1(nome, morada, cervFav)**
 2. **Cientes2(nome, cervQGosta, empr)**

Formas normais

59

Decomposição FNBC: exemplo

- Ainda não acabou; precisamos de verificar se Clientes1 e Clientes2 estão na FNBC.
- Para este caso a projecção das DFs é fácil.
- Para **Clientes1(nome, morada, cervFav)**, as DFs relevantes são nome->morada e nome->cervFav.
 - Assim, {nome} é a única chave e Clientes1 está na FNBC.

Formas normais

60

Decomposição FNBC: exemplo

- Para $\text{Clientes2}(\underline{\text{nome}}, \underline{\text{cervQGosta}}, \text{empr})$, a única DF é $\text{cervQGosta} \rightarrow \text{empr}$, e a única chave é $\{\text{nome}, \text{cervQGosta}\}$.
 - Violação da FNBC.
- $\text{cervQGosta}^+ = \{\text{cervQGosta}, \text{empr}\}$, por isso decomponemos Clientes2 em:
 1. $\text{Clientes3}(\underline{\text{cervQGosta}}, \text{empr})$, com a DF $\text{cervQGosta} \rightarrow \text{empr}$
 2. $\text{Clientes4}(\underline{\text{nome}}, \underline{\text{cervQGosta}})$, sem DFs

Formas normais

61

Decomposição FNBC: exemplo

- A decomposição resultante de *Clientes*:
 1. *Clientes1*(nome, morada, cervFav)
 2. *Clientes3*(cervQGosta, empr)
 3. *Clientes4*(nome, cervQGosta)
- Nota: *Clientes1* é sobre clientes, *Clientes3* sobre cervejas, e *Clientes4* sobre a relação entre os clientes e as cervejas de que eles gostam.

Formas normais

62

Decomposição 3FN: exemplo

Clientes(nome, morada, cervQGosta, empr, cervFav)

$F = \text{nome} \rightarrow \text{morada}$, $\text{nome} \rightarrow \text{cervFav}$, $\text{cervQGosta} \rightarrow \text{empr}$

1. DFs com lados direitos de um só atributo: **ok**.
2. Repetidamente tentar remover uma DF e ver se as DFs restantes são equivalentes à original: **ok**.
3. Repetidamente tentar remover um atributo do lado esquerdo e ver se as DFs resultantes são equivalentes à original: **ok**.
4. Uma relação por cada lado esquerdo diferente nas DFs:
 - a) **Clientes1(nome, morada, cervFav)**, com as DFs $\text{nome} \rightarrow \text{morada}$ e $\text{nome} \rightarrow \text{cervFav}$.
 - b) **Clientes2(cervQGosta, empr)**, com a DF $\text{cervQGosta} \rightarrow \text{empr}$.
5. Se nenhuma das relações criadas tiver uma super-chave, acrescentar uma relação com uma chave da relação R original:
 - c) **Clientes3(nome, cervQGosta)**, sem DFs.



Universidade do Porto

Faculdade de Engenharia

FEUP

Bases de Dados

1

- **INTRODUÇÃO**
- **MODELOS CONCEPTUAIS**
 - Diagrama de Classes UML
 - Modelo Entidade-Associação (E-A)
- **MODELO RELACIONAL**
- **LINGUAGEM DE DEFINIÇÃO DE DADOS**
- **INTERROGAÇÃO DE DADOS**
 - Álgebra relacional
 - Linguagem de Manipulação de Dados (LMD)

Observação: baseado em slides desenvolvidos pelo Prof. Jeffrey D. Ullman

João Mendes Moreira

FEUP

Índice

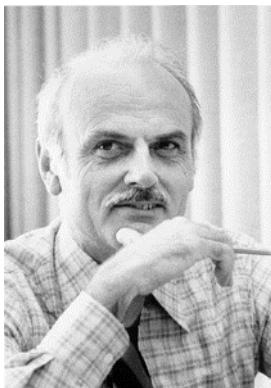
2

- História da SQL.
- Linguagem de definição de dados
- CREATE TABLE
- Restrições
 - Chaves
 - Chaves externas
 - Restrições de valor
 - Restrições entre atributos
 - Asserções
- Modelação UML das restrições
- Gatilhos
- Instruções da LDD SQL

História da SQL

3

1. Publicação do artigo "A Relational Model of Data for Large Shared Data Banks" por Edgar F. Codd na revista *Communications of the ACM*, em Junho de 1970.



Edgar F. Codd

2. Desenvolvimento da linguagem SEQUEL, mais tarde designada SQL, na IBM, por Donald D. Chamberlin e Raymond F. Boyce.

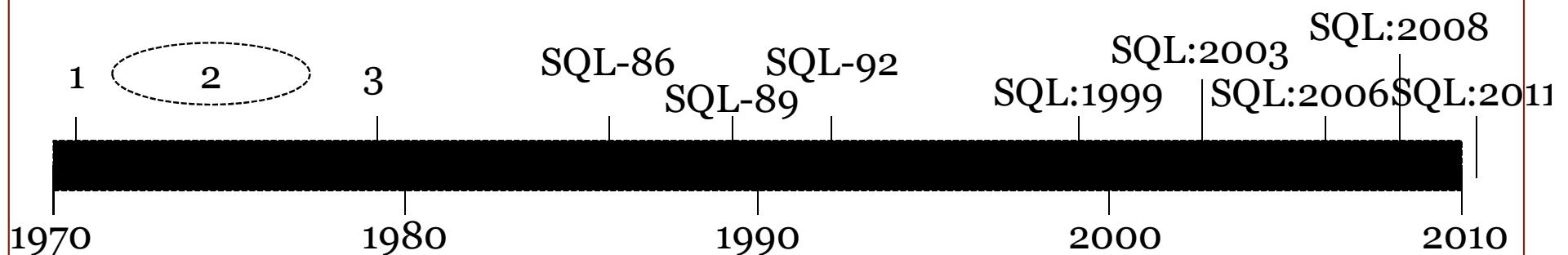


Donald D.
Chamberlin



Raymond F. Boyce

3. Aparecimento da primeira versão comercial por parte da *Relational Software* (agora Oracle), em 1979.



Linguagem de Definição de Dados

4

- A linguagem SQL (*Structured Query Language*) tem duas componentes:
 - Linguagem de Definição de Dados (LDD);
 - Linguagem de Manipulação de Dados (LMD).
- A Linguagem de Definição de Dados (LDD) é uma linguagem de programação para definição de estruturas de dados.
- A Linguagem de Manipulação de Dados (LMD) é uma linguagem de programação para manipulação de dados.
- Vamos estudar agora a LDD SQL.
- Mais tarde estudaremos a LMD SQL.

Linguagem de Definição de Dados

5

A LDD permite:

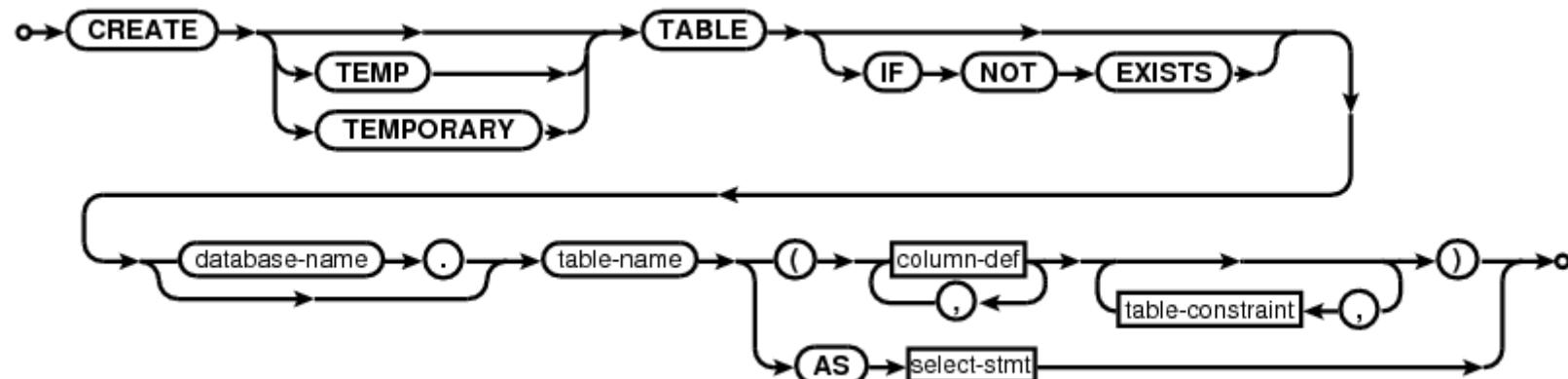
- Criar, alterar e remover estruturas de dados da base de dados
- Definir a política de controlo de acessos
- Optimizar a base de dados com vistas, índices, ...
- Desenvolver funções e procedimentos através de uma linguagem procedural própria (PSM: Persistent Stored Modules)
- Definir a política de cópias de degurança e de recuperação
- Definir opções de administração a um nível mais baixo (armazenamento em disco, ...)
- Auditar a base de dados
- E mais algumas coisas ...

CREATE TABLE

6

Create relational tables using SQLite:

http://www.sqlite.org/lang_createtable.html



CREATE TABLE

7

Exemplo

Pessoa
nome: NVARCHAR2(60) data de nascimento: DATE peso: NUMBER(3,2) = 75

Classe do diagrama de classes UML

```
CREATE TABLE Pessoa (
    idPessoa           NUMBER PRIMARY KEY,
    nome                NVARCHAR2( 20 ) ,
    dataNascimento      DATE ,
    peso                NUMBER DEFAULT 75 );
```

CREATE TABLE

8

Tipos de dados mais comuns (ORACLE)

NVARCHAR2(size [BYTE | CHAR]): *Variable-length national character string having maximum length size bytes or characters. Maximum size is 4000 bytes or characters, and minimum is 1 byte or 1 character.*

CHAR [(size [BYTE | CHAR])]: *Fixed-length character data of length size bytes or characters. Maximum size is 2000 bytes or characters. Default and minimum size is 1 byte.*

NUMBER[(p [, s])]: *Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127. Both precision and scale are in decimal digits. A NUMBER value requires from 1 to 22 bytes.*

DATE: Valid date range from January 1, 4712 BC, to December 31, 9999 AD. The size is fixed at 7 bytes. [...] This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

TIMESTAMP [(fractional_seconds_precision)]: Year, month, and day values of date, as well as hour, minute, and second values of time, where fractional_seconds_precision is the number of digits in the fractional part of the SECOND datetime field. Accepted values of fractional_seconds_precision are 0 to 9. [...] This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. [...]

RAW(size): Raw binary data of length size bytes. Maximum size is 2000 bytes. [...]

LONG RAW: Raw binary data of variable length up to 2 gigabytes.

BLOB: A binary large object. Maximum size is (4 gigabytes - 1) *(database block size).

NOTA: Estes são alguns dos tipos de dados ORACLE mais comuns. Mas há mais:

http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/datatype.htm

CREATE TABLE

9

Tipos de dados (SQLite)

O SQLite é compatível com os tipos de dados usados pelos Sistemas Gestores de Bases de Dados mais comuns. Ex.: Oracle, MySQL, etc.

No entanto o SQLite usa o conceito de **Afinidade (Affinity)**.

Converte os tipos de dados mais comuns numa das cinco afinidades existentes de acordo com o quadro ao lado e pela ordem indicada.

Example Typenames From The CREATE TABLE Statement or CAST Expression	Resulting Affinity	Rule Used To Determine Affinity
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	1
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	2
BLOB <i>no datatype specified</i>	NONE	3
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL	4
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	5

Restrições e gatilhos

10

- Uma *restrição* é uma condição que é imposta aos dados.
 - Exemplo: chaves.
- *Gatilhos define uma acção a executar quando ocorre um determinado tipo de evento.*
 - Exemplo: actualizar o número de pontos que um piloto tem no campeonato de F1 depois de inserir a classificação numa corrida.
 - São mais fáceis de implementar do que restrições complexas.

Restrições

11

Tipos de restrições

- *Chaves.*
- *Chaves-externas*, ou integridade referencial.
- Restrições de *valor*.
 - Limites ao valor de um certo atributo (ex.: preço > 0).
 - NOT NULL
- Restrições *entre atributos*.
 - Associações entre diferentes atributos (ex.: dataFim > dataInicio).
- *Asserções*: qualquer expressão SQL booleana.

Chaves

12

Chaves de um só atributo

- Pôr PRIMARY KEY ou UNIQUE depois do tipo de dados na declaração do atributo.
 - Só pode haver uma chave primária (primary key) por tabela, mas podem haver várias chaves (unique).
 - Os atributos da chave primária não podem ter valores NULL, mas alguns dos atributos da restrição UNIQUE podem ser NULL desde que a combinação desses atributos seja única.
- Exemplo:

```
CREATE TABLE Cervejas (
    nome CHAR( 20 ) PRIMARY KEY ,
    empr CHAR( 20 ) );
```

Chaves

13

Chaves multi-atributo

- Bar e cerveja juntos são a chave primária de Vende:

```
CREATE TABLE Vende (
    bar        CHAR( 20 ) ,
    cerveja   NVARCHAR( 20 ) ,
    preco      NUMBER ,
PRIMARY KEY (bar, cerveja)
) ;
```

Chaves externas

14

- Valores que aparecem em atributos de uma relação devem também aparecer em certos atributos de outra relação.
- Exemplo: em `Vende(bar, cerveja, preço)`, espera-se que o valor de `cerveja` também apareça em `Cervejas.nome`.

Chaves externas

15

- Utilizar a palavra chave REFERENCES em qualquer uma das seguintes situações:
 1. Depois de um atributo (a seguir a uma chave externa com um só atributo).
 2. Como elemento do esquema:
FOREIGN KEY (<lista de atributos>)
REFERENCES <relação> (<atributos>)
- Atributos referenciados devem ser declarados como PRIMARY KEY ou UNIQUE.

Chaves externas

16

Exemplos

```
CREATE TABLE Cervejas (
    nome      CHAR( 20 ) PRIMARY KEY ,
    empr      CHAR( 20 ) );
```

```
CREATE TABLE Vende (
    bar      CHAR( 20 ) ,
    cerveja CHAR(20) REFERENCES Cervejas(nome) ,
    preço     NUMBER );
```

Chaves externas

17

Exemplos

```
CREATE TABLE Cervejas (
    nome      CHAR(20) PRIMARY KEY,
    empr      CHAR(20) );
```

```
CREATE TABLE Vende (
    bar       CHAR(20),
    cerveja  CHAR(20),
    preço     NUMBER,
    FOREIGN KEY(cerveja) REFERENCES
Cervejas(nome) );
```

Chaves externas

18

Restrições de chave externa

- Se houver uma restrição de chave-externa da relação R para a relação S , podem acontecer 2 violações:
 1. Um INSERT ou UPDATE de R introduz valores que não existem em S .
 2. Um DELETE ou UPDATE de S faz com que alguns registos de R “fiquem pendurados”.

Chaves externas

19

Restrições de chave externa

- Exemplo: vamos supor que $R = \text{Vende}$, $S = \text{Cervejas}$.
- Um INSERT ou UPDATE de Vende que introduza uma cerveja não existente deve ser rejeitado.
- Um DELETE ou UPDATE de Cervejas que remove uma cerveja existente nos registos de Vende pode ser resolvido de 3 maneiras diferentes (ver slide seguinte).

Chaves externas

20

Restrições de chave externa

Se ao apagar ou alterar o atributo nome da tabela Cervejas existirem registo(s) na tabela Vende com referência aos registo(s) que se quer apagar/alterar:

1. *Por defeito* : Rejeita a modificação.
2. *Em cascata*: Faz as mesmas alterações em Vende.
 - Cerveja removida: apagar registo(s) de Vende.
 - Cerveja actualizada: alterar valor de Vende.
3. *Atribuir NULL* : Alterar a cerveja para NULL.

Chaves externas

21

Restrições de chave externa: em cascata

- Apaga o registo Abadia da tabela Cervejas:
 - Depois apaga todos os registos de Vende com cerveja = 'Abadia'.
- Altera o registo Abadia da tabela Cervejas substituindo 'Abadia' por 'Super Bock Abadia':
 - Depois altera os registos de Vende com cerveja = 'Abadia' para cerveja = 'Super Bock Abadia'.

Chaves externas

22

Restrições de chave externa: atribuir NULL

- Apagar o registo Abadia de Cervejas:
 - Mudar todos os registos de Vende que tenham cerveja = 'Abadia' para cerveja = NULL.
- Alterar o registo Abadia de Cervejas substituindo 'Abadia' por 'Super Bock Abadia':
 - A mesma alteração que para o apagar.

Chaves externas

23

Restrições de chave externa: escolha de política

- Quando declaramos uma chave externa, podemos escolher como políticas SET NULL ou CASCADE independentemente para DELETEs e UPDATEs.
- A seguir à declaração de chave externa pôr:
ON [UPDATE, DELETE][SET NULL | CASCADE]
- Podem ser usadas estas duas cláusulas.
- Caso contrário, é usada a acção de rejeição por defeito.

Chaves externas

24

Restrições de chave externa: escolha de política

Exemplo :

```
CREATE TABLE Vende(
    bar      CHAR( 20 ) ,
    cerveja CHAR( 20 ) ,
    preco    NUMBER ,
    FOREIGN KEY(cerveja)
        REFERENCES Cervejas(nome)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ;
```

Restrições de valor

25

- Restrições ao valor de um dado atributo.
- Acrescentar CHECK(<condição>) à declaração do atributo.
- A condição pode usar o nome do atributo, mas **o nome de qualquer outra relação ou atributo de outra relação tem de estar numa subconsulta (*subquery*)**.

Restrições de valor

26

Exemplo

```
CREATE TABLE Vende (
    bar    CHAR( 20 ) NOT NULL ,
    cerveja CHAR( 20 )
        CHECK (cerveja IN
            (SELECT nome FROM Cerveja)) ,
    preco   NUMBER CHECK (preco <= 5.00) );
```

Sub-consultas em restrições do tipo CHECK não são possíveis em SQLite.

Restrições de valor

27

Momento de realização dos testes

- Os testes aos atributos só são realizados quando o valor do atributo é inserido ou actualizado.
 - Exemplo: CHECK (preco <= 5.00) testa todos os preços novos e rejeita a modificação (para esse registo) se o preço for superior a 5 €.
 - Exemplo: CHECK (cerveja IN (SELECT nome FROM Cervejas)) não é testado se a cerveja for apagada de Cervejas (ao contrário das chaves-externas).

Restrições entre atributos

28

- CHECK (<condição>) pode ser acrescentado.
- A condição pode-se referir a qualquer atributo da relação.
 - Mas quaisquer outros atributos ou relações requerem uma subconsulta.
- Os testes só são efectuados aquando das inserções ou actualizações.

Restrições entre atributos

29

Exemplo

- Só o bar Mercedes vende cerveja por mais de 5 €:

```
CREATE TABLE Vende(  
    bar      CHAR( 20 ) ,  
    cerveja CHAR( 20 ) ,  
    preco   NUMBER ,  
    CHECK (bar = 'Mercedes' OR  
        preco <= 5.00) );
```

Asserções

30

- São elementos das bases de dados, da mesma forma que as relações o são.
- Define-se da seguinte forma:

```
CREATE ASSERTION <nome>
    CHECK (<condição>);
```

- A condição pode-se referir a qualquer relação ou atributo da base de dados.
- No entanto a grande maioria dos Sistemas Gestores de Bases de Dados comerciais não têm esta funcionalidade implementada. O SQLite também não tem.

Asserções

31

Exemplo

- Em **Vende(bar, cerveja, preco)**, nenhum bar pode cobrar em média mais de 5 €.

```
CREATE ASSERTION NoBaresCaros CHECK (
    NOT EXISTS (
```

```
        SELECT bar FROM Sells
        GROUP BY bar
        HAVING AVG(preco) > 5
    ));
```

Bares com preços médios acima de 5 €

Asserções

32

Exemplo

- Em **Cientes(nome, morada, telefone)** e **Bares(nome, morada, licenca)**, não pode haver mais bares do que clientes.

```
CREATE ASSERTION PoucosBares CHECK (
    (SELECT COUNT(*) FROM Bares) <=
    (SELECT COUNT(*) FROM Clientes)
);
```

Asserções

33

Momento de realização dos testes

- Em princípio, temos de testar todas as asserções depois de qualquer modificação a uma relação da base de dados.
- Um sistema inteligente deve ter em conta que só algumas mudanças podem originar violações das asserções.
 - Exemplo: Nenhuma alteração a Cervejas afecta PoucosBares. Nem nenhuma inserção a Clientes.

UML: Diagrama de Classes

34

Restrições

- Uma restrição especifica uma condição que tem de se verificar no estado do sistema (objectos e ligações)
- Uma restrição é indicada por uma expressão ou texto entre chavetas ou por uma nota posicionada junto aos elementos a que diz respeito, ou a eles ligada por linhas a traço interrompido (sem setas, para não confundir com relação de dependência)
- Podem ser formalizadas em UML com a OCL - "Object Constraint Language"
- Também podem ser formalizadas (como invariantes) numa linguagem de especificação formal como VDM++

Modelação UML das restrições

35

Classes

```
CREATE TABLE Pessoa (
    idPessoa      NUMBER PRIMARY KEY,
    nome          NVARCHAR2( 20 ) ,
    dataNascimento DATE ,
    localNascimento NUMBER ,
    dataFalecimento DATE ,
    UNIQUE ( nome, dataNascimento, localNascimento ),
    CHECK ( dataFalecimento > dataNascimento)
);
```

Pessoa

nome
dataNascimento
localNascimento
dataFalecimento

{chave candidata: (nome, dataNascimento, localNascimento)}

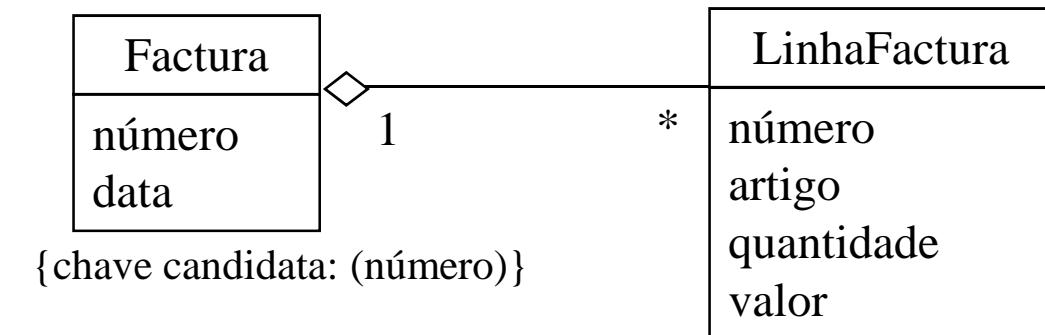
{dataFalecimento > dataNascimento}

Modelação UML das restrições

36

Chaves candidatas e chaves externas

```
CREATE TABLE Factura (
    numero NUMBER PRIMARY KEY,
    data      DATE);
```



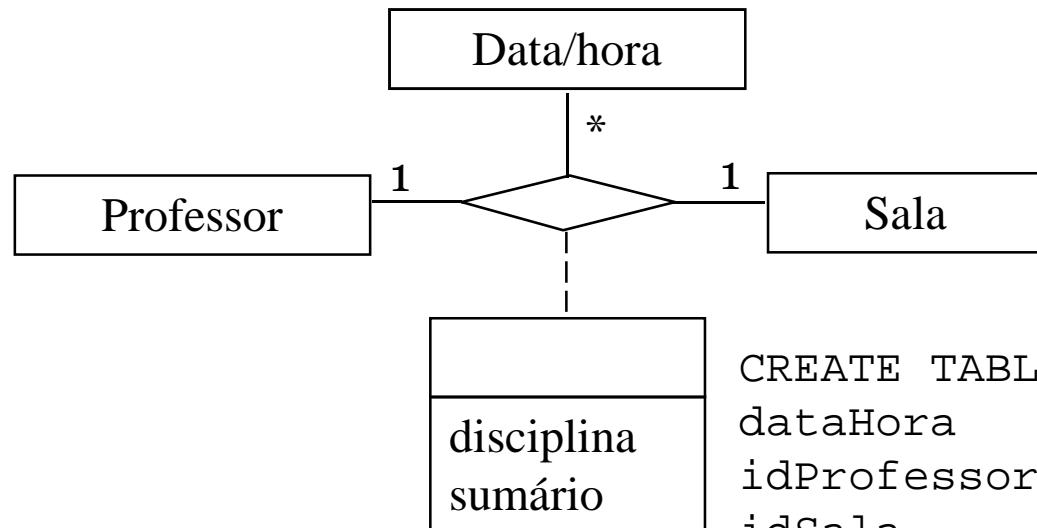
```
CREATE TABLE LinhaFactura (
    idLinhaFactura NUMBER PRIMARY KEY,
    numero  NUMBER,
    artigo   NVARCHAR2(20),
    quantidade NUMBER,
    valor    NUMBER,
    numFactura NUMBER REFERENCES Factura(numero),
    UNIQUE (numFactura, numero));
```

{chave candidata: (factura, número)}

Modelação UML das restrições

37

Associações ternárias



```
CREATE TABLE aula (
    dataHora      TIMESTAMP,
    idProfessor  NUMBER,
    idSala        NUMBER,
    disciplina    NVARCHAR2(20),
    sumario       NVARCHAR2(100),
    PRIMARY KEY (dataHora, idProfessor),
    UNIQUE (dataHora, idSala)
);
```

Gatilhos

38

Motivação

- As asserções são poderosas, mas os SGBD não costumam ser capazes de dizer quando é que o teste deve ser feito.
- Testes a atributos e a tuplos são efetuados em momentos conhecidos, mas não são poderosos.
- Os gatilhos deixam o responsável pela BD decidir quando é que deve ser efetuado o teste a uma dada condição.

Gatilhos

39

Regras acontecimento-condição-acção

- Um gatilho pode ser visto como uma regra *acontecimento-condição-acção*.
- *Acontecimento* : são tipos de modificações que acontecem numa BD, ex.: inserir em Vende.
- *Condição* : Qualquer expressão SQL cujo resultado seja do tipo Booleano.
- *Acção* : Qualquer comando SQL.

Gatilhos

40

Exemplo

- Em vez de usar uma restrição de chave-externa e rejeitar inserções em **Vende(bar, cerveja, preco)** com cervejas desconhecidas, um gatilho pode adicionar a cerveja a Cervejas, atribuindo o valor NULL a empresa.

Gatilhos

41

Exemplo

Este exemplo não
funciona em SQLite.

CREATE TRIGGER GatCerveja

AFTER INSERT ON Vende

O acontecimento

REFERENCING NEW ROW AS NovoTuplo

FOR EACH ROW

WHEN (NovoTuplo.cerveja NOT IN
(SELECT nome FROM Cervejas))

A condição

INSERT INTO Cervejas(nome)

VALUES(NovoTuplo.cerveja);

A acção

Gatilhos

42

CREATE TRIGGER

- CREATE TRIGGER <nome>
- Ou:

CREATE OR REPLACE TRIGGER <nome>

- Útil no caso de existir um gatilho com esse nome e se queira modificá-lo.

Gatilhos

43

O acontecimento

- Em vez de AFTER pode ser BEFORE.
 - Ou, INSTEAD OF, se a relação for uma vista.
 - ✖ Uma forma inteligente de executar alterações a vistas: substitui a ação por modificações adequadas nas tabelas que participam na vista.
- INSERT pode ser DELETE ou UPDATE.
 - E UPDATE pode ser UPDATE OF ... ON um dado atributo.

Gatilhos

44

FOR EACH ROW

- Os gatilhos podem ser definidos ao “nível-de-linha” ou ao “nível-de-instrução”.
- FOR EACH ROW indica nível-de-linha; a sua omissão indica nível-de-instrução.
- *Gatilhos ao nível-de-linha*: executa uma vez por cada linha modificada.
- *Gatilhos ao nível-de-instrução*: executa uma vez para uma instrução SQL, independentemente do número de linhas modificadas.

Gatilhos

45

REFERENCING

- As instruções INSERT implicam um novo registo (para o nível-de-linha) ou uma nova “tabela” (para o nível-de-instrução).
 - A “tabela” é o conjunto de novas linhas.
- DELETE implica um registo ou “tabela” velhos.
- UPDATE implica ambos (novo e velho).
- A referência é feita da seguinte forma:
NEW ou OLD antes do respectivo atributo.

Gatilhos

46

A condição

- Qualquer expressão SQL com resultado Booleano.
- Avaliação efectuada na base de dados de acordo com o seu estado antes ou depois do evento do gatilho, dependendo da utilização de BEFORE ou AFTER.
 - **Mas sempre antes das alterações serem efectuadas.**
- Acede ao novo/velho registo/tabela através dos nomes NEW e OLD.

Gatilhos

47

A acção

- Pode existir mais do que uma instrução SQL statement na acção.
 - Limitada por BEGIN . . . END caso existam várias.
- Mas como as consultas (SELECT) não fazem sentido numa acção, estamos limitados a INSERT, UPDATE, e/ou DELETE.

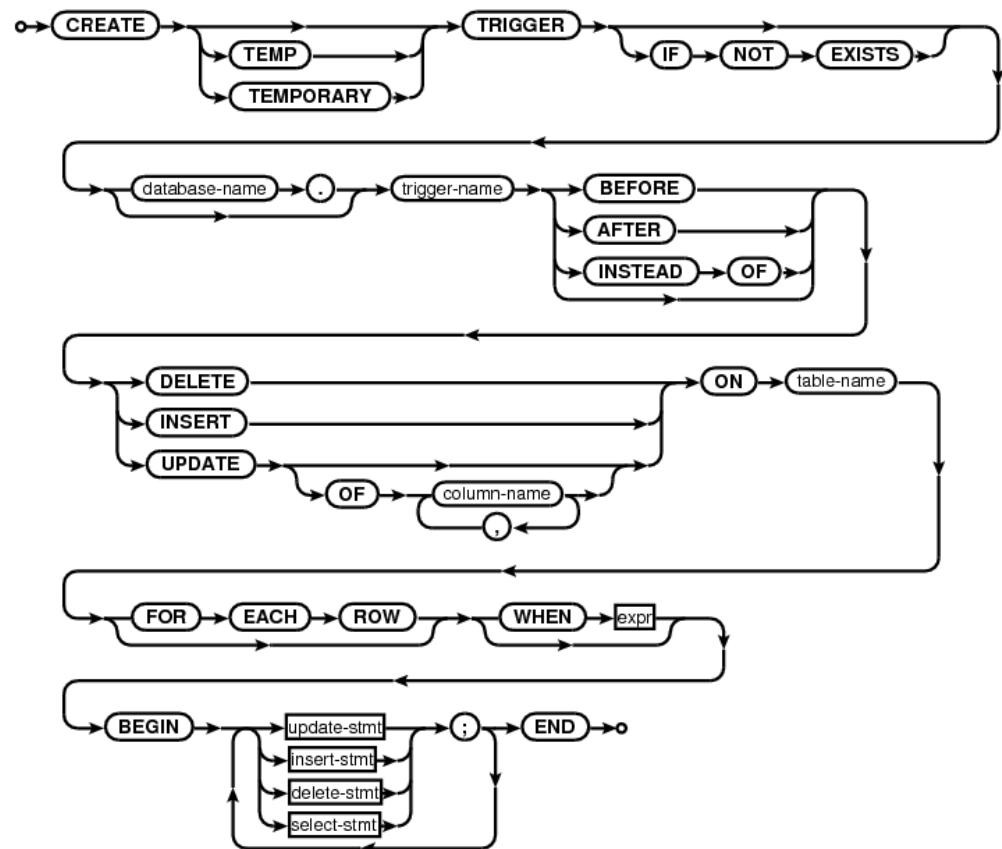
Gatilhos

48

Usando SQLite

A sintaxe dos TRIGGERS em SQLite tem algumas diferenças em relação à sintaxe dos standars.

Observe essas diferenças no exemplo que se segue.



Gatilhos

49

Exemplo

- Usando `Vende(bar, cerveja, preco)` e a relação unária `BaresCaros(bar)`, manter a lista dos bares que aumentam o preço de alguma cerveja em mais de 1 €.

Gatilhos

50

Exemplo

CREATE TRIGGER GatilhoPreco

AFTER UPDATE OF preco ON Vende

← O evento: apenas mudanças de preço

FOR EACH ROW

← Temos de considerar a mudança de cada preço

WHEN(NEW.preco > OLD.preco + 1.00)

← Condição: acréscimo de preço > 1€

BEGIN

INSERT INTO BaresCaros
VALUES(NEW.bar);

← Quando a mudança de preço é suficientemente grande, acrescentar o bar a BaresCaros

END;

InSTRUÇÕES DA LDD SQL

51

```
CREATE TABLE IF NOT EXISTS Cervejas (
    idCerveja INTEGER AUTOINCREMENT,
    nome      CHAR(20),
    empr     CHAR(20),
CONSTRAINT pk_Cervejas PRIMARY KEY (idCerveja));
CREATE TABLE IF NOT EXISTS Vende (
    idVenda    INTEGER AUTOINCREMENT,
    bar        CHAR(20),
    idCerveja NUMBER,
    preco      NUMBER,
CONSTRAINT pk_Vende PRIMARY KEY (idVenda),
CONSTRAINT k1_Vende UNIQUE (bar,idCerveja),
CONSTRAINT fk_Vende_cerveja FOREIGN KEY (idCerveja) REFERENCES Cervejas (idCerveja));
```

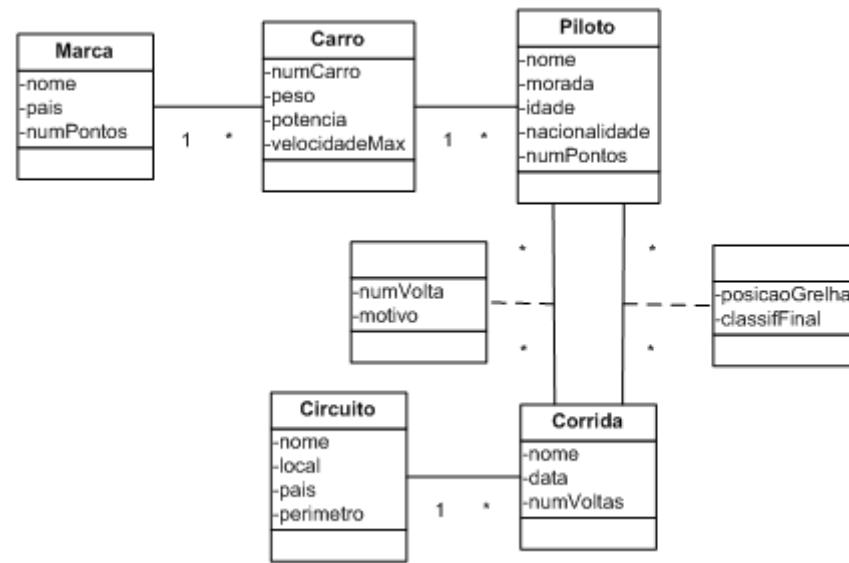
Criação da BD

```
INSERT INTO Cervejas (nome, empr)
VALUES ('Abadia', 'UNICER');
INSERT INTO Vende (bar, idCerveja, preco)
VALUES ('Pipa velha', 1, 2.5);
```

Inserção dos dados

Exercício

52



Bases de Dados



Universidade do Porto

Faculdade de Engenharia

FEUP

1

- INTRODUÇÃO
- MODELOS CONCEPTUAIS
 - Diagrama de Classes UML
 - Modelo Entidade-Associação (E-A)
- MODELO RELACIONAL
- LINGUAGEM DE DEFINIÇÃO DE DADOS
- INTERROGAÇÃO DE DADOS
 - ÁLGEBRA RELACIONAL
 - Linguagem de Manipulação de Dados (LMD)

Observação: baseado em slides desenvolvidos pelo Prof. Gabriel David da FEUP

João Mendes Moreira

FEUP

Índice

2

- Linguagens relacionais
- Operações da álgebra relacional
- Leis algébricas
- Linguagem de interrogação
- Extensões à álgebra relacional

Linguagens relacionais

3

- Notações para expressar perguntas:
 - Algébrica – aplicação de operadores a relações
 - Lógica – fórmula que os tuplos da resposta devem satisfazer
- Limitações da álgebra relacional:
 - independência física dos dados
 - ✖ linguagem só com construções relativas ao modelo de dados (operações sobre relações) que não dependam da implementação
 - optimização das perguntas
 - ✖ restrição no poder expressivo: sem recursividade (incapaz de computar o fecho transitivo)
 - relações finitas
 - ✖ Noção de complementar é proibida

Operações da álgebra relacional

4

1 - Reunião $R \cup S$

Reunião

- é o conjunto dos tuplos que estão em R, em S, ou em ambas
 - R e S da mesma aridade (mesma aridade = mesmo nº de atributos)
 - Domínios dos atributos de R e S devem ser compatíveis:
 - Domínios do 1º atributo de R e S devem ser compatíveis
 - Domínios do 2º atributo de R e S devem ser compatíveis
 - etc.

R		
A	B	C
a	b	c
d	a	f
c	b	d

S		
D	E	F
b	g	a
d	a	f

R \cup S		
A	B	C
a	b	c
d	a	f
c	b	d
b	g	a

Operações da álgebra relacional

5

Intersecção

2 - Intersecção $R \cap S$

- contém os tuplos que pertencem a R e a S simultaneamente
 - ✖ R e S da mesma aridade
 - ✖ R e S com domínios compatíveis
- $R \cap S = R - (R - S)$

R	A	B	C
a	b	c	
d	a	f	
c	b	d	

S	D	E	F
b	g	a	
d	a	f	

$R \cap S$	A	B	C
d	a	f	

Operações da álgebra relacional

6

Diferença

3 - Diferença $R - S$

- tuplos de R que não estão em S
 - ✖ R e S da mesma aridade
 - ✖ R e S com domínios compatíveis

R	A	B	C
	a	b	c
	d	a	f
	c	b	d

S	D	E	F
	b	g	a
	d	a	f

R - S	A	B	C
	a	b	c
	c	b	d

Operações da álgebra relacional

7

Produto cartesiano

4 - Produto cartesiano $R \times S$

- aridades de R e S são k_1 e $k_2 \Rightarrow$ aridade de $R \times S$ é $k_1 + k_2$
 - ▶ contém todos os tuplos tais que os primeiros k_1 componentes formam um tuplo de R e os restantes k_2 componentes formam um tuplo em S

R	A	B	C
	a	b	c
	d	a	f
	c	b	d

S	D	E	F
	b	g	a
	d	a	f

$R \times S$	A	B	C	D	E	F
	a	b	c	b	g	a
	a	b	c	d	a	f
	d	a	f	b	g	a
	d	a	f	d	a	f
	c	b	d	b	g	a
	c	b	d	d	a	f

Operações da álgebra relacional

8

Projecção

5 - Projecção $\Pi_{i_1, i_2, \dots, i_m}(R)$

- para cada tuplo em R existe um tuplo na projecção com os componentes (e pela ordem) indicados pelos i_j
 - ✖ se aridade de R for k então os $i_j \in 1, \dots, k$ são distintos; aridade da projecção é m
 - ✖ números de componentes podem ser substituídos por atributos, se existirem: $\Pi_{1,3}(R) = \Pi_{A,C}(R)$

A	B	C
a	b	c
d	a	f
c	b	d

A	C
a	c
d	f
c	d

Projecção escolhe colunas da tabela

Operações da álgebra relacional

9

Selecção

6 - Selecção $\sigma_F(R)$

- contém os tuplos de R que satisfazem F
 - ✖ a fórmula F pode envolver
 - ⇒ operandos constantes ou número de componente (\$i)
 - ⇒ operadores aritméticos de comparação (<, =, >, ≤, ≠, ≥)
 - ⇒ operadores lógicos (\wedge , \vee , \neg) (e, ou, não)
 - ✖ Números de componentes podem ser substituídos por atributos, se existirem: $\sigma_{\$2=b}(R) = \sigma_{B=b}(R)$

A	B	C
a	b	c
d	a	f
c	b	d

$\sigma_{\$2=b}(R)$

A	B	C
a	b	c
c	b	d

Selecção escolhe linhas da tabela

Operações da álgebra relacional

10

Quociente

7 - Quociente R / S

- aridade de R é r e de S é s, $r > s$, $S \neq \emptyset$
- contém os $(r-s)$ -tuplos (a_1, \dots, a_{r-s}) tais que, para **todos** os s-tuplos (a_{r-s+1}, \dots, a_r) em S, o tuplo (a_1, \dots, a_r) está em R

R	S	R/S
A B C D	E F	A B
a b c d	c d	a b
a b e f	e f	e d
b c e f		
e d c d		
e d e f		
a b d e		

\Rightarrow

R/S	S	$(R/S) \times S$
A B	E F	A B E F
a b	c d	a b c d
e d	e f	a b e f
		e d c d
		e d e f

$2 \times 3 + 1 = 7$

7 | 3
1 | 2

Operações da álgebra relacional

11

Quociente: explicação alternativa

- Forma de proceder à divisão
 - reordenar as colunas de forma a que as últimas correspondam ao divisor
 - ordenar a tabela pelas primeiras colunas
 - cada subtuplo das primeiras colunas pertence ao resultado se o conjunto de subtuplos das últimas colunas que lhe corresponde contiver o divisor

R	S	R/S
A	E	A
B	F	B
a	c	a
b	d	b
	e	
	f	
c	c	
d	d	
e	e	
f	f	
b	e	
c	f	
e	c	
d	d	
	e	
	f	

/ =

Operações da álgebra relacional

12

Quociente: expressão

R	A	B	C	D
a	b	c	d	
a	b	e	f	
b	c	e	f	
e	d	c	d	
e	d	e	f	
a	b	d	e	

S	E	F
	c	d
	e	f

T	A	B
a	b	
b	c	
e	d	

TxS	A	B	E	F
	a	b	c	d
	a	b	e	f
	b	c	c	d
	b	c	e	f
	e	d	c	d
	e	d	e	f

- $T = \Pi_{1, \dots, r-s} (R)$ = universo dos tuplos possíveis no resultado
- $W = (T \times S) - R$ = todas as linhas T combinadas com S mas que não estão em R, i.e., em que a condição falha
- $V = \Pi_{1, \dots, r-s} (W)$ = tuplos que não interessam
- $R / S = T - V$ = tuplos que interessam
 - ✖ reunindo numa só expressão algébrica

$$R / S = \Pi_{1, \dots, r-s} (R) - \Pi_{1, \dots, r-s} [(\Pi_{1, \dots, r-s} (R) \times S) - R]$$

W	A	B	E	F
	b	c	c	d

V	A	B
	b	c

R/S	A	B
	a	b
	e	d

Operações da álgebra relacional

13

Exercícios

Quais as frases verdadeiras?

- a - $(R-S) \cup S = R$
- b - $(R-S) \cup S \supseteq R$
- c - $(R-S) \cup (R \cap S) = R$
- d - $(R-S) \cup (S-R) = (R \cup S) - (R \cap S)$
- e - $(R/S) \times S = R$
- f - $(R/S) \times S \subseteq R$

- **Respostas**

- **Erradas - a, e**
- **Correctas - b, c, d, f**

Operações da álgebra relacional

14

8 - θ -Junção $R \bowtie_{i\theta j} S$ Junção

- se a aridade de R for r e a de S s, a aridade da θ -junção é r+s
- contém os tuplos do produto cartesiano de R por S tais que o componente i está na relação θ com o componente r+j (i.e., o correspondente ao j em S).
- Expressão da θ -junção
 - $R \bowtie_{i\theta j} S = \sigma_{\$i \theta \$r+j} (R \times S)$
 - se θ for $=$, a operação designa-se equijunção
 - $(7, 8, 9)$ é um tuplo pendente de R pois não aparece na θ -junção

R			S	
A	B	C	D	E
1	2	3	3	1
4	5	6	6	2
7	8	9		

$$R \bowtie S = R \bowtie S$$

$2 < 1$ $B < D$

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

Operações da álgebra relacional

15

Junção natural

9 - Junção natural $R \bowtie S$

- só é aplicável se os componentes dos tuplos em R e S forem designados por atributos.
- a operação implícita na junção natural é a igualdade dos atributos com o mesmo nome.
- cada par de atributos iguais dá origem a um único atributo, com o mesmo nome, no resultado
- expressão:
$$R \bowtie S = \Pi_{i_1, \dots, i_m} (\sigma_{R.A_1 = S.A_1 \wedge \dots \wedge R.A_k = S.A_k} (R \times S))$$
- k é o número de atributos comuns a R (aridade r) e S (aridade s) e m= r+s-k

R			S		
A	B	C	B	C	D
a	b	c	b	c	d
d	b	c	b	c	e
b	b	f	a	d	b
c	a	d			

R \bowtie S			
A	B	C	D
a	b	c	d
a	b	c	e
d	b	c	d
d	b	c	e
c	a	d	b

Operações da álgebra relacional

16

Junção externa

- tuplos pendentes, isto é desemparelhados, quer em R quer em S, desaparecem na θ -junção e na junção natural
- a **junção externa** (θ - ou natural) inclui os tuplos pendentes de R ou S completados a nulos
 - ✖ $(7, 8, 9)$ é um tuplo pendente de R pois não aparece na θ -junção
 - ✖ (b, b, f) idem, na junção natural

R	A	B	C
1	2	3	
4	5	6	
7	8	9	

S	D	E
3	1	
6	2	

$R \bowtie^+ S$

$R \bowtie^+ S$ $B < D$				
A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2
7	8	9	⊥	⊥

R	A	B	C
a	b	c	
d	b	c	
b	b	f	
c	a	d	

S	B	C	D
b	c	d	
b	c	e	
a	d	b	

$R \bowtie^+ S$	A	B	C	D
a	b	c	d	
a	b	c	e	
d	b	c	d	
d	b	c	e	
c	a	d	b	
b	b	f	⊥	

Operações da álgebra relacional

17

Semi-junção

10 - Semi-junção $R \bowtie S$

- projecção nos atributos de R da junção natural de R e S
- $R \bowtie S = \Pi_R(R \bowtie S)$
 - ✖ R em Π_R representa os atributos de R (o seu esquema); em $R \bowtie S$ R representa a relação (a instância)
- outra expressão: $R \bowtie S = R \bowtie \Pi_{R \cap S}(S)$
- dá os tuplos de R que têm par em S

R			S		
A	B	C	B	C	D
a	b	c	b	c	d
d	b	c	b	c	e
b	b	f	a	d	b
c	a	d			

$R \bowtie S$		
A	B	C
a	b	c
d	b	c
c	a	d

Operações da álgebra relacional

18

Relações com atributos

- na junção natural e na semi-junção os atributos são importantes; para os tornar explícitos escreve-se $R(A_1, \dots, A_n)$
- é possível renomear colunas e fazer junções naturais como:

$$S = S(B, C, D)$$

B	C	D
b	c	d
b	c	e
a	d	b

$$S(E, F, G) \bowtie S(G, H, I)$$

E	F	G	H	I
a	d	b	c	d
a	d	b	c	e

- uma junção natural entre duas relações sem atributos comuns redonda num produto cartesiano porque, após este, não há nenhuma selecção a fazer (equivalente a fazer uma selecção com a condição *True*)
- $R(A, B, C) \bowtie S(G, H, I) = R \times S$

Leis algébricas

19

- Reunião

- associativa:
- comutativa:

$$R \cup (S \cup T) = (R \cup S) \cup T$$
$$R \cup S = S \cup R$$

- Produto cartesiano

- associativo:
- não comutativo:

$$R \times (S \times T) = (R \times S) \times T$$
$$R \times S \neq S \times R$$

- Junção natural

- associativa e comutativa (independência da ordem das colunas devida aos atributos):

$$R \bowtie S = S \bowtie R$$

- Por isso \bowtie generaliza facilmente:

$$R = R_1 \bowtie \dots \bowtie R_n$$

- R contém os tuplos μ tais que, para $1 \leq i \leq n$, μ restrinido aos atributos de R_i é um tuplo de R_i

- θ - junção

- não é comutativa mas é associativa (no caso de os índices serem válidos)

$$R \underset{i \theta_1 j}{\bowtie} (S \underset{k \theta_2 l}{\bowtie} T) = (R \underset{i \theta_1 j}{\bowtie} S) \underset{(r+k) \theta_2 l}{\bowtie} T$$

Esquema relacional de Cursos

20

- 1** Curso(codcurso, designacur)
- 2** Disciplina(coddis, sigla, designadis)
- 3** Turma(codcurso→Curso, ano, letra)
- 4** Professor(bip, nome, morada, telefone, habilitação, grupo)
- 5** Aluno(bia, nome, morada, telefone, data_nasc, [codcurso, ano, letra]→Turma)
- 6** Plano(codcurso→Curso, coddis→Disciplina)
- 7** Inscrito(coddis→Disciplina, bia→Aluno, resultado)
- 8** Lecciona(bip→Professor, coddis→Disciplina, [codcurso, ano, letra]→Turma)

Só as tabelas 4 – 7 são necessárias para os exercícios seguintes.

Linguagem de Interrogação

21

- Álgebra Relacional pode ser usada como linguagem de interrogação à BD
- *P1 - Relativamente à BD “Cursos” (ver atrás), quais os nomes dos professores do 12º grupo?*

$$\Pi_{\text{nome}} [\sigma_{\text{grupo} = '12'} (\text{Professor})]$$

- *P2 - Quais os nomes e datas de nascimento dos alunos do curso ‘CG1’ nascidos antes de 1983?*

$$\Pi_{\text{nome, data_nasc}} [\sigma_{\text{codcurso} = 'CG1'} \wedge \text{data_nasc} < 1983-01-01 (\text{Aluno})]$$

Linguagem de interrogação

22

Perguntas com junção

- *P3 - Nomes dos alunos inscritos à disciplina 327?*
 - nenhuma relação contém nomes de alunos e códigos de disciplina
 - mas a junção $\text{Aluno} \bowtie \text{Inscrito} = R$ contém:
 - R (bia, nome, morada, telefone, data_nasc, codcurso, ano, letra, coddis, resultado)
 - (i) - $\Pi_{\text{nome}} [\sigma_{\text{coddis} = 327} (\text{Aluno} \bowtie \text{Inscrito})]$
 - (ii) - $\Pi_{\text{nome}} [\text{Aluno} \bowtie \sigma_{\text{coddis} = 327} (\text{Inscrito})]$
 - esta maneira de ligar informações no modelo de dados dá muita liberdade para exprimir perguntas arbitrárias mas exige uma fase de optimização para executar (ii) mesmo que a pergunta seja (i)

Núcleo da álgebra relacional : Π, σ, \bowtie

Extensões à álgebra relacional

23

Eliminação de duplicados

- R
- $R' = \delta(R)$
 - elimina os tuplos repetidos de R
- *P4 – Quais os alunos inscritos a alguma disciplina?*
 - $\Pi_{\text{nome}} [\text{Aluno} \bowtie \delta [\Pi_{\text{bia}} (\text{Inscrito})]]$
 - A relação Inscrito contém os bi dos alunos tantas vezes quantas as cadeiras a que o aluno está inscrito
 - Assim, é necessário eliminar os bi repetidos para que os nomes dos alunos não apareçam repetidos

Extensões à álgebra relacional

24

Renomeações de atributos

- $R(A, B, C)$
- $R'(X, Y, Z) = \Pi_{X=A, Y=B, Z=C} [R(A, B, C)]$
 - ▶ onde não houver ambiguidades, a simples menção dos atributos, em conjunto com o nome da relação, faz a renomeação
 - OU
 - $R'(X, Y, Z) = R(A, B, C)$
 - OU
 - $R' = \Pi_{X=A, Y=B, Z=C}(R)$
 - ▶ expressões aritméticas (+, -, *, /)
- este mecanismo serve para dar nomes a expressões
- $S = \Pi_{W=A * B - C, U=C/B, A}(R)$
 - OU
- $S(W, U, A) = \Pi_{A * B - C, C/B, A}(R)$

Linguagem de interrogação

25

Expressões aritméticas

- P5 – *Obtenha a relação das inscrições com as classificações inflaccionadas de 20%.*
 - $\Pi_{\text{codd}, \text{bia}, \text{resultado}, \text{novo} = \text{resultado} * 1.2}$ (Inscrito)
- Outro exemplo:
- $\Pi_{\text{codd}, \text{bia}, \text{resultado}, \text{novo} = \text{resultado} + (20-\text{resultado})/10}$ (Inscrito)
- nos parâmetros da projecção:
 - no membro direito só podem ser usados nomes de atributos do argumento de Π ; no esquerdo só pode estar um atributo (novo...)

Extensões à álgebra relacional

26

Agregações

- Operadores de agregação
 - CNT (contagem), SUM (adição), AVG (média), MAX (máximo), MIN (mínimo)
- $S = \prod_{V = \text{CNT}(B)}(R)$
 - $S(V)$ tem um único valor, o número de tuplos de R com valor não nulo no atributo B ($\text{CNT}(\ast)$ conta todas as linhas) \rightarrow toda a relação agregada
- $T = \prod_{A, M = \text{MAX}(B)}(R)$
 - $T(A, M)$ tem tantos pares quantos os valores diferentes de A , sendo indicado para cada A o respectivo valor máximo de B (não nulo ...);
 - é feita uma partição segundo os atributos de projecção sem operadores de agregação e cada classe é agregada num só tuplo;
 - é possível misturar agregações e aritmética

Linguagem de interrogação

27

Perguntas com agregação

- P6 - Qual o número de inscrições, nota média das inscrições, soma de todas as notas e número de resultados não nulos ?
 - $R(NI, M, T, NR) = \Pi_{CNT(*)}, AVG(resultado), SUM(resultado), CNT(resultado)$ (Inscrito)
 - pode ser $M \neq T/NI$ se houver inscrições ainda sem resultado (valor nulo); tem que ser $M = T/NR$

Inscrito

coddis	bia	resultado
PA	97	14
PA	38	12
ITI	97	17
ITI	25	14
H	97	10
H	25	

R

NI	M	T	NR
6	13.4	67	5

Linguagem de interrogação

28

Agregação com partição

- P7 - *Quais as notas mínima, média e máxima de cada disciplina (independentemente do aluno)?*
 - $R = \prod_{\text{codd}is}, MI = \text{MIN}(\text{resultado}), ME = \text{AVG}(\text{resultado}), MA = \text{MAX}(\text{resultado})$ (Inscrito)

Inscrito

coddis	bia	resultado
PA	97	14
PA	38	12
ITI	97	17
ITI	25	14
H	97	10
H	25	

R

coddis	MI	ME	MA
PA	12	13	14
ITI	14	15.5	17
H	10	10	10

Linguagem de interrogação

29

Quantificação existencial

- P8 - *Obtenha os códigos dos alunos com inscrição a pelo menos uma das disciplinas do curso LEEC.*
 - $\delta(\Pi_{BIA}(\text{INSCRITO} \bowtie \sigma_{\text{CODCURSO}=\text{'LEEC'}}(\text{PLANO})))$

Linguagem de interrogação

30

Quantificação universal

- P9 - *Obtenha o código dos alunos com inscrição a **todas** as disciplinas do curso LEEC.*
 - $A = \Pi_{BIA}(\text{ALUNO})$
 - ▶ conjunto dos alunos
 - $D = \Pi_{CODDIS}[\sigma_{CODCURSO='LEEC'}(\text{PLANO})]$
 - ▶ conjunto das disciplinas do curso LEEC
 - $A \times D$
 - ▶ conjunto de todos os pares (aluno, disciplina da LEEC)
 - $NI = A \times D - \Pi_{BIA, CODDIS}(\text{INSCRITO})$
 - ▶ pares (aluno, disciplina da LEEC) tais que o aluno não tem inscrição à disciplina
 - $R = A - \Pi_{BIA}(NI)$
 - ▶ resultado (notar a dupla subtração)
- $R = \Pi_{BIA, CODDIS}(\text{INSCRITO}) / D$

Query language

31

More aggregation

- P10 - Qual o nome e média actual do aluno com melhor média do curso 'LEEC'?
- Só a média actual dos alunos
 - $MA = \Pi_{bia, media = AVG(resultado)} (Inscrito \bowtie \sigma_{CODCURSO='LEEC'} (PLANO))$
- Só a média máxima
 - $MaxM = \Pi_{media = MAX(media)} (MA)$
- Aluno cuja média é igual à média máxima
 - $R = \Pi_{nome, MA.media} (Aluno \bowtie MA \bowtie MaxM)$

Bases de Dados

1



Universidade do Porto

Faculdade de Engenharia

FEUP

- **INTRODUÇÃO**
- **MODELOS CONCEPTUAIS**
 - Diagrama de Classes UML
 - Modelo Entidade-Associação (E-A)
- **MODELO RELACIONAL**
- **LINGUAGEM DE DEFINIÇÃO DE DADOS**
- **INTERROGAÇÃO DE DADOS**
 - Álgebra relacional
 - **LINGUAGEM DE MANIPULAÇÃO DE DADOS (LMD) SQL**

Observação: baseado em slides desenvolvidos pelo Prof. Gabriel David (FEUP)

Origem

2

- Introduzida em 1976 como Linguagem de Manipulação de Dados (LMD) para System R (IBM)
- Primeira implementação comercial em 1979 (Oracle)
- Linguagem padrão de acesso a BD relacionais
- Normalização ANSI e ISO: SQL89, SQL92, SQL3
- Objectivo principal
 - Tratamento unificado da definição, manipulação e controlo dos dados, sendo útil para todas as classes de utilizadores.

InSTRUÇÕES DA LMD SQL

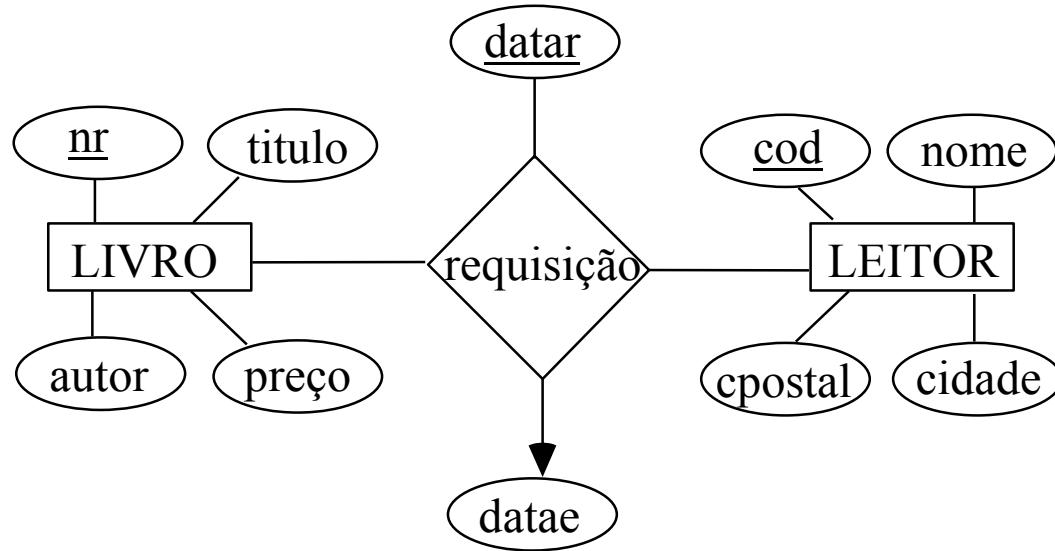
3

- INSERT: permite inserir registos;
- UPDATE: permite alterar valores de registos já existentes;
- DELETE: permite eliminar registos;
- SELECT: permite obter respostas a perguntas a partir dos dados existentes nas tabelas.
- Existem outras mas estas são as standard.

BD Biblioteca

4

- **esquema conceptual**



- Um leitor não pode requisitar o mesmo livro mais do que uma vez por dia.

- **esquema relacional**

- **livro(nr, titulo, autor, preço)**
- **leitor(cod, nome, cpostal, cidade)**
- **req(liv → livro.nr, lei → leitor.cod, datar, datae)**

Definição do esquema em SQL

5

create table livro

```
( nr          number(4)  primary key,  
  titulo      varchar2(20) not null,  
  autor       varchar2(20),  
  preço       number(4) );
```

create table leitor

```
( cod        number(4)  primary key,  
  nome       varchar2(20) not null,  
  cpost      number(4),  
  cidade     varchar2(20) );
```

create table req

```
( liv        number(4)  references livro,  
  lei        number(4)  references leitor,  
  datar      date,  
  datae      date,  
  constraint req_ck check (datar<=datae),  
  constraint req_pk primary key(liv, lei, datar) );
```

Carregamento das tabelas

6

- **INSERT INTO tabela VALUES(val, val, ...);**
 - adiciona uma linha com todos os valores e pela ordem correcta
- **INSERT INTO tabela(col, col, ...) VALUES(val, val, ...);**
 - adiciona uma linha só com os valores das colunas referidas
 - **INSERT INTO req VALUES (130, 6, '2013-06-15', null);**
equivalente a:
INSERT INTO req (liv, lei, datar) VALUES (130,6,'2013-06-15');

BIBLIOTECA

7

LIVRO

NR	TITULO	AUTOR	PREÇO
100	Os Maias	Eça de Queiroz	6.50€
110	Os Lusíadas	Luís de Camões	2.50€
120	A Selva	Ferreira de Castro	3.50€
130	A Capital	Eça de Queiroz	5.25€
140	Terra Fria	Ferreira de Castro	4.25€
150	A Relíquia	Eça de Queiroz	4.50€

BIBLIOTECA (2)

8

LEITOR

COD	NOME	CPOST	CIDADE
1	António	1000	Lisboa
2	Chico	4000	Porto
3	Marina	1100	Lisboa
4	Zeca	4100	Porto
5	Manuel	4400	Gaia
6	Mafalda	4470	Matosinhos
7	Rui	1200	Lisboa

BIBLIOTECA (3)

9

REQ

LIV	LEI	DATAR	DATAE
100	1	2013-01-01	2013-02-06
110	2	2013-01-05	2013-03-05
120	2	2013-02-15	2013-02-25
100	3	2013-03-10	2013-03-20
130	6	2013-06-15	
140	5	2013-04-15	2013-05-02
100	1	2013-04-30	2013-05-08
110	4	2013-04-21	2013-04-26
150	6	2013-06-30	2013-07-08
130	5	2013-07-04	2013-07-12

Instrução UPDATE

10

- **UPDATE tabela SET col=val, col = val, ... [WHERE ...];**
 - Atribui valores às colunas.
 - [] indica que é opcional.
 - Caso não exista a cláusula WHERE, as alterações afectam todos os registos da tabela.
 - `UPDATE req SET datae= '2013-06-22'
WHERE liv =130 AND datar= '2013-06-15';`
 - As funções para lidar com datas variam entre Sistemas Gestores de Bases de Dados (SGBDs)
 - O SQLite tem 5 funções para lidar com datas. Ver:
 - http://www.sqlite.org/lang_datefunc.html

Instrução DELETE

11

- **DELETE FROM tabela [WHERE ...];**
 - Elimina registos de uma tabela.
 - [] indica que é opcional.
 - Caso não exista a cláusula WHERE, são eliminados todos os registos da tabela.
 - `DELETE FROM leitor WHERE cod =7;`

InSTRUÇÃO SELECT

12

- **A instrução SELECT:**

- É muito versátil permitindo responder a quase todas as perguntas;
- Não consegue resolver o fecho transitivo. Por exemplo: saber todos os descendentes de uma dada pessoa;
- Mas para ser assim tão versátil tem necessariamente uma sintaxe com alguma complexidade. Por isso, vamos aprender a instrução SELECT através da resolução de casos.

Primeira pergunta

13

1 Mostrar toda a informação sobre todos os livros.

SELECT * FROM livro;

- as perguntas podem ocupar mais do que uma linha e em formato livre, por uma questão de legibilidade
- fim da pergunta: ;
- sintaxe errada : < mensagem explicativa >
- todas as colunas da tabela : *
- obrigatório haver **select** e **from**
 - ✖ No SQLite é possível não colocar o FROM;
 - ✖ Ex.: SELECT 25*363+8;

Resposta 1

14

NR	TITULO	AUTOR	PREÇO
100	Os Maias	Eça de Queiroz	6.50€
110	Os Lusíadas	Luís de Camões	2.50€
120	A Selva	Ferreira de Castro	3.50€
130	A Capital	Eça de Queiroz	5.25€
140	Terra Fria	Ferreira de Castro	4.25€
150	A Relíquia	Eça de Queiroz	4.50€

Selecção simples

15

2 Listar código e nome dos leitores cujo código é menor que 5.

```
SELECT cod, nome  
FROM leitor  
WHERE cod < 5;
```

- **select-from-where** assemelha-se ao cálculo relacional e faz uma escolha horizontal (selecção), seguida de uma escolha vertical (projecção)

COD	NOME
1	António
2	Chico
3	Marina
4	Zeca

Filtro mais elaborado

16

3 Listar o nome e a cidade dos leitores com nome a começar por 'M' e código entre 2 e 5.

```
SELECT nome, cidade  
FROM leitor  
WHERE nome LIKE 'M%'  
AND cod BETWEEN 2 AND 5;
```

=

```
SELECT nome, cidade  
FROM leitor  
WHERE nome LIKE 'M%'  
AND cod >= 2 AND cod <=5;
```

NOME	CIDADE
Marina	Lisboa
Manuel	Gaia

Pesquisa com cadeias

17

- Comparação com uma cadeia usando like:
 - % vale por qualquer sequência de 0 ou mais caracteres:
nome like 'M%' (Oracle, SQLite) **nome like 'M*' (Access)**
 - ▶ é comparação verdadeira com 'Marina', 'M'
 - O _ (?) vale por qualquer letra (uma e uma só);
nome like 'M_r%' **nome like 'M?r*''**
 - ▶ é comparação verdadeira com 'Mar', 'Maria', 'Moreira'
- Usando = faz-se a igualdade literal:
nome = 'M_r%'
 - ▶ só é verdade se nome for 'M_r%'

Eliminação de repetidos

18

4 Seleccionar as cidades com código postal superior a 2000.

```
SELECT cidade  
FROM leitor  
WHERE cpost > 2000;
```

- como vários leitores são da mesma cidade vão aparecer valores repetidos no resultado

CIDADE
Porto
Porto
Gaia
Matosinhos

Resposta com conjunto

19

```
SELECT DISTINCT cidade  
FROM leitor  
WHERE cpost > 2000;
```

- ❑ Em Oracle, forçar valores distintos tem como efeito lateral a ordenação

CIDADE
Gaia
Matosinhos
Porto

Filtro complexo

20

5 Seleccionar os livros do Eça com preço superior a 5€ e todos os livros de preço inferior a 3.75 € indicando o autor, o título, o preço e o número.

```
SELECT autor, titulo, preco, nr  
FROM livro  
WHERE autor LIKE '%Eca%' AND preco > 5 OR preco < 3.75;
```

AUTOR	TITULO	PREÇO	NR
Eça de Queiroz	Os Maias	6.50€	100
Luís de Camões	Os Lusíadas	2.50€	110
Ferreira de Castro	A Selva	3.50€	120
Eça de Queiroz	A Capital	5.25€	130

Expressões aritméticas

21

6 Escrever o número de dias que durou cada requisição nos casos em que duraram até 10 dias.

```
SELECT liv, lei, julianday(datae) – julianday(datar) Duracao
FROM req
WHERE Duracao <= 10;
```

- para renomear uma coluna, indica-se o novo nome a seguir à especificação da mesma, como no exemplo.

Expressões lógicas

22

LIV	LEI	Duração
120	2	10
100	3	10
100	1	8
110	4	5
130	5	8

- operadores reconhecidos, por ordem de precedência:
- operadores aritméticos
 - + , - (binário); || (concatenação)
 - * , /
 - + , - (unário)
- operadores de comparação
- operadores lógicos
 - not
 - and
 - or

Operadores de comparação

23

=, <>, <, >, <=, >=	igual, diferente, menor, maior, menor ou igual, maior ou igual
[not] in	pertença a conjunto
[not] between x and y	$x \leqslant \text{valor} \leqslant y$
exists	Sub-pergunta com pelo menos uma linha no resultado
x [not] like y	compara com padrão
is [not] null	é valor nulo

Dificuldades com operadores

24

- **in**

```
select * from leitor  
where cidade  
in ('Lisboa','Porto')
```

- **not in** dá nulo (sem resultado) se algum dos elementos do conjunto for nulo

cidade not in ('Lisboa','Porto', null) é equivalente a
cidade != 'Lisboa' and cidade != 'Porto' and cidade != null

- qualquer comparação com nulo dá nulo, excepto a comparação **is null**.

Ordenação da saída

25

7 Obtenha uma lista com os autores, livros e preços ordenada decrescentemente por autor e decrescentemente por preço.

```
SELECT autor, titulo, preco  
FROM livro  
ORDER BY autor DESC, preco DESC;
```

AUTOR	TITULO	PREÇO
Luís de Camões	Os Lusíadas	2.50
Ferreira de Castro	Terra Fria	4.25
Ferreira de Castro	A Selva	3.50
Eça de Queiroz	Os Maias	6.50
Eça de Queiroz	A Capital	5.25
Eça de Queiroz	A Relíquia	4.50

Funções de agregação

26

8 Obtenha o preço médio, valor total e o número de livros da biblioteca, bem como o valor do livro mais caro e o do mais barato (ufff...).

```
SELECT AVG(preco), SUM(preco), COUNT(*),  
MAX(preco), MIN(preco)  
FROM livro;
```

avg(preco)	sum(preco)	count(*)	max(preco)	min(preco)
4.4167	26.5	6	6.5	2.5

Agrupamento de linhas

27

9 Calcule o preço médio dos livros de cada autor.

```
SELECT autor, AVG(preco)  
FROM livro  
GROUP BY autor;
```

AUTOR	AVG(PREÇO)
Eça de Queiroz	5.4167
Luís de Camões	2.5
Ferreira de Castro	3.875

Agrupamento de linhas com filtro

28

10 Calcule o preço médio dos livros de cada autor, mas só para médias inferiores a 2.5€.

```
SELECT autor, AVG(preco)
FROM livro
WHERE AVG(preco) <= 2.5
GROUP BY autor;
```

ERRO!

misuse of aggregate: AVG()

having selecciona as linhas da agregação
where selecciona as linhas da tabela base

```
SELECT autor, AVG(preco)
FROM livro
GROUP BY autor
HAVING AVG(preco) <= 2.5;
```

certo!

AUTOR	AVG(PREÇO)
Luís de Camões	2.5

Perguntas encaixadas

29

11 Obtenha o título e preço do livro mais caro dos autores que começam por E.

Pergunta (1^a tentativa, parece a mais lógica ... para alguns):

```
SELECT titulo, MAX(preco)  
FROM livro  
WHERE autor LIKE 'E%';
```

Funciona em SQLite, mas não funciona na maioria dos outros SGBDs.

Mas cuidado! Esta forma não faz muito sentido! Veja o que acontece se em vez de usar MAX usar AVG ou SUM. Nesses casos, o título apresentado é o último devolvido pela selecção antes da agregação.

Subpergunta

30

Na verdade, este pedido é constituído por duas perguntas:

- 1 Qual é o preço máximo dos livros escritos por autores que começam por E?
- 2 Qual o título do livro cujo preço é igual ao determinado acima e cujo autor começa por E? (esta condição de começar por E não é redundante...)

```
SELECT titulo, preco  
FROM livro  
WHERE preco = (  
    SELECT MAX(preco)  
    FROM livro  
    WHERE autor LIKE 'E%' )  
AND autor LIKE 'E%';
```

TITULO	PREÇO
Os Maias	6.5

Exagerando...

31

12 Seleccione o título do segundo livro mais caro.

```
SELECT titulo
FROM livro
WHERE preco = (
    SELECT MAX(preco)
FROM livro
WHERE nr NOT IN (
    SELECT nr
    FROM livro
    WHERE preco = (
        SELECT MAX(preco)
        FROM livro));
```

TITULO
A Capital

É possível demonstrar teoricamente que qualquer relação que se consiga extrair da BD com SQL, extrai-se com uma única pergunta (nem sempre dá muito jeito...).

Análise

32

1) preçomax :=

SELECT MAX(preco)
FROM livro;

determina o preço máximo de todos os livros.

2) numeromax :=

SELECT nr
FROM livro
WHERE preco = preçomax;
números dos livros que custam o preço máximo.

3) segundopreço :=

SELECT MAX(preco)
FROM livro
WHERE nr NOT IN numeromax;
máximo preço dos livros cujo número é diferente do dos livros com preço máximo (ou seja, o segundo maior preço...).

4) resultado :=

SELECT titulo
FROM livro
WHERE preco = segundopreço;
determina o título dos livros com preço igual ao segundo maior preço. E já está!

Perguntas com várias tabelas

33

13 Escreva os títulos e datas de requisição dos livros requisitados depois de 2013-04-01.

```
SELECT titulo, datar  
FROM livro, req  
WHERE nr = liv  
AND datar >= '2013-04-01';
```

TITULO	DATAR
Os Maias	2013-04-30
Os Lusíadas	2013-04-21
A Capital	2013-07-04
A Capital	2013-06-15
Terra Fria	2013-04-15
A Relíquia	2013-06-30

Núcleo da álgebra relacional

34

- o conjunto de cláusulas **select-from-where** é equivalente a um produto cartesiano, seguido de uma selecção e de uma projecção:

select campo₁, ..., campo_n

from tabela₁, ..., tabela_m

where F;

\Leftrightarrow

$\Pi_{\text{campo}_1, \dots, \text{campo}_n} (\sigma_F (\text{tabela}_1 \times \dots \times \text{tabela}_m))$

\Leftrightarrow

$\Pi_{\text{campo}_1, \dots, \text{campo}_n} (\text{tabela}_1 \bowtie_{F'} \dots \bowtie_{F''} \text{tabela}_m)$

Inclusão em conjunto

35

14 Liste, para cada requisição, o título do livro e o nome do leitor, no caso de o código postal ser 1000, 4000 ou 4470.

- **SELECT titulo, nome
FROM livro, req, leitor
WHERE nr = liv AND lei = cod
AND cpost IN (1000, 4000, 4470);**
- Equivalente a:
**SELECT titulo, nome
FROM livro, req, leitor
WHERE nr = liv AND
lei = cod AND
(cpost =1000 OR cpost = 4000 OR cpost = 4470);**
parênteses obrigatórios, atendendo à precedência

Resposta 13

36

TITULO	NOME
Os Maias	Antonio
Os Lusíadas	Chico
A Selva	Chico
A Capital	Mafalda
A Reliquia	Mafalda

Condições sobre tuplos

37

15 Quantos Antónios moram em Lisboa e quantos Zecas moram no Porto?

```
SELECT nome, cidade, COUNT(*) (Oracle)  
FROM leitor  
WHERE (nome, cidade) IN  
    (('Antonio','Lisboa'),('Zeca', 'Porto'))  
GROUP BY nome, cidade;  
• Equivalente a:  
SELECT nome, cidade, COUNT(*)  
FROM leitor  
WHERE nome = 'Antonio' AND cidade = 'Lisboa'  
OR nome = 'Zeca' AND cidade = 'Porto'  
GROUP BY nome, cidade;
```

Resposta 14

38

NOME	CIDADE	COUNT(*)
Zeca	Porto	1

Agregação de agregação

39

16 Procure o livro cujas requisições têm maior duração média, exceptuando 'Terra Fria'.

```
SELECT titulo, AVG(julianday(datae) – julianday(datar))  
FROM livro, req  
WHERE nr = liv AND titulo != 'Terra Fria'  
GROUP BY titulo  
HAVING AVG(julianday(datae) – julianday(datar)) = (  
    SELECT MAX(media) FROM (  
        SELECT AVG(julianday(datae) – julianday(datar)) media  
        FROM req, livro  
            WHERE titulo != 'Terra Fria' AND nr = liv  
        GROUP BY titulo));
```

Autojunção

40

17 Obtenha a lista dos pares de pessoas que moram na mesma cidade.

```
SELECT p.nome, q.nome  
FROM leitor p, leitor q  
WHERE p.cod != q.cod AND p.cidade = q.cidade;
```

- para responder a esta pergunta, precisamos de duas *cópias* da tabela de leitores; como não temos duas cópias físicas, criamos duas cópias lógicas: p e q.
- Ex: **p.cidade = q.cidade** faz a junção das duas tabelas **p** e **q** sobre o atributo **cidade**.

Subtracção de conjuntos

41

18 Obtenha os leitores que não requisitaram o livro 150.

```
SELECT nome  
FROM leitor  
WHERE cod NOT IN  
(SELECT lei  
FROM req  
WHERE liv = 150);
```

○ Alternativa:

```
SELECT cod FROM leitor  
MINUS  
SELECT lei FROM req  
WHERE liv = 150;
```

Reunião e intersecção

42

19 Quais os dias em que houve requisições ou entregas de livros?
E quais os dias em que houve requisições e entregas?

Pergunta da reunião:

```
SELECT datae FROM req  
UNION  
SELECT datar FROM req;
```

Pergunta da intersecção:

```
SELECT datae FROM req  
INTERSECT  
SELECT datar FROM req;
```

Operador all

43

20 Quais os livros mais caros do que (todos) os livros do Ferreira de Castro?

```
SELECT titulo      ¬SQLite  
FROM livro  
WHERE preco > ALL  
      (SELECT preco  
       FROM livro  
       WHERE autor =  
             'Ferreira de Castro');
```

Alternativa (viável em SQLite):

- operador **all** exige que a comparação seja verdadeira para todos os valores do resultado da subpergunta.

```
SELECT titulo  
FROM livro  
WHERE preco >  
      (SELECT MAX(preco)  
       FROM livro  
       WHERE autor =  
             'Ferreira de Castro');
```

Operador some (any).

44

21 Quais os livros mais baratos do que algum livro do Eça?

SELECT titulo

FROM livro

WHERE preco < **SOME**
(SELECT preco

FROM livro

WHERE autor LIKE 'Eça%');

¬SQLite

ou ANY Alternativa (viável em SQLite):

```
SELECT titulo FROM livro  
WHERE preco <  
(SELECT MAX(preco)  
FROM livro  
WHERE autor LIKE 'Eça%');
```

- o operador some (any) exige que a comparação seja verdadeira para pelo menos um dos valores do resultado da subpergunta

Operadores in e exists

45

22 Quais os titulos dos livros requisitados depois de 2013-06-20?

```
SELECT titulo  
FROM livro  
WHERE nr IN  
(SELECT liv FROM req  
WHERE datar > '2013-06-20') ;
```

Alternativa:

```
SELECT titulo  
FROM livro  
WHERE EXISTS -- where 0 <  
(SELECT * -- (SELECT COUNT(*)  
FROM req  
WHERE nr = liv  
AND datar > '2013-06-20') ;
```

Símbolo que significa comentário

Sub-pergunta variável

46

- operador **exists** testa se o resultado da sub-pergunta não é vazio (o mesmo que **0 <**); **not exists** — **0 =**
- Sub-pergunta *constante*: na primeira versão, a sub-pergunta pode ser substituída pelo seu resultado (130,150) e este usado na pergunta exterior — avaliação de dentro para fora
- Sub-pergunta *variável*: na segunda versão, para cada tuplo da pergunta exterior (linha de livro), a sub-pergunta interior tem que ser reavaliada, pois contém uma referência a um atributo (**nr**) da tabela declarada na pergunta exterior — avaliação de fora para dentro.

Contagem

47

23 Obtenha os números e títulos dos exemplares que foram requisitados mais do que uma vez.

```
SELECT nr, titulo  
FROM livro, req  
WHERE nr = liv  
GROUP BY nr, titulo  
HAVING COUNT(*) > 1 ;
```

```
SELECT DISTINCT nr, titulo  
FROM livro, req r1, req r2  
WHERE nr = r1.liv AND nr = r2.liv  
AND r1.datar != r2.datar;
```

Alternativa (se em cada dia, cada exemplar for requisitado no máximo uma vez):

```
SELECT nr, titulo  
FROM livro, req  
WHERE nr = liv AND  
datar != SOME  
(SELECT datar  
FROM req  
WHERE livro.nr = liv);
```

---SQLite

Pertença de tuplos

48

24 Obtenha o número, título e preço das obras que têm mais do que um exemplar na biblioteca, com preço inferior a 4.50€.

```
SELECT l1.nr, l1.titulo, l1.preco  
FROM livro l1, livro l2  
WHERE l1.preco < 4.5 AND  
l1.preco < 4.5 AND  
l1.autor = l2.autor AND  
l1.titulo = l2.titulo AND  
l1.nr != l2.nr;
```

- Utilizando uma subpergunta

```
SELECT nr, titulo, preco  
FROM livro l  
WHERE (titulo, autor) IN  
(SELECT titulo, autor  
FROM livro  
WHERE nr != l.nr)  
AND preco < 4.5;
```

¬SQLite

Sub-pergunta variável

49

- estratégia de *divisão e conquista*:

SELECT nr, titulo, preco

FROM livro l

WHERE 1<

(SELECT COUNT(*)

FROM livro

WHERE titulo = l.titulo AND autor = l.autor AND preco < 4.5) ;

Quantificação universal

50

25 Quais os leitores que leram todos os livros?

```
SELECT DISTINCT nome      -- R  
FROM leitor  
WHERE cod NOT IN  
(SELECT cod AS cod1 --  $\Pi_{\text{lei}}(T-S)$   
FROM livro, leitor -- T  
WHERE NOT (nr IN  
(SELECT liv      -- S  
FROM req  
WHERE lei=cod1)));
```

- Manipulação de conjuntos:
 - $T = \Pi_{\text{nr}}(\text{livro}) \times \Pi_{\text{cod}}(\text{leitor})$
 - $S = \Pi_{\text{liv}, \text{lei}}(\text{req})$
 - $R = \text{leitor} - \Pi_{\text{lei}}(T-S)$

Utiliza uma sub-
pergunta variável.

Formulações alternativas

51

```
select nome  
from leitor  
where not exists  
  (select nr  
   from livro  
   where not exists  
     (select lei  
      from req  
      where liv = nr and lei = cod ));
```

- formulação directa das expressões de cálculo, usando uma sub-pergunta variável para cada elemento do produto cartesiano leitor x livro e o operador not exists

Estratégia da contagem

52

```
select nome  
from leitor  
where cod in  
(select lei  
from req  
group by lei  
having count(distinct liv) =  
(select count(*)  
from livro));
```

- operador **distinct** crucial para não contar duplicados

Inserção

53

26 Insira a requisição dos livros 100 e 120 feita pelo leitor 4 a 88-07-11.

```
insert into req(liv,lei,datar)  
values(120, 4, '88-07-11');  
insert into req(liv,lei,datar)  
values(100, 4, '88-07-11') ;
```

- se se dessem valores a todos os atributos não era necessário indicar a lista de atributos a seguir ao nome da tabela
- os atributos não preenchidos ficam com os valores por omissão definidos para a coluna ou com valor nulo

Memorização de resultado

54

27 Insira, na tabela dos perdidos, os livros requisitados há mais de 30 dias.

```
create table perdidos  
(nr number(4) primary key,  
 titulo varchar2(20) not null,  
 autor varchar2(20),  
 preco number(4) );
```

```
insert into perdidos  
(select * from livro where nr in  
(select liv from req  
where julianday("now") –  
julianday(datar) > 30  
and datae is null );
```

- a tabela perdidos tem existir antes de fazer o insert
- “now” quando usado numa função de datas/tempo devolve o timestamp corrente.
- Podem-se fazer os dois passos num só através da forma create table perdidos as select ...

```
create table if not exists perdidos as  
select * from livro where nr in  
(select liv from req  
where julianday("now") –  
julianday(datar) > 30  
and datae is null);
```

Apagar

55

28 Retire os livros mencionados na pergunta anterior da tabela dos livros.

```
DELETE FROM livro
where nr in
(select liv from req
    where julianday("now") -
        julianday(datar) > 30
    and datae IS NULL);
```

- só se pode apagar numa tabela de cada vez
- pode ser usada qualquer pergunta para seleccionar os registos a apagar.

Modificar

56

29 Actualize o preço dos livros de código superior a 130 com 20% de inflação.

UPDATE livro

SET preco = preco * 1.2

WHERE nr > 130;

- só se pode actualizar numa tabela de cada vez, mas pode haver **set** para vários atributos em simultâneo
- a cláusula **set** admite qualquer expressão para modificar um campo, inclusive o resultado de uma pergunta, no caso de retornar apenas um valor

Língua natural

57

30 Quais os códigos dos leitores que requisitaram o livro 110 ou o livro 120? Quais os códigos dos leitores que requisitaram o livro 110 e o livro 120?

select lei

from req

where liv = 110 or liv = 120;

- para a disjunção, a resposta inclui os leitores 2 e 4; no caso da conjunção a pergunta

select lei

from req

where liv = 110 and liv = 120;

Conjunção

58

- dá um resultado vazio quando se estava à espera que desse 2! O problema é que não há nenhuma requisição que seja simultaneamente dos livros 110 e 120.
- reformulação, considerando que se tem que comparar duas requisições:
**select a.lei
from req a, req b
where a.lei=b.lei and a.liv = 110 and b.liv = 120;**
- a línguagem natural é muito traiçoeira.

Vistas implícitas

59

31 Quais os títulos dos livros que estão requisitados?

```
select titulo  
from livro, (select liv, lei  
            from req  
            where datae is null) requisitados  
where nr = requisitados.liv;
```

- Em SQL/92 é possível usar sub-perguntas como se fossem vistas, em várias situações, em especial na cláusula **from**.

Junções explícitas

60

32 Qual o número de livros requisitados por cada leitor, indicando o seu código e nome.

```
select cod, nome, count(liv)
from req inner join leitor on lei=cod
group by cod, nome;
```

- Em SQL/92 é possível usar junções explícitas como perguntas ou na cláusula **from**, com as variantes:
 - **cross** (produto cartesiano) (sem condição **on**)
 - **inner** (junção normal, por omissão)
 - **outer** (junção externa), ver próximo slide
 - **natural** (junção natural; pode também ser externa) (sem condição)

Junções externas

61

33 Qual o número de livros requisitados por cada leitor, indicando o seu código e nome.

```
select cod, nome, count(liv)
from leitor left join req on cod=lei
group by cod, nome;
```

- junção externa
 - **left** – todas as linhas da esquerda
 - **right** – todas as linhas da direita
 - **SQLite** **full** – todas as linhas de ambas
- Pode ser **natural left** ...

COD	NOME	COUNT(LIV)
1	Antonio	2
2	Chico	2
3	Marina	1
4	Zeca	1
5	Manuel	2
6	Mafalda	2
7	Rui	0

Vistas

62

34 Crie uma vista para os livros requisitados indicando o título do livro, o nome do leitor e a duração da requisição.

```
create view requisitado as
select titulo, 'Sr. ' || nome, date('now')-dataR
from req, livro, leitor
where liv=nr and lei=cod and dataE is null;
select * from requisitado;
```

- só se podem alterar as vistas que assentem numa única tabela base
- o operador `||` concatena cadeias de caracteres

Vista e junção externa

63

35 Crie uma vista com o código e o nome do leitor e o número de livros que já requisitou.

```
create view estatistica as
select cod, nome, count(distinct liv)
from leitor left join req
on lei = cod
group by cod, nome;
```

Tratamento de valores nulos

64

36 Calcule as durações de todas as requisições, contabilizando até à data actual os não entregues.

- Utilizar a função
`coalesce(col, valorSeNulo)`
 - Devolve o valor *col* se não for nulo ou o *valorSeNulo* caso *col* seja nulo

```
select liv, lei,  
coalesce(julianday(datae),julianday(date('now')))  
-julianday(datar) duracao  
from req;
```

- A função coalesce existe em quase todos os SGBDs.

LIV	LEI	DURACAO
100	1	36
110	2	59
120	2	10
100	3	10
130	6	5974,0265625
140	5	17
100	1	8
110	4	5
150	6	8

CASE

65

37 Substitua Lisboa por Alfacinha, Porto por Tripeiro, e Gaia por Marroquino, deixando os outros casos com Ignoto.

```
SELECT nome,  
       CASE cidade  
         WHEN 'Lisboa' THEN 'Alfacinha'  
         WHEN 'Porto' THEN 'Tripeiro'  
         WHEN 'Gaia' THEN 'Marroquino'  
         ELSE 'Ignoto'  
       END AS origem  
FROM leitor;
```

Nome	Origem
Antonio	Alfacinha
Chico	Tripeiro
Marina	Alfacinha
Zeca	Tripeiro
Manuel	Marroquino
Mafalda	Ignoto
Rui	Alfacinha

Uma pseudo-colunas

66

38 Crie uma vista sobre os livros em que, para além das colunas respectivas se mostre também o número de linha do resultado.

```
select rowid, livro.*  
from livro;
```

- **Rowid** – endereço interno da linha na BD
 - Permite acessos muito rápidos mas é afectado por operações de exportação/importação, pelo que não pode ser usado de forma geral

Top-ten

67

39 Obtenha a lista dos três livros mais caros.

```
SELECT titulo, preco  
FROM livro  
ORDER BY preco DESC LIMIT 3;
```

Titulo	Preço
Os Maias	1100
A Capital	1050
A Relíquia	900

Álgebra relacional vs. LMD:SQL

68

Vamos considerar as tabelas:

Aluno(idAluno, nomeAluno, idade, idCurso->Curso)

AlunoExt(idAluno, nomeAluno, idade, idCurso->Curso)

Curso(idCurso, nomeCurso)

Professor(idProfessor, idade)

Operadores	Álgebra relacional	LMD:SQL
União	$\text{Aluno} \cup \text{AlunoExt}$	<code>SELECT * FROM aluno UNION SELECT * FROM AlunoExt;</code>
Intersecção	$\text{Aluno} \cap \text{AlunoExt}$	<code>SELECT * FROM aluno INTERSECT SELECT * FROM AlunoExt;</code>
Diferença	$\text{Aluno} - \text{AlunoExt}$	<code>SELECT * FROM aluno EXCEPT SELECT * FROM AlunoExt;</code>
Produto cartesiano	$\text{Aluno} \times \text{Curso}$	<code>SELECT * FROM aluno, Curso;</code>
Projecção	$\pi_{\text{idAluno}, \text{nomeAluno}}(\text{Aluno})$	<code>SELECT idAluno, nomeAluno FROM aluno;</code>
Selecção	$\sigma_{\text{idAluno} < 5}(\text{Aluno})$	<code>SELECT * FROM ALUNO WHERE idAluno < 5;</code>

Álgebra relacional vs. LMD:SQL

69

Vamos considerar as tabelas:

Aluno(idAluno, nomeAluno, idade, idCurso->Curso)

AlunoExt(idAluno, nomeAluno, idade, idCurso->Curso)

Curso(idCurso, nomeCurso)

Professor(idProfessor, idade)

Operadores	Álgebra relacional	LMD:SQL
Divisão	Ver questão 24	Ver questão 24
θ-junção	Aluno $\bowtie_{\text{Aluno.idade} > \text{Professor.idade}}$ Professor	SELECT * FROM Aluno, Professor WHERE Aluno.idade > Professor.idade;
Junção natural	Aluno \bowtie Curso	SELECT * FROM Aluno NATURAL JOIN Curso;
Junção externa	Aluno \bowtie^+ Curso	SELECT Aluno.* , Curso.* FROM Aluno LEFT JOIN Curso ON Aluno.idCurso=Curso.idCurso UNION ALL SELECT Aluno.* , Curso.* FROM Curso LEFT JOIN Aluno ON Curso.idCurso = Aluno.idCurso WHERE Aluno.idCurso IS NULL;
Semi-junção	Curso \bowtie Aluno	SELECT * FROM Curso WHERE idCurso IN (SELECT idCurso From Aluno);

SQL - Access control

- Why an access control ?
- Views
- Authorization

Claudine Tacquard, Ecole Polytechnique de l'Université de Tours - France

Presentation based on the slides created by Prof. Jeffrey D. Ullman

Access control : Why ?

- **Secrecy:** Users should not be able to see things they are not supposed to.
- **Integrity:** Users should not be able to modify things they are not supposed to.
- **Availability:** Users should be able to see and modify things they are allowed to.



Access control : Views

- Views can be used to present necessary information (or a summary), while **hiding details** in underlying relation(s).
- A view is a “**virtual table**” : a relation that is defined in terms of the contents of other tables and views.
- In contrast, a relation whose value is **really stored** in the database is called a ***base table***.

Declare by:

CREATE VIEW <name> AS <query>;



Access control : Views

View Definition

- Example:

Person (Id_Pers, first_name, last_name, #id_City)

City (id_City, CityName, Area)

Persons from Paris ?

```
CREATE VIEW PersonFromParis AS
    SELECT first_name, last_name
    FROM Person, City
    WHERE Person.id_City = City.id_City
        and CityName = 'Paris';
```



Access control : Views

Accessing a View

- You may query a view as if it were a base table.
 - There is a limited ability to modify views if the modification makes sense as a modification of the underlying base table.
- Example:

```
SELECT first_name FROM PersonFromParis  
WHERE = last_name = 'Dupont';
```



Access control : Views

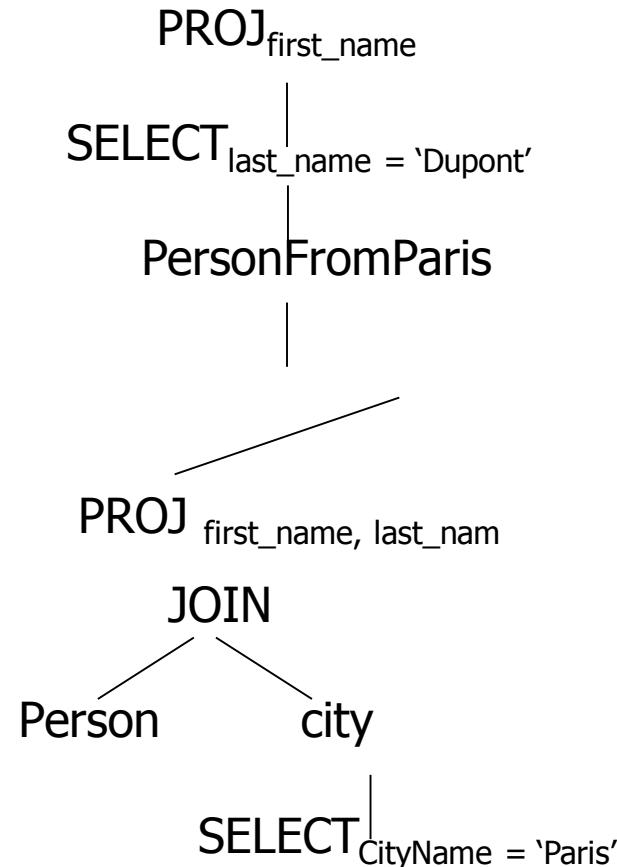
What Happens When a View Is Used?

1. The DBMS starts by interpreting the query as if the view were a base table.
 - Typical DBMS turns the query into something like relational algebra.
2. The queries defining any views used by the query are also replaced by their algebraic equivalents, and “spliced into” the expression tree for the query.



Access control : Views

Example: View Expansion



Access control : Views

DMBS Optimization

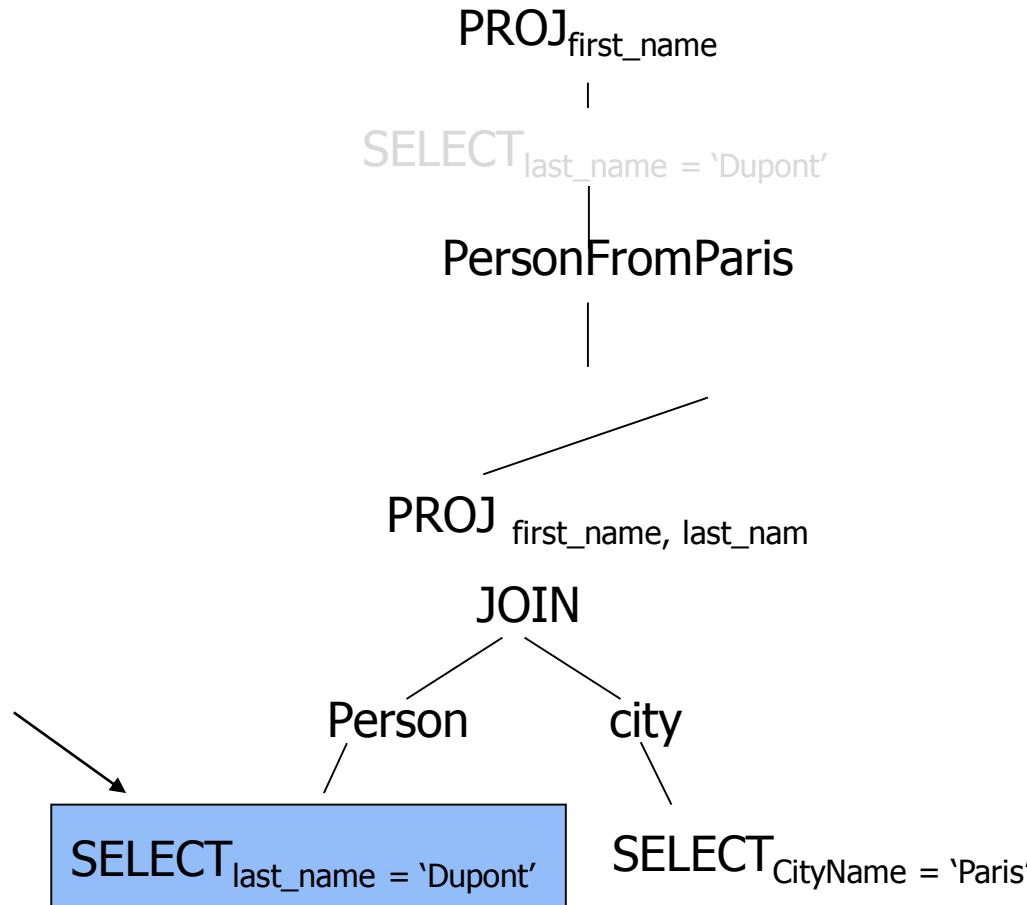
- It is interesting to observe that the typical DBMS will then “optimize” the query by transforming the algebraic expression to one that can be executed faster.
- Key optimizations:
 1. Push selections down the tree.
 2. Eliminate unnecessary projections.



Access control : Views

Example: Optimization

Most tuples
are eliminated
from Person
before the
expensive join.



Access control : Materialized View

- A **materialized view (MV)** is a database object that stores the **results of a query** at a single point in time. Unlike a view, materialized view is not virtual.
- MV may be stored locally or remotely in other site.
- MV are used by applications that do not require current data or that require data valid at a specific point in time.
- MV increases query performance since it contains results of a query



Access control : Materialized View

Refresh Methods :

- 1. Complete Refresh:** essentially re-creates the materialized view
- 2. Fast Refresh:**
 - First identifies the changes that occurred since the most recent refresh of the materialized view and then applies these changes to the materialized view.
 - Fast refreshes are **more efficient** than complete refreshes when there are few changes or the view is refreshed frequently.

CREATE MATERIALIZED VIEW <name> AS <query>; (not supported by SQLite)



Access control : Authorization

- A file system identifies certain privileges on the objects (files) it manages.
 - Typically read, write, execute.
- A file system identifies certain participants to whom privileges may be granted.
 - Typically the owner, a group, all users.



Access control: Authorization Privileges (1)

- SQL identifies a more detailed set of privileges on objects (relations, views,...) than the typical file system.
- Nine privileges in all, some of which can be restricted to one column of one relation.



Access control: Authorization Privileges (2)

Some important privileges on a relation:

- SELECT = right to query the relation.
- INSERT = right to insert tuples (May apply to only one attribute)
- DELETE = right to delete tuples.
- UPDATE = right to update tuples (May apply to only one attribute).



Access control: Authorization

Example: Privileges

- For the statement below:

INSERT INTO R1 (A)

SELECT A FROM R2

WHERE NOT EXISTS

(SELECT * FROM R1

WHERE R2.A = R1.A);

- We require privileges SELECT on R1 and R2, and INSERT on R1 or R1.A

R1	A	B
a1	b1	
a2	b1	
a3	b1	
a4	b1	
a5	b3	
a6	@	

R2	A	C
a6	c5	
a3	c1	

Access control: Authorization

Authorization ID's

- Privileges are actually assigned to **authorization ids**, which can denote a **single user** or a **group of users**.
- For a single user the *authorization ID*, is typically his name.
- There is an authorization ID PUBLIC.
Granting a privilege to PUBLIC makes it available to any authorization ID.

CREATE USER <user> [IDENTIFIED BY <password>]

ex : CREATE USER peter IDENTIFIED BY 'mypass'



Access control: Authorization Granting Privileges

- You have all possible privileges on the objects, such as relations, that you create.
- You may grant privileges to other users (authorization ID's), including PUBLIC.
- You may also grant privileges WITH GRANT OPTION, which lets the grantee also grant this privilege.



Access control: Authorization

The GRANT Statement

- To grant privileges, say:

GRANT <list of privileges>

ON <relation or other object>

TO <list of authorization ID's>;

- If you want the recipient(s) to be able to pass the privilege(s) to others add:

WITH GRANT OPTION



Access control: Authorization

Example: GRANT

- Suppose you are the owner of Person. You may say:

```
GRANT SELECT, UPDATE (id_City)
```

```
ON Person
```

```
TO peter;
```

- Now Peter has the right to issue any query on Person and can update the id_City component only.



Access control: Authorization

Example: Grant Option

- Suppose we also grant:

```
GRANT UPDATE ON Person TO peter  
WITH GRANT OPTION;
```

- Now, Peter can not only update any attribute of Person, but can grant to others the privilege UPDATE ON Person.
 - Also, he can grant more specific privileges like UPDATE(id_City) ON Person.



Access control: Authorization Revoking Privileges

REVOKE <list of privileges>

ON <relation or other object>

FROM <list of authorization ID's>;

- Your grant of these privileges can no longer be used by these users to justify their use of the privilege.
 - But they may still have the privilege because they obtained it independently from elsewhere.



Access control: Authorization

Set of Privileges

- A **role** is a set of privileges that can be granted to users or to other role.

```
CREATE ROLE role_name [ AUTHORIZATION owner_name ]
```

Ex :

```
CREATE ROLE resource_management;  
GRANT SELECT, INSERT, UPDATE ON Person  
TO resource_management;  
CREATE USER jean;  
GRANT resource_management TO jean;
```



SQL - Trigger

- Motivation
- Définition
- Exemple



Trigger Motivation

- Attribute and tuple-based checks have limited capabilities.
 - `CHECK(<condition>)` →Attribute or tuple
 - They are checked at known times : only on insert or update.
- Assertions are sufficiently general for most constraint applications, but they are hard to implement efficiently.
 - In principle, they should be checked after every modification to any relation of the database
 - The DBMS must have real intelligence to avoid checking assertions that couldn't possibly have been violated.



Trigger Solution

- A trigger allows the user to specify **when the check occurs**.
- Like an assertion, a trigger has a general-purpose condition and also can perform any sequence of SQL database modifications.



Trigger

Event-Condition-Action Rules

- Another name for “trigger” is *ECA rule*, or Event-Condition-Action rule.
- *Event* : typically a type of database modification, e.g., “insert on Table R1”
- *Condition* : Any SQL Boolean-valued expression.
- *Action* : Any SQL statements.



Trigger Example

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

- Instead of using a foreign-key constraint and rejecting insertions into Sells(bar, beer, price) with **unknown beers**, a trigger can add that beer to Beers, with a NULL manufacturer.

Trigger

Example of Trigger Definition

CREATE TRIGGER BeerTrig

AFTER INSERT ON Sells

The event

REFERENCING NEW ROW AS NewTuple

FOR EACH ROW

WHEN (NewTuple.beer NOT IN

(SELECT name FROM Beers))

The condition

INSERT INTO Beers(name)

VALUES(NewTuple.beer);

The action

Trigger

CREATE TRIGGER

CREATE TRIGGER <name>

Option:

CREATE OR REPLACE TRIGGER <name>

if you want to modify an existing trigger.



Trigger The Event

AFTER can be BEFORE.

Also, INSTEAD OF, if the relation is a view.

A clever way to execute view modifications: have triggers translate them to appropriate modifications on the base tables.

INSERT can be DELETE or UPDATE.

And UPDATE can be UPDATE . . . ON a particular attribute.



Trigger

Event Options: FOR EACH ROW

Triggers are either *row-level* or *statement-level*.

FOR EACH ROW indicates row-level; its absence indicates statement-level.

Row level triggers are executed once for each modified tuple.

Statement-level triggers execute once for an SQL statement, regardless of how many tuples are modified.



Trigger

Event Options: REFERENCING

- INSERT statements imply a new tuple (for row-level) or new set of tuples (for statement-level).
- DELETE implies an old tuple or table.
- UPDATE implies both.
- Refer to these by

[NEW OLD][TUPLE TABLE] AS <name>



Trigger The Condition

Any boolean-valued condition is appropriate.

It is **evaluated before or after** the triggering event,
depending on whether BEFORE or AFTER is used in the event.

But always before the changes take effect

Access the new/old tuple or set of tuples through the names declared in the REFERENCING clause.



Trigger The Action

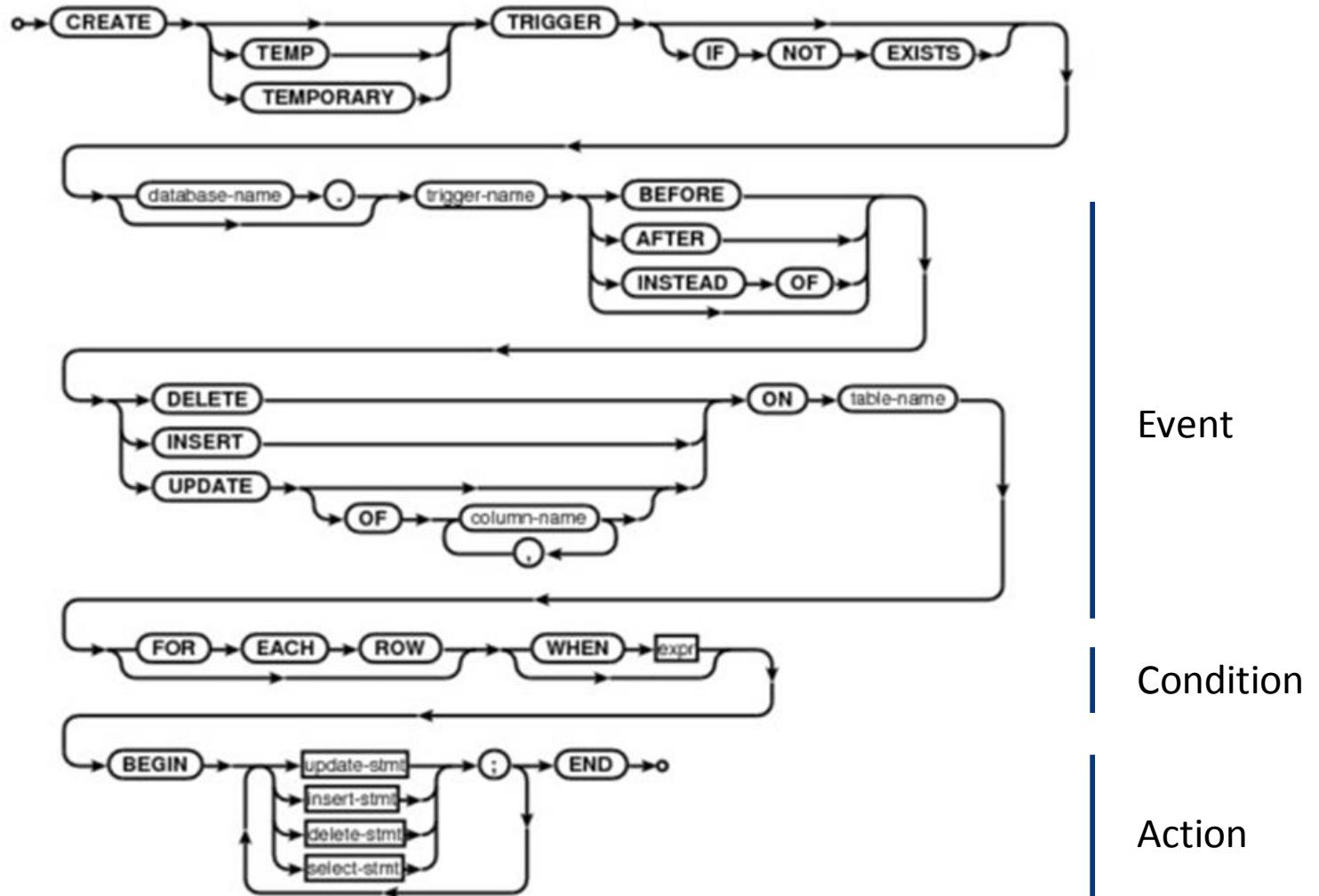
There can be more than one SQL statement in the action.

Surround by BEGIN . . . END if there is more than one.

But queries make no sense in an action, so we are really limited to modifications.



Trigger SQLite syntax



Trigger

Another Example (1)

Using `Sells(bar, beer, price)`

and

a unary relation `RipoffBars(bar)` created for the purpose,
maintain a list of bars that raise the price of any beer by more
than \$1.



Trigger

Another Example (2)

CREATE TRIGGER PriceTrig

AFTER UPDATE OF price ON Sells

REFERENCING

OLD ROW as old1

NEW ROW as new1

FOR EACH ROW

WHEN(new1.price > old1.price + 1.00)

INSERT INTO RipoffBars

VALUES(new1.bar);

The event –
only changes
to prices

Updates let us
talk about old
and new tuples

We need to consider
each price change

Condition:
a raise in
price > \$1

When the price change
is great enough, add
the bar to RipoffBars

Triggers and Views

Generally, it is impossible to modify a view, because it doesn't exist.

But an **INSTEAD OF** trigger lets us interpret view modifications in a way that makes sense.

Example:

We'll design a view Synergy that has (drinker, beer, bar) triples such that the bar serves the beer, the drinker frequents the bar and likes the beer.



Trigger on View

CREATE VIEW Synergy AS

Pick one copy of
each attribute

```
SELECT Likes.drinker, Likes.beer, Sells.bar
```

FROM Likes, Sells, Frequents

WHERE Likes.drinker = Frequents.drinker

AND Likes.beer = Sells.beer

AND Sells.bar = Frequents.bar;

Natural join of Likes,
Sells, and Frequents

Trigger on View

We cannot insert into Synergy --- it is a view.

But we can use an **INSTEAD OF trigger** to turn a (drinker, beer, bar) triple into three insertions of projected pairs, one for each of Likes, Sells, and Frequents.

The Sells.price will have to be NULL.



Trigger on View

CREATE TRIGGER ViewTrig

INSTEAD OF INSERT ON Synergy

REFERENCING NEW ROW AS n

FOR EACH ROW

BEGIN

 INSERT INTO LIKES VALUES(n.drinker, n.beer);

 INSERT INTO SELLS(bar, beer) VALUES(n.bar, n.beer);

 INSERT INTO FREQUENTS VALUES(n.drinker, n.bar);

END;



Índices

1

- *Índice* = estrutura de dados utilizada para acelerar as consultas à base de dados.
- Pode ser uma tabela de dispersão (tabela *hash*), mas é quase sempre uma árvore de pesquisa balanceada com nós gigantes chamada *B-tree*.
- Pode ainda ser do tipo *bitmap* especialmente quando:
 - O atributo tem poucos valores distintos (por ex.: género, masculino ou feminino) numa tabela com muitos registo;
 - Consultas que envolvem operadores de agregação pelo atributo indexado.

Índices

2

Declaração

- Sintaxe:

```
CREATE INDEX idxCerveja  
    ON Cervejas(empr);
```

```
CREATE INDEX idxVende ON Vende(bar,  
cerveja);
```

Índices

3

Utilização de índices (1)

- **Exemplo:** utilizar idxCerveja e idxVende para encontrar os preços de cervejas produzidas pela UNICER e vendidas no bar Pipa Velha.

Índices

4

Utilização de índices (2)

```
SELECT preco FROM Cervejas, Vende  
WHERE empr = 'UNICER' AND  
Cervejas.nome = Vende.cerveja AND  
bar = 'Pipa Velha';
```

1. Utiliza idxCerveja para obter todas as cervejas feitas pela UNICER.
2. Depois utiliza idxVende para obter os preços das cervejas, com bar = 'Pipa velha'

Índices

5

Afinação da base de dados

- A decisão mais importante a tomar pelo administrador da base de dados para que ela seja convenientemente rápida diz respeito à decisão sobre os índices a criar.
- **Vantagens:** Um índice acelera as consultas que dele tiram proveito;
- **Desvantagens:** Um índice torna mais lentas as modificações na tabela sobre a qual está definido, porque o índice também tem de ser modificado.

Índices

6

Exemplo: afinação da base de dados

- Suponhamos que as únicas coisas que fazemos sobre a base de dados das cervejas é:
 1. Inserir registos novos na tabela Cervejas (10%).
 2. Procurar o preço de uma dada cerveja num dado bar, que se obtém da tabela Vende (90%).
- Assim **idxVende** sobre Vende(bar, cerveja) seria útil, mas **idxCerveja** sobre Cervejas(empr) seria prejudicial.

Índices

7

A criação de índices

- As chaves primárias e candidatas são índices, não sendo por isso necessário criá-los.
- Tabelas com poucos registo não necessitam de índices.
- Aconselha-se a criação de índices nas seguintes situações:
 1. Tabelas grandes das quais se seleciona frequentemente uma pequena percentagem de registo;
 2. Colunas usadas para junção de tabelas (são tipicamente as chaves externas);
 3. Colunas com poucos valores repetidos ou com uma gama de valores grande;
 4. Colunas com poucos valores distintos em tabelas com muitos registo (índices bitmap, mas não existem em todos os SGBD).

Índices

8

Afinação automática (*tuning advisor*) - 1

- Uma ferramenta muito útil.
 - Porque a afinação manual consome muito tempo.
- O aconselhamento é feito a partir do *histórico das consultas*, por ex.:
 1. Escolhe aleatoriamente consultas do histórico das consultas executadas sobre a base de dados, ou
 2. O administrador da base de dados fornece uma amostra de consultas.

Índices

9

Afinação automática (*tuning advisor*) - 2

- O afinador automático gera candidatos a índices e avalia cada um deles na amostra.
 - Executa cada consulta da amostra assumindo que só existe esse índice.
 - Mede a melhoria/degradação do tempo médio de execução das consultas.

Índices

10

EXPLAIN

- EXPLAIN SELECT ...;
- EXPLAIN QUERY PLAN SELECT ...;



Universidade do Porto

Faculdade de Engenharia

FEUP

1

- **INTRODUÇÃO**
- **MODELOS CONCEPTUAIS**
 - Diagrama de Classes UML
 - Modelo Entidade-Associação (E-A)
- **MODELO RELACIONAL**
- **LINGUAGEM DE DEFINIÇÃO DE DADOS**
- **INTERROGAÇÃO DE DADOS**
 - Álgebra relacional
 - **LINGUAGEM DE MANIPULAÇÃO DE DADOS (LMD) SQL**

Observação: baseado em slides desenvolvidos pelo Prof. Gabriel David (FEUP)

Índice

2

- *Persistent Stored Modules (PSM)*
- SQL embebido

PSM

3

SQL na prática

- Até agora viu-se como se usa SQL através de uma interface própria para a execução de consultas à base de dados.
- A realidade costuma ser diferente: programas convencionais interagem com a(s) base(s) de dados através de SQL.

PSM

4

Opções

1. Código escrito em linguagens especializadas é guardado na base de dados (ex:, PL/SQL do SGDB Oracle, SQL PL do DB2, PL/pgPSM do PostgreSQL, ...).
2. Instruções SQL são embedidas numa *linguagem hospedeira* (ex: C).
3. Ferramentas de ligação são utilizadas de forma a permitir a uma linguagem conventional aceder a uma base de dados (ex: CLI, JDBC, PHP/DB).

PSM

5

Procedimentos guardados

- PSM, ou “*persistent stored modules*,” permitem-nos guardar procedimentos como elementos da estrutura da base de dados.
- PSM = mistura de instruções convencionais (if, while, etc.) e SQL.
- Permitem-nos fazer coisas que não se conseguem fazer só com SQL ou fazê-las de forma mais eficiente.

PSM

6

Estrutura básica de PSMs

CREATE PROCEDURE <nome> (

 <lista de parâmetros>)

 <declarações locais opcionais>

 <corpo>;

- Função alternativa:

CREATE FUNCTION <nome> (

 <lista de parâmetros>) RETURNS <tipo>

PSM

7

Passagem de parâmetros em PSMs

- Ao contrário dos pares nome-tipo de linguagens como C, PSM usa tuplos modo-nome-tipo, onde o *modo* pode ser:
 - IN = o procedimento usa valores, não os altera.
 - OUT = o procedimento altera, não os usa.
 - INOUT = ambos.

PSM

8

Exemplo: procedimento guardado

- Vamos escrever um procedimento com dois argumentos c e p , que adiciona um registo a **Vende(bar, cerveja, preco)** com bar = 'Pipa Velha', cerveja = c , e preco = p .
 - Utilizado pelo dono do bar Pipa Velha para adicionar este menu mais facilmente.

PSM

9

O procedimento

```
CREATE PROCEDURE MenuPipaVelha (
```

IN	c	CHAR(20),
IN	p	REAL

)

Ambos os parâmetros são só de leitura, não são modificados.

```
INSERT INTO Vende  
VALUES('Pipa Velha', c, p);
```

O corpo do procedimento:
apenas um INSERT

PSM

10

Invocação de procedimentos

- Usar a instrução CALL, com o nome do procedimento desejado e respectivos argumentos.
- Exemplo:

```
CALL MenuPipaVelha( 'Guinness' , 3.00 );
```
- As funções podem ser usadas em expressões SQL desde que o valor devolvido seja do tipo apropriado.

PSM

11

Tipos de instruções PSM (1)

- RETURN <expressão> define o valor devolvido pela função.
 - Ao contrário de C, etc., RETURN *não* termina a execução da função.
- DECLARE <nome> <tipo> usado para declarar variáveis locais.
- BEGIN . . . END quando há mais do que uma instrução.
 - Separar as instruções por “;”.

PSM

12

Tipos de instruções PSM (2)

- Instruções de atribuição:

SET <variável> = <expressão>;

○ Exemplo: SET c = 'Abadia' ;

- Etiquetas: dar nome às instruções colocando um nome e “:” como prefixo.

PSM

13

Instruções IF

- Forma mais simples:
IF <condição> THEN <instrução(ões)>
END IF;
- Acrescentar ELSE <instrução(ões)> caso se queira,
IF ... THEN ... ELSE ... END IF;
- Acrecentar casos adicionais fazendo ELSEIF
<instrução(ões)>: IF ... THEN ... ELSEIF ... THEN ...
ELSEIF ... THEN ... ELSE ... END IF;

PSM

14

Exemplo: IF (1)

- Vamos classificar os bares em função do seu número de clientes, com base em `Frequenta(cliente,bar)` e com as seguintes regras:
 - <100 clientes: ‘impopular’.
 - 100-199 clientes: ‘média’.
 - ≥ 200 clientes: ‘popular’.
- A função `Classifica(b)` classifica o bar `b`.

PSM

15

Exemplo: IF (2)

```
CREATE FUNCTION Classifica (IN b CHAR(20) )
```

```
    RETURNS CHAR(10)
```

```
DECLARE custo INTEGER;
```

```
BEGIN
```

```
    SET custo = (SELECT COUNT(*) FROM Frequenta  
                 WHERE bar = b);
```

```
    IF custo < 100 THEN RETURN 'impopular'
```

```
    ELSEIF custo < 200 THEN RETURN 'média'
```

```
    ELSE RETURN 'popular'
```

```
END IF;
```

```
END;
```

Número de clientes do bar b

Instrução IF

A devolução do valor da função acontece aqui e não quando a instrução RETURN aparece

PSM

16

Ciclos

- Estrutura básica:
`<nome do ciclo>: LOOP <instruções>
END LOOP;`
- Sai-se de um ciclo fazendo:
`LEAVE <nome do ciclo>`

PSM

17

Exemplo: sair de um ciclo

ciclo1: LOOP

...

LEAVE ciclo1; ← Se esta instrução for executada . . .

...

END LOOP;

← O controlo passa para aqui

Outros tipos de ciclos

- WHILE <condição> DO
<instruções>
END WHILE;
- REPEAT <instruções>
UNTIL <condição> END REPEAT;

PSM

19

Consultas

- Consultas genéricas do tipo SELECT-FROM-WHERE *não* são permitidas em PSM.
- Há três formas de obter os valores de uma consulta:
 1. Consultas que devolvem um só valor podem ser o lado direito de uma instrução de atribuição.
 2. SELECT . . . INTO em consultas que devolvem um só registo.
 3. Cursos.

PSM

20

Exemplo: instrução de atribuição/consulta

- Utilizando a variável local *p* e *Vende(bar, cerveja, preco)*, podemos obter o preço da cerveja Abadia no bar Pipa Velha fazendo:

```
SET p = (SELECT preco FROM Vende  
WHERE bar = 'Pipa Velha' AND  
cerveja = 'Abadia') ;
```

PSM

21

SELECT ... INTO

- Outra forma de obter um valor de uma consulta que devolve um só tuplo consiste em colocar a cláusula **INTO <variável>** depois da cláusula SELECT.
- Exemplo:

```
SELECT preco INTO p FROM Vende  
WHERE bar = 'Pipa Velha' AND  
cerveja = 'Abadia';
```

PSM

22

Cursos

- Um *cursor* é basicamente um índice de uma estrutura constituída pelos tuplos resultado de uma dada consulta.
- Declara-se um cursor *c* fazendo:
DECLARE *c* CURSOR FOR <consulta>;

PSM

23

Abrir e fechar cursores

- Para se usar o cursor c , é necessário fazer:
OPEN c;
 - A consulta de c é avaliada, e c (o cursor) é colocado no primeiro tuplo do resultado.
- Para terminar a utilização de c fazer:
CLOSE c;

Obtenção de tuplos através de cursores

- Para obter o tuplo seguinte do cursor *c*, fazer:
 FETCH FROM *c* INTO *x₁*, *x₂*,...,*x_n* ;
- Os *x*’s são uma lista de variáveis, uma para cada componente do tuplo referenciado por *c*.
- *c* passa automaticamente para o tuplo seguinte.

Finalização de ciclos com cursores (1)

- A forma mais comum de utilizar um cursor é através da criação de um ciclo, usando a instrução FETCH a cada iteração e fazendo alguma operação com o tuplo referenciado.
- Uma questão importante é a de saber como se sai do ciclo quando o cursor chegou ao fim.

Finalização de ciclos com cursores (2)

- Cada operação SQL devolve um *estado*, que é uma *string* com 5 dígitos.
 - Por exemplo, 00000 = “Everything OK,” e 02000 = “Failed to find a tuple.”
- Em PSM, pode-se obter o valor do estado através da variável SQLSTATE.

Finalização de ciclos com cursores (3)

- Podemos declarar uma *condição*, que é verdade se e só se SQLSTATE tiver um dado valor.
- **Exemplo:** Podemos declarar a condição NaoExiste para representar o 2000 através de:

```
DECLARE NaoExiste CONDITION FOR  
SQLSTATE '02000';
```

Finalização de ciclos com cursores (4)

- A estrutura de um ciclo com cursor é:

```
cicloCursor: LOOP
```

```
...
```

```
    FETCH c INTO ... ;  
    IF NaoExiste THEN LEAVE cicloCursor;  
    END IF;  
  
    ...  
END LOOP;
```

PSM

29

Exemplo: cursor

- Vamos escrever um procedimento que examine **Vende(bar, cerveja, preco)**, e que aumente em 1€ o preço de todas as cervejas do bar Pipa Velha cujo preço seja inferior a 3€.
 - É claro que se pode fazer o mesmo usando um simples UPDATE, mas fica aqui como exemplo da utilização de cursores.

PSM

30

Exemplo: cursor

```
CREATE PROCEDURE AumentoPV()
```

```
    DECLARE aCerveja CHAR(20);
```

```
    DECLARE oPreco NUMBER;
```

```
    DECLARE NaoExiste CONDITION FOR
```

```
        SQLSTATE '02000';
```

```
    DECLARE c CURSOR FOR
```

```
        (SELECT cerveja, preco FROM Vende  
         WHERE bar = 'Pipa velha');
```

Usam-se para guardar os valores de cerveja e preço ao percorrer os registos com o cursor c

Devolve o menu do bar Pipa Velha

PSM

31

Exemplo: o corpo do procedimento

```
BEGIN  
    OPEN c;  
    cicloMenu: LOOP  
        FETCH c INTO aCerveja, oPreco;  
        IF NaoExiste THEN LEAVE cicloMenu END IF;  
        IF oPreco < 3.00 THEN  
            UPDATE Vende SET preco = oPreco + 1.00  
            WHERE bar = 'Pipa Velha' AND cerveja = aCerveja;  
        END IF;  
    END LOOP;  
    CLOSE c;  
END;
```

Verifica se o último
FETCH falhou na
obtenção de um tuplo

Se a Pipa Velha cobrar menos de 3€
pela cerveja, o preço aumenta 1€
(no bar Pipa Velha, claro).

SQL embebido

32

- **Ideia chave:** Um pre-processador transforma instruções SQL em chamadas de procedimentos compatíveis com o código da linguagem hospedeira.
- Todas as instruções SQL embebidas começam com EXEC SQL, para que o pre-processador as possa encontrar mais facilmente.

SQL embedido

33

Variáveis partilhadas

- Para ligar SQL e programas em linguagens hospedeiras, as duas partes têm de partilhar algumas variáveis.
- Declarações de variáveis partilhadas são delimitadas por:

 EXEC SQL BEGIN DECLARE SECTION;

É sempre
necessário

<declarações na linguagem hospedeira>

 EXEC SQL END DECLARE SECTION;

SQL embbebido

34

Utilização de variáveis partilhadas

- Em SQL as variáveis partilhadas têm de ser precedidas por “:”.
 - Podem ser usadas como constantes providenciadas pelo programa em linguagem hospedeira.
 - Os seus valores podem ser obtidos através de instruções SQL e serem posteriormente passados para o programa em linguagem hospedeira.
- Na linguagem hospedeira, as variáveis partilhadas comportam-se como qualquer outra variável.

SQL embebido

35

Exemplo: consulta de preços

- Vamos usar C com SQL embebido para exemplificar quais as partes importantes de uma função que obtém uma cerveja e um bar, e consulta o preço dessa cerveja nesse bar.
- Assume-se qua a base de dados tem a tabela **Vende(bar, cerveja, preco)**, como de costume.

SQL embestado

36

Exemplo: C e SQL

```
EXEC SQL BEGIN DECLARE SECTION;
char oBar[21], aCerveja[21]; ← Notar que são precisos
float oPreco;                  arrays de 21-char para
                                20 chars + marca de fim
EXEC SQL END DECLARE SECTION;
/* obtém valores para oBar e aCerveja */
EXEC SQL SELECT preco INTO :oPreco
FROM Vende
WHERE bar = :oBar AND cerveja = :aCerveja;
/* faz qualquer coisa com oPreco */  $\wedge$  SELECT-INTO
                                         como em PSM
```

SQL embebido

37

Consultas embebidas

- SQL embebido tem as mesmas limitações que PSM no que diz respeito a consultas:
 - SELECT-INTO para uma consulta que garantidamente dê um único tuplo como resultado.
 - Caso contrário, tem de se usar cursores.
 - ✖ Pequenas diferenças sintáticas, mas as mesmas ideias chave.

SQL embebido

38

Consultas embebidas

- Declarar um cursor *c* com:

EXEC SQL DECLARE *c* CURSOR FOR <consulta>;

- Abrir e fechar o cursor *c* com:

EXEC SQL OPEN CURSOR *c*;

EXEC SQL CLOSE CURSOR *c*;

- Obter valores a partir de *c* com:

EXEC SQL FETCH *c* INTO <variavel(is)>;

- Macro NOT FOUND é TRUE se e só se o FETCH falha na procura de um tuplo.

SQL embbebido

39

Exemplo: imprimir o menu de Pipa Velha

- Vamos usar C + SQL para imprimir o menu de Pipa Velha – a list de pares cerveja-preco existentes em **Vende(bar, cerveja, preco)** com bar = Pipa Velha.
- Um cursor vai percorrer os tuplos da tabela Vende com bar = ‘Pipa Velha’.

SQL embedido

40

Exemplo: declarações

```
EXEC SQL BEGIN DECLARE SECTION;  
    char aCerveja[21]; float oPreco;  
EXEC SQL END DECLARE SECTION;  
EXEC SQL DECLARE c CURSOR FOR  
    SELECT cerveja, preco FROM Vende  
    WHERE bar = 'Pipa Velha';
```

O cursor é declarado fora
da secção de declaração

SQL embestado

41

Exemplo: parte executável

```
EXEC SQL OPEN CURSOR c;
```

```
while(1) {
```

```
    EXEC SQL FETCH c
```

```
        INTO :aCerveja, :oPreco;
```

```
        if (NOT FOUND) break;
```

```
        /* formatar e imprimir aCerveja e oPreco */
```

```
}
```

```
EXEC SQL CLOSE CURSOR c;
```

A forma como se
sai de ciclos em C

SQL embbebido

42

A necessidade de SQL dinâmico

- A maior parte das aplicações usam instruções pré-definidas para interagir com a base de dados.
 - O SGBD compila instruções EXEC SQL ... para chamadas de procedimentos específicos e produz um programa na linguagem hospedeira que utiliza uma biblioteca.
- E quanto ao sqlplus, que não sabe do que precisa até ser executado?
 - As instruções só são conhecidas no momento em que são executadas.

SQL embedido

43

SQL dinâmico

- Preparação de uma consulta:

EXEC SQL PREPARE <nome da consulta>
 FROM <texto da consulta>;

- Execução da consulta:

EXEC SQL EXECUTE <nome da consulta>;

- “Prepare” = optimizar consulta.

- Prepara uma vez, executa muitas vezes.

SQL embbebido

44

Exemplo: uma interface genérica

```
EXEC SQL BEGIN DECLARE SECTION;
char consulta[MAX_LENGTH];
EXEC SQL END DECLARE SECTION;
while(1) {
    /* Disponibiliza o prompt para introdução da consulta */
    /* lê a consulta escrita pelo utilizador para um array */
    EXEC SQL PREPARE c FROM :consulta;
    EXEC SQL EXECUTE c;
}
```

c é uma variável SQL que representa a forma optimizada de uma qualquer instrução escrita pelo utilizador e que se encontra em :consulta

SQL embbebido

45

Execução imediata

- Se só se vai executar a consulta uma vez, podemos combinar os passos PREPARE e EXECUTE num só.
- Usar:

```
EXEC SQL EXECUTE IMMEDIATE <texto>;
```

SQL embedido

46

Exemplo: outra vez a interface genérica

```
EXEC SQL BEGIN DECLARE SECTION;
char consulta[MAX_LENGTH];
EXEC SQL END DECLARE SECTION;
while(1) {
    /* Disponibiliza o prompt */
    /* Lê a consulta para um array*/
    EXEC SQL EXECUTE IMMEDIATE :consulta;
}
```

Bases de Dados



Universidade do Porto

Faculdade de Engenharia

FEUP

1

- **INTRODUÇÃO**
- **MODELOS CONCEPTUAIS**
 - Diagrama de Classes UML
 - Modelo Entidade-Associação (E-A)
- **MODELO RELACIONAL**
- **LINGUAGEM DE DEFINIÇÃO DE DADOS**
- **INTERROGAÇÃO DE DADOS**
 - Álgebra relacional
 - Linguagem de Manipulação de Dados (LMD)

Índice

2

- Transações
- Controlo de concorrência
- Recuperação

Transações

3

Contexto

- Numa arquitectura cliente/servidor (sobretudo mas não só) a execução de um mesmo programa (ou de programas diferentes) por parte de um ou mais utilizadores pode pôr em risco a consistência da informação contida na BD.
 - *Exemplo: O utilizador U1 lê os dados da conta de um cliente e credita 5€ mas entre o momento de leitura do valor e o momento de escrita do novo valor na BD, o utilizador U2 lê os dados da conta do mesmo cliente e debita 2€. Neste caso, quando U1 escreve o valor que leu adicionado de 5€ na BD, o débito feito por U2 perde-se.*
- Para além disso, interrupções imprevistas na execução de um programa podem também originar inconsistências na BD.
 - *Exemplo: Numa transferência bancária, a quebra de energia eléctrica pode fazer com que o montante a transferir tenha sido debitado na conta de origem mas não tenha sido creditado na conta destino.*

transações

4

Definição

- ***transação*** = uma unidade lógica de trabalho que envolve diversas operações sobre a base de dados.
- São uma sequência de instruções SQL.
- São definidas de forma:
 - ***implícita***, a partir de uma instrução SQL (por defeito), ou
 - ***explícita***, por parte do programador através de instruções próprias para o efeito (a ver mais à frente), envolvendo várias instruções SQL.

transações

5

Propriedades

As transações devem ser **ACID**:

- **Atómicas**: cada uma das instruções da transação só toma efeito se todas tomarem efeito
- **Consistentes**: a execução de uma transação isoladamente não coloca a BD inconsistente (esta propriedade é da responsabilidade do programador)
- **Isoláveis**: a execução de uma transação em concorrência produz o mesmo efeito da execução isolada
- **Duráveis**: quando uma transação é dada como bem sucedida, os efeitos devem perdurar na BD mesmo que haja falhas de sistema.

transações

6

Escalonamento

Estabelece o plano de execução das transações:

- Cada transação é descrita pela sequência de acções
- Cada acção é de um destes quatro tipos:
 - Read: leitura da BD
 - Write: escrita na BD
 - Commit: ação que finaliza uma transação bem sucedida guardando de forma persistente as alterações feitas
 - Abort: ação que finaliza uma transação mal sucedida descartando as alterações feitas pela transação e repondo a BD num estado de consistência
- A lista de ações é obtida pelo SGBD a partir das instruções SQL. Não é da responsabilidade do programador defini-la.

transações

7

Escalonamento

- Notação:
 - **READ**: Leitura
 - **WRITE**: Escrita
 - **COMMIT**: Cometimento
 - **ABORT**: Cancelamento
 - Entre parêntesis coloca-se o objecto da BD afetado pela operação
- Uma transação é finalizada por um **COMMIT** ou por um **ABORT**:
 - Tanto um como o outro não têm argumentos
 - Estas duas ações são muitas vezes omitidas na representação do escalonamento

Exemplo	
T ₁	T ₂
READ(A)	
WRITE(A)	
	READ(B)
	WRITE(B)
READ(C)	
WRITE(C)	

transações

8

Escalonamento

- Os escalonamentos que contêm um *commit* ou um *abort* por cada transação diferente que contenha, designam-se por **escalonamentos completos**
- Os escalonamentos que não têm ações alternadas de diferentes transações, i.e., cada transação só começa quando a transação precedente termina, designam-se por **escalonamentos em série** (ou **seriais**)

transações

9

Motivação para escalonamentos alternados

- Quando os escalonamentos são em série, algumas transações, especialmente as mais pequenas, podem ter tempos de resposta inesperadamente grandes por serem precedidas por transações grandes
- Permitindo, de forma controlada, a execução alternada de acções de diferentes transações, é possível aumentar o nível de satisfação global dos utilizadores das BDs
- Isso é potenciado pela possibilidade de executar actividades de CPU e de I/O (i.e., de leitura/escrita na BD), em paralelo

transações

10

Definição informal de escalonamento serializável

- Um escalonamento diz-se **serializável** quando o resultado da sua execução é idêntico ao resultado de uma das possíveis execuções em série das transações que o compõem.
 - É óbvio que a ordem pela qual as transações são executadas (em série) pode afetar o resultado final
 - A execução serializável deve garantir a equivalência para com uma das possíveis execuções em série, não sendo possível determinar a priori qual delas

transações

11

Possíveis problemas dos escalonamentos alternados (1)

- Conflito WR¹: leitura de dados não cometidos
- Conhecido por **leitura suja** (*dirty read*)
- Exemplo:
 - (1) T1 transfere 100€ de A para B começando por debitar o valor de A; (2) T2 incrementa tanto A como B em 2% (juros anuais); e (3) T1 finaliza a transação creditando o valor (entretanto aumentado) em B.

¹lê-se: T_j faz R após T_i ter feito W sobre o mesmo objecto

T1	T2
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
	READ(B)
	WRITE(B)
	COMMIT
READ(B)	
WRITE(B)	
	COMMIT

transações

12

Possíveis problemas dos escalonamentos alternados (2)

- Conflito RW: escrita de dados entretanto lidos por outra transação
- Conhecido por **leitura irrepetível**
- Exemplo:
 - (1) T1 lê a quantia disponível em A; (2) T2 lê e credita 20€ em A; e (3) T1 finaliza a transação creditando o valor de 100€ em A.

T1	T2
READ(A)	
	READ(A)
	WRITE(A)
	COMMIT
WRITE(A)	
COMMIT	

transações

13

Possíveis problemas dos escalonamentos alternados (3)

- Conflito WW: reescrita de dados não cometidos
- Conhecido por **escrita cega** (*blind write*)
- Exemplo:
 - (1) T1 escreve 100 em A; (2) T2 escreve 200 em B; (3) T1 escreve 100 em B; e (4) T2 escreve 200 em A.
 - O resultado deste escalonamento não é idêntico a nenhuma das duas possíveis execuções em série. É só testar ...

T1	T2
WRITE(A)	
	WRITE(B)
WRITE(B)	
	WRITE(A)
COMMIT	
	COMMIT

transações

14

Exercícios

- Considere as seguintes acções da transação T1 sobre os objectos A e B:
 - READ(A), WRITE(A), READ(B), WRITE(B)
- a) Dê um exemplo de uma outra transação T2 que, se executada em concorrência com T1 sem nenhuma forma de controlo de concorrência, possa interferir com T1.

transações

15

Exercícios

T2: READ(B), WRITE(B)

T1: READ(A); A:=2*A; READ(B); B:=A+B

T2: READ(A); B:=2*A

Exemplo para A=100 e B=10

Escalonamento alternado

T1	T2	A	B
READ(A)		100	
WRITE(A)		$2^*A=200$	
	READ(A)	200	
	WRITE(B)		$2^*A=400$
READ(B)			400
WRITE(B)			$A+B=600$

Escalonamento em série (T1;T2)

T1	T2	A	B
READ(A)		100	
WRITE(A)		$2^*A=200$	
READ(B)			10
WRITE(B)			$A+B=210$
	READ(A)	200	
	WRITE(B)		$2^*A=400$

Escalonamento em série (T2;T1)

T1	T2	A	B
	READ(A)	100	
	WRITE(B)		$2^*A=200$
READ(A)		100	
WRITE(A)		$2^*A=200$	
READ(B)			200
WRITE(B)			$A+B=400$

transações

16

Definição formal de escalonamento serializável

- Até agora não se considerou a possibilidade de uma transação ser cancelada. Ver exemplo ...

- Um **escalonamento serializável** sobre um conjunto de transações S é um escalonamento cujo efeito sobre qualquer BD consistente é garantidamente idêntico ao de um escalonamento em série completo sobre o conjunto de transações cometidas de S .

T1	T2	Exemplo
READ(A)		
WRITE(A)		A:=A-100
	READ(A)	
	WRITE(A)	A:=1.02*A
	READ(B)	
	WRITE(B)	B:=1.02*B
	COMMIT	
ABORT		

transações

17

Escalonamento recuperável

- Um escalonamento diz-se **recuperável** se cada transação que o compõe é cometida só depois de (e se) todas as transações cujas alterações foram lidas por essa transação terem sido cometidas
- No último exemplo dado, o escalonamento era irrecuperável.
- Num escalonamento **recuperável** é possível cancelar a transação sem ter de o fazer em cascata.

transações

18

Possíveis problemas ao cancelar uma transação

- Quando uma transação é cancelada, deve-se colocar a BD num estado consistente, desfazendo as ações até então executadas
- Exemplo de problemas ao cancelar uma transação:
 - Neste exemplo, ao cancelar T1, o valor de A é reposto a 5, perdendo-se as alterações feitas por T2, mesmo que o COMMIT seja feito antes do ABORT
 - Este problema pode acontecer em conflitos WW
 - Para o evitar são necessários mecanismos de recuperação complexos

T1	T2	Exemplo
READ(A)		A:=5
WRITE(A)		A:=6
	WRITE(A)	A:=7
ABORT		
	COMMIT	

Contr. de concorrência c/ bloqueios

19

Protocolo de bloqueios

- Um **protocolo de bloqueios** é um conjunto de regras que cada transação deve seguir (e que são asseguradas pelo SGBD) de forma a que o escalonamento definido seja serializável

Contr. de concorrência c/ bloqueios

20

Protocolo em duas fases estrito

- As duas fases são:
 1. Se uma transação T quer ler ou modificar um objecto, tem de previamente requerer respetivamente um bloqueio partilhado ou exclusivo sobre o objecto;
 - Os bloqueios partilhados só permitem ler
 - Os bloqueios exclusivos permitem ler e escrever
 - Uma transação que requer um bloqueio fica em espera até que o SGBD possa executá-lo
 2. Todos os bloqueios requeridos por uma transação só são libertados ^{j1} quando a transação termina.
- O cumprimento do protocolo em duas fases estrito garante escalonamentos recuperáveis

Diapositivo 20

j1

Ver se está certo. Isso não é só o estrito?

jmoreira; 04-07-2012

Contr. de concorrência c/ bloqueios

21

Protocolo em duas fases estrito

T1	T2
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
	READ(B)
	WRITE(B)
	COMMIT
READ(B)	
WRITE(B)	
COMMIT	

- Notação:
 - **RLOCK**: Bloqueio partilhado
 - **WLOCK**: Bloqueio exclusivo
 - **UNLOCK**: Desbloqueio
 - Entre parêntesis coloca-se o objecto da BD afectado pela operação
- O exemplo do lado esquerdo ficaria ...
 - Neste caso resultaria num escalonamento em série, mas nem sempre é assim.

T1	T2
WLOCK(A)	
READ(A)	
WRITE(A)	
WLOCK(B)	
READ(B)	
WRITE(B)	
COMMIT	
UNLOCK(A)	
UNLOCK(B)	
	WLOCK(A)
	READ(A)
	WRITE(A)
	WLOCK(B)
	READ(B)
	WRITE(B)
	COMMIT
	UNLOCK(A)
	UNLOCK(B)

Contr. de concorrência c/ bloqueios

22

Protocolo em duas fases

- A diferença do protocolo em duas fases para o protocolo em duas fases estrito é:
 - A segunda regra é substituída por: uma transação não pode requerer bloqueios adicionais se já tiver feito algum desbloqueio;
 - ✖ Ou seja, a diferença para a versão estrita é que a segunda fase, apesar de só conter desbloqueios, é feita de forma progressiva, em vez de desbloquear tudo só no fim
 - ✖ O cumprimento do protocolo em duas fases não garante escalonamentos recuperáveis

Diapositivo 22

j2

Rever isto.

jmoreira; 04-07-2012

Contr. de concorrência c/ bloqueios

23

Equivalência em conflitos

- Diz-se que dois escalonamentos têm **equivalência de conflitos** se compreendem as ações das mesmas transações e ordenam cada par de ações conflituosas de duas transações cometidas da mesma forma
 - Lembrar que há três tipos de acções conflituosas: WR, RW e WW
 - O resultado de um escalonamento só depende da ordem das ações conflituosas
- Um escalonamento diz-se **serializável por conflitos** se tem **equivalência de conflitos** com algum escalonamento em série
- Qualquer escalonamento **serializável por conflitos** é serializável, sob o pressuposto que não se insere nem se apaga registo da BD (mas pode-se alterar valores)
 - O contrário pode não se verificar

Contr. de concorrência c/ bloqueios

24

Grafo de precedências

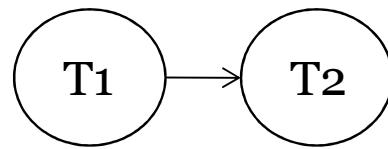
- O **grafo de precedências** para o escalonamento S contém
 - Um nó por cada transação cometida em S
 - Um arco de T_i para T_j se uma acção de T_i precede e entra em conflito com uma acção de T_j
- Um escalonamento S é **serializável por conflitos** se e só se o seu grafo de precedência for acíclico
 - Qualquer escalonamento que assegure o bloqueio em duas fases ou em duas fases estrito tem um grafo de precedências acíclico

Contr. de concorrência c/ bloqueios

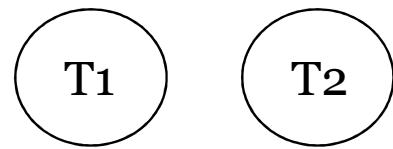
25

Exemplos: grafo de precedências

T ₁	T ₂
WLOCK(A)	
READ(A)	
WRITE(A)	
WLOCK(B)	
READ(B)	
WRITE(B)	
COMMIT	
	WLOCK(A)
	READ(A)
	WRITE(A)
	WLOCK(B)
	READ(B)
	WRITE(B)
	COMMIT



T ₁	T ₂
RLOCK(A)	
READ(A)	
	RLOCK(A)
	READ(A)
	WLOCK(B)
	READ(B)
	WRITE(B)
	COMMIT
WLOCK(C)	
READ(C)	
	WRITE(C)
	COMMIT

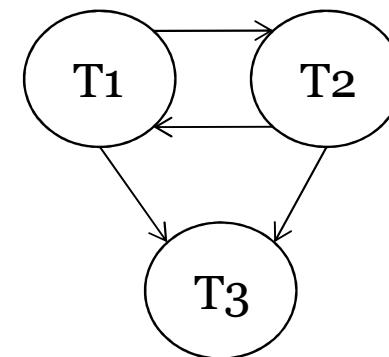


Contr. de concorrência c/ bloqueios

26

Exemplos: grafo de precedências

T1	T2	T3
READ(A)		
	WRITE(A)	
		COMMIT
WRITE(A)		
COMMIT		
		WRITE(A)
		COMMIT



Contr. de concorrência c/ bloqueios

27

Gestor de bloqueios

- A componente do SGBD que guarda informação dos bloqueios existentes designa-se por **gestor de bloqueios**
- O **gestor de bloqueios** mantém uma tabela de bloqueios
 - Cada registo dessa tabela contém o número de transações que têm um bloqueio sobre um dado objecto, a natureza desse bloqueio (partilhado ou exclusivo) e um apontador para a lista de pedidos de bloqueio
 - O número de transações que têm um bloqueio sobre um dado objecto pode ser maior do que um se o bloqueio for partilhado

Contr. de concorrência c/ bloqueios

28

Pedidos de bloqueio ou de desbloqueio

- Os bloqueios podem ser feitos sobre objetos de diferentes tipos (dependendo do SGBD): páginas, registos, ...
 - As páginas são as unidades elementares para acesso a disco. A identificação da página onde se encontra cada registo é feita pelo gestor de ficheiros do SGBD
- Quando uma transação faz um pedido de bloqueio sobre um objeto ao gestor de bloqueios:
 - Se é um bloqueio partilhado, se a fila de pedidos sobre esse objecto está vazia e se o objeto não se encontra bloqueado por um bloqueio exclusivo, o gestor de bloqueios executa o bloqueio e actualiza a tabela de bloqueios
 - Se é um bloqueio exclusivo e não há nenhum bloqueio sobre esse objeto (o que implica não existir nenhum pedido na fila), o bloqueio é executado e a tabela de bloqueios é atualizada
 - Caso contrário, o pedido não é concedido, sendo adicionado à fila de pedidos de bloqueio desse objecto, e a transação fica suspensa
- Quando um bloqueio sobre um objeto é libertado, o próximo pedido na fila de bloqueios desse objeto é avaliado

Contr. de concorrência c/ bloqueios

29

Possíveis problemas dos protocolos de bloqueios

- Bloqueio ativo (*livelock*):
 - T_1 e T_2 pedem bloqueio de A; T_1 obtém; T_3 pede bloqueio de A; quando T_1 desbloqueia, T_3 é servido e T_2 fica à espera...
 - Hipótese de resolução: servir sempre o pedido mais antigo. Fácil.
- Encravamento (*deadlock*):
 - Seja:
 - ✖ T_1 : WLOCK(A); WLOCK(B); UNLOCK(A); UNLOCK(B);
 - ✖ T_2 : WLOCK(B); WLOCK(A); UNLOCK(B); UNLOCK(A).
 - Caso o escalonamento seja feito, por exemplo, pela ordem:
 - ✖ $\text{WLOCK}_{T_1}(A)$; $\text{WLOCK}_{T_2}(B)$; $\text{WLOCK}_{T_1}(B)$; $\text{WLOCK}_{T_2}(A)$; ...
 - Encrava ...
 - Hipótese de resolução: complexo ... Tema não lecionado.
- O problema fantasma: Tema não lecionado.

Contr. de concorrência c/ bloqueios

30

transações SQL-92

- As transações são iniciadas de forma automática quando o utilizador executa uma instrução que modifica a base de dados ou o catálogo
- As transações terminam:
 - Explicitamente com uma das instruções:
 - COMMIT: faz o cometimento da transação
 - ROLLBACK: cancela a transação
 - Implicitamente: quando é executada uma instrução LDD-SQL

Contr. de concorrência c/ bloqueios

31

transações SQL-92

- Características das transações:
 - Dimensão do diagnóstico: especifica o número de erros que podem ser registados na área de diagnóstico
 - Modo de acesso:
 - ✖ READ ONLY: só permite leituras
 - ✖ READ WRITE: permite leituras e escritas
 - Nível de isolamento: controla o quanto uma transação fica exposta a outras transações concorrentes, ou seja, permite dosear entre escalonamentos mais alternados e o risco de não garantir a seriabilidade desse escalonamento. Existem quatro níveis de isolamento:
 - ✖ Serializável (*serializable*)
 - ✖ Leitura repetível (*repeatable read*)
 - ✖ Leitura cometida (*read committed*)
 - ✖ Leitura não cometida (*read uncommitted*)

Contr. de concorrência c/ bloqueios

32

transações SQL-92

- O nível de isolamento **serializável** assegura que a transação T:
 - Só lê alterações feitas por transações já cometidas;
 - Nenhum valor lido ou escrito por T é alterado por outra transação até T terminar;
 - O conjunto de valores correspondente a um dado critério de pesquisa e que tenha sido lido por T, não pode ser alterado por nenhuma transação até T completar (ou seja, evita o problema fantasma).

Na implementação baseada em bloqueios, uma transação serializável faz bloqueios antes de ler ou escrever nos objetos, incluindo bloqueios em conjuntos de objetos de forma a evitar o problema fantasma

Contr. de concorrência c/ bloqueios

33

transações SQL-92

- O nível de isolamento **leitura repetível** assegura que a transação T:
 - Só lê alterações feitas por transações já cometidas;
 - Nenhum valor lido ou escrito por T é alterado por outra transação até T terminar.

A única diferença entre os níveis de isolamento serializável e leitura repetível é que este último não faz bloqueio aos índices, ou seja pode bloquear objetos mas não garante o bloqueio a todos os objetos associados (dinamicamente) a um critério de pesquisa

Contr. de concorrência c/ bloqueios

34

transações SQL-92

- O nível de isolamento **leitura cometida** assegura que a transação T:
 - Só lê alterações feitas por transações já cometidas;
 - Nenhum valor escrito por T é alterado por outra transação até T terminar sendo, no entanto, possível que um valor lido por T seja modificado por outra transação antes de T terminar.

A única diferença entre os níveis de isolamento leitura repetível e leitura cometida é que, neste último, os bloqueios de leitura são libertados logo após a leitura. Os bloqueios de escrita, em ambas as situações, só são libertados no fim.

Contr. de concorrência c/ bloqueios

35

transações SQL-92

- O nível de isolamento **leitura não cometida** assegura que a transação T:
 - Pode ler alterações feitas a um objeto por uma transação em curso

Na leitura não cometida não são efetuados bloqueios partilhados antes da leitura dos objetos. Devido ao elevado nível de exposição a alterações não cometidas, a SQL só permite associar este nível de isolamento a transações com modo de acesso READ ONLY. Sendo só de leitura e não fazendo bloqueios de leitura, significa que não faz qualquer tipo de bloqueio

Contr. de concorrência c/ bloqueios

36

transações SQL-92

- Resumo:

Nível	Leitura suja	Leitura irrepetível	Fantasma
Serializável	Não	Não	Não
Leitura repetível	Não	Não	Eventualmente
Leitura cometida	Não	Eventualmente	Eventualmente
Leitura não cometida	Eventualmente	Eventualmente	Eventualmente

Contr. de concorrência c/ bloqueios

37

transações SQL-92

- O nível de isolamento serializável é o mais seguro, sendo o mais adequado para a maioria das situações. No entanto, se outro nível de isolamento for adequado para uma dada transação, a sua utilização poderá melhorar o desempenho do sistema
 - Exemplo: utilizando o nível de isolamento leitura cometida ou leitura não cometida, é possível calcular a média de idades dos alunos da FEUP de forma mais eficiente e sem distorcer, de forma significativa, o valor verdadeiro, pois a perda ou ganho de um ou dois valores não alterará significativamente a média

Contr. de concorrência c/ bloqueios

38

transações SQL-92

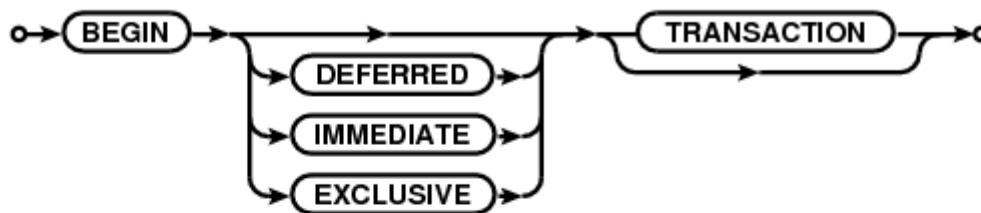
- Instrução para definir o modo de acesso e o nível de isolamento de uma transação: SET TRANSACTION
- Usando o exemplo anterior:
 - SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
READ ONLY
- Quando uma transação se inicia, os valores por defeito para o nível de isolamento e o modo de acesso são, respetivamente, SERIALIZABLE e READ WRITE

Contr. de concorrência c/ bloqueios

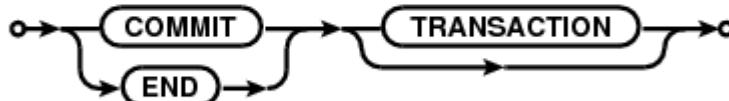
39

transações em SQLite

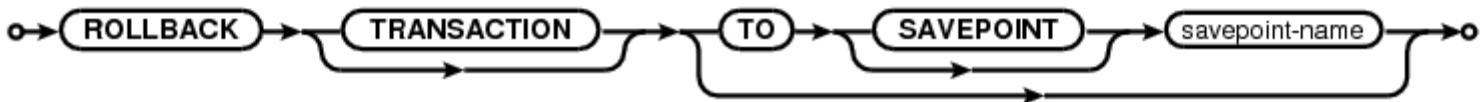
begin-stmt:



commit-stmt:



rollback-stmt:



Contr. de concorrência c/ bloqueios

40

transações em SQLite

- Modos:
 - **Deferred**: É o modo usado por defeito. *Deferred* significa que nenhum *LOCK* é feito até à à primeira operação de leitura ou escrita. Ou seja, a declaração *BEGIN* em si não faz nada. A primeira operação de leitura da BD cria um bloqueio partilhado e a primeira operação de gravação cria um bloqueio exclusivo;
 - **Immediate**: Neste caso, bloqueios exclusivos são feitos logo após o *BEGIN*. Depois de um *BEGIN IMMEDIATE*, nenhum outro processo pode escrever na base de dados nem fazer *BEGIN IMMEDIATE* ou *BEGIN EXCLUSIVE*. Mas podem, no entanto fazer leitura da base de dados;
 - **Exclusive**: Após um *BEGIN EXCLUSIVE*, nenhuma outra ligação à base de dados, excepto ligações *read_uncommitted* poderão ler a base de dados e nenhum outro processo, sem excepção, poderá escrever na base de dados até a transacção estar completa.

Contr. de concorrência s/ bloqueios

41

- A utilização de protocolos de bloqueio para controlo de concorrência é uma abordagem pessimista na medida em que cancelam transações ou bloqueiam objectos de forma preventiva para resolver conflitos. Em sistemas com níveis baixos de concorrência, o custo deste tipo de abordagens pode ser significativo.
- Existem protocolos alternativos que optam por, mais do que prevenir os conflitos, tentar resolvê-los. Parte-se do princípio que os conflitos são eventos raros e, por isso, o custo de resolução, sendo elevado por cada conflito, tem um custo global baixo.

Contr. de concorrência s/ bloqueios

42

Controlo de concorrência baseada em marcas temporais

- Princípios do controlo de concorrência por marcas temporais:
 1. É atribuída a cada transação T_i uma marca temporal $TS(T_i)$ quando T_i é iniciada
 2. Assegurar durante a execução que, no caso de haver conflito entre a acção A_i de T_i e a acção A_j de T_j , A_i ocorre antes de A_j se $TS(T_i) < TS(T_j)$
 3. Se uma acção viola esta ordenação, a transação é cancelada e reiniciada
- Para implementar esta forma de controlo de concorrência, define-se, para cada objecto (O), a marca temporal da transação que fez a última leitura e a última escrita, respetivamente $TL(O)$ e $TE(O)$.

Contr. de concorrência s/ bloqueios

43

Controlo de concorrência baseada em marcas temporais

- Recuperabilidade:
 - Infelizmente, o protocolo de marcas temporais permite escalonamentos que não são recuperáveis (ver exemplo).
 - Este problema pode ser resolvido modificando o protocolo de marcas temporais de forma a não permitir estes escalonamentos, passando todas as ações de escrita para *buffer* até a transação cometer.
 - No exemplo, quando T1 quer escrever A, TE(A) é atualizado, mas a alteração de A só é registada no *buffer*. Quando T2 quer, a seguir, ler A, T2 é bloqueada até T1 completar (depois de TS(T2) ser comparada com TE(A) e a leitura ser vista como permissiva). Se T1 cometer, a alteração que fez a A é copiada do *buffer*; caso contrário, as alterações que constam no *buffer* são descartadas.

T1	T2
WRITE(A)	
	READ(A)
	WRITE(B)
	COMMIT

Recuperação

44

Gestor de recuperações

- O gestor de recuperações de um SGBD é responsável por assegurar:
 - Atomicidade: desfazendo as ações das transações ainda não cometidas quando uma transação é cancelada
 - Durabilidade: garantindo que todas as ações das transações cometidas perduram a falhas do sistema
- Guarda num jornal (*log*) todas as alterações feitas à BD

Recuperação

45

Causas para a necessidade de recuperação

- Existem diferentes causas para que seja necessário iniciar um processo de recuperação:
 - Falha lógica de sistema (falha de energia, *core dump*, ...)
 - Falha física do sistema de armazenamento (disco, ...)
 - Interação de outros algoritmos com o controlo de concorrência

Recuperação

46

O jornal

- Contém o historial das ações executadas pelo SGBD.
- É importante minimizar a possibilidade de ele perder informação. Exemplo: escrevendo duas cópias em discos diferentes e, porque não, em locais diferentes.
- A componente mais recente do jornal é mantida na memória principal e guardada periodicamente em local seguro.
- A cada entrada do jornal é atribuída um número sequencial crescente (i.e., números maiores correspondem a entradas mais recentes), designado *Last Sequence Number (LSN)*.
- Para facilitar a recuperação, cada página da BD tem o número sequencial da última ação de alteração feita sobre essa página, designado por *pageLSN*.

Recuperação

47

O jornal

- Acrescenta-se uma entrada ao jornal nas seguintes situações:
 - Página alterada: acrescenta-se um registo do **tipo alteração**. O page LSN é atualizado com o valor do LSN do novo registo.
 - Transação cometida: acrescenta-se um registo do **tipo cometimento**. De seguida, guarda-se em disco a componente do jornal mantida em memória, incluindo a entrada referente ao cometimento. O cometimento fica, assim, terminado, mas há ainda outros passos a realizar (ex.: remover a transação da tabela de transações).
 - Transação cancelada: acrescenta-se um registo do **tipo cancelamento**. De seguida, a ação desfazer é iniciada.
 - Transação finalizada: acrescenta-se um registo do **tipo fim**. Este registo só é criado após terem sido realizadas todas as ações efectuadas após o cancelamento ou o cometimento de uma transação (já referidas anteriormente).
 - Alteração desfeita: acrescenta-se um registo do **tipo compensação** quando a ação descrita por um registo do tipo alteração é desfeita. Tal pode ocorrer após cancelamento de uma transação ou após recuperação devido a falha.

Recuperação

48

O jornal

- Atributos comuns a todos os registo (seja qual for o tipo):
 - *prevLSN*: *LSN* do registo anterior referente à mesma transação;
 - *transID*: *ID* da transação que gerou o registo;
 - *tipo*: tipo do registo (alteração, cometimento, cancelamento, fim ou compensação).
- Alguns dos atributos adicionais dos registo do tipo ‘alteração’:
 - *pageID*: *ID* da página alterada.
 - *before_image*: valor antes da alteração. Permite desfazer uma alteração.
 - *after_image*: valor após a alteração. Permite refazer uma alteração.
- Atributo adicional dos registo de ‘compensação’:
 - *before_image*:
 - *undoNextLSN*: contém o *LSN* do próximo registo do jornal a desfazer. Este atributo terá o valor de *prevLSN* do registo da alteração a ser desfeita.

Recuperação

49

Ponto de verificação

- Até onde percorrer o jornal numa recuperação?
- Operação de verificação (*checkpoint*)
 - proibir o início de transações e esperar que as activas estejam cometidas ou abortem
 - copiar os blocos alterados na memória central para memória estável
 - registar no jornal o ponto de verificação e escrever o jornal
- Só é necessário analisar o jornal até ao último ponto de verificação, para recuperar de uma falha
 - a parte anterior do jornal só interessa para arquivo de segurança

Recuperação

50

Algoritmo de recuperação ARIES

- **ARIES** (Aggregate Recoverable Item Evaluation System)
- Quando o gestor de recuperações é invocado após falha do sistema, a reinicialização é feita em três fases:
 - **Analisar**: identifica páginas sujas do *buffer* (i.e., que não tenham sido escritas em disco) e transações ativas no momento da falha do sistema
 - **Refazer**: repete todas as ações desde um determinado ponto do jornal, designado por ponto de verificação (*checkpoint*), repondo a base de dados no estado em que se encontrava antes da falha do sistema.
 - ✖ **ATENÇÃO**: As transações não são re-executadas! O que acontece é que são atribuídos aos objetos os valores consequentes das ações. Exemplo: Se a ação foi a de atribuir ao objeto A o valor de $A + 5$, e se o resultado dessa operação foi 20, ao refazer, atribui-se a A o valor de 20, de acordo com a informação do jornal, mas a operação não é feita de novo, só o valor é que é reposto.
 - **Desfazer**: desfaz as ações das transações que não foram cometidas

Recuperação

51

Exemplo: algoritmo de recuperação ARIES

LSN	Jornal
1	Update: T1 writes P5
2	Update: T2 writes P3
3	T2 commit
4	T2 end
5	Update: T3 writes P1
6	Update: T3 writes P3
x	CRASH, RESTART

- **Analisar:** identifica as transações que estavam activas quando o sistema falhou, T1 e T3; identifica as transações cometidas, T2; e identifica as páginas que possam estar sujas, P1, P3 e P5.
- **Refazer:** repõe os valores que estão guardados no jornal, pela ordem registada no jornal, para as transações T1, T2 e T3.
- **Desfazer:** finalmente, as acções T1 e T3 são desfeitas, nomeadamente, as escritas de P3 por T3, de P1 por T3 e de P5 por T1.

LSN: Log Sequence Number

Recuperação

52

Algoritmo de recuperação ARIES

- Princípios básicos:
 - Escrita prévia no jornal: uma alteração, antes de ser realizada, é registada no jornal; o registo feito no jornal deve ser guardado em local seguro (disco) antes de a alteração à base de dados ser escrita em disco.
 - Repetir todas as ações durante o refazer: depois de uma falha do sistema, todas as ações são feitas de novo mesmo as ações daquelas transações que, não tendo sido cometidas no momento da falha, tenham de ser canceladas.
 - Registar no jornal as ações da fase ‘desfazer’: as alterações feitas à base de dados quando se desfazem as ações das transações que não foram cometidas, devem ser registadas no jornal para garantir a recuperabilidade no caso de haver nova falha de sistema. O seguro morreu de velho ☺

NoSQL

NoSQL – O que é?

- Classe de sistemas de bases de dados (vários)
- “Not only SQL”
 - Não usam SQL como linguagem para pesquisas
 - Arquitectura distribuída, tolerante a falhas
 - Não há esquema fixo
 - Não existem joins
 - Operações caras para combinar registos de duas ou mais tabelas num único set
 - Os joins requerem grande consistência e esquemas fixos
 - No entanto são sistemas mais flexíveis
- Não pretendem substituir RDBMS

Where NoSQL Is Used?

- Google (BigTable, LevelDB)
- LinkedIn (Voldemort)
- Facebook (Cassandra)
- Twitter (Hadoop/Hbase, FlockDB, Cassandra)
- Netflix (SimpleDB, Hadoop/HBase, Cassandra)
- CERN (CouchDB)



Cronologia do NoSQL

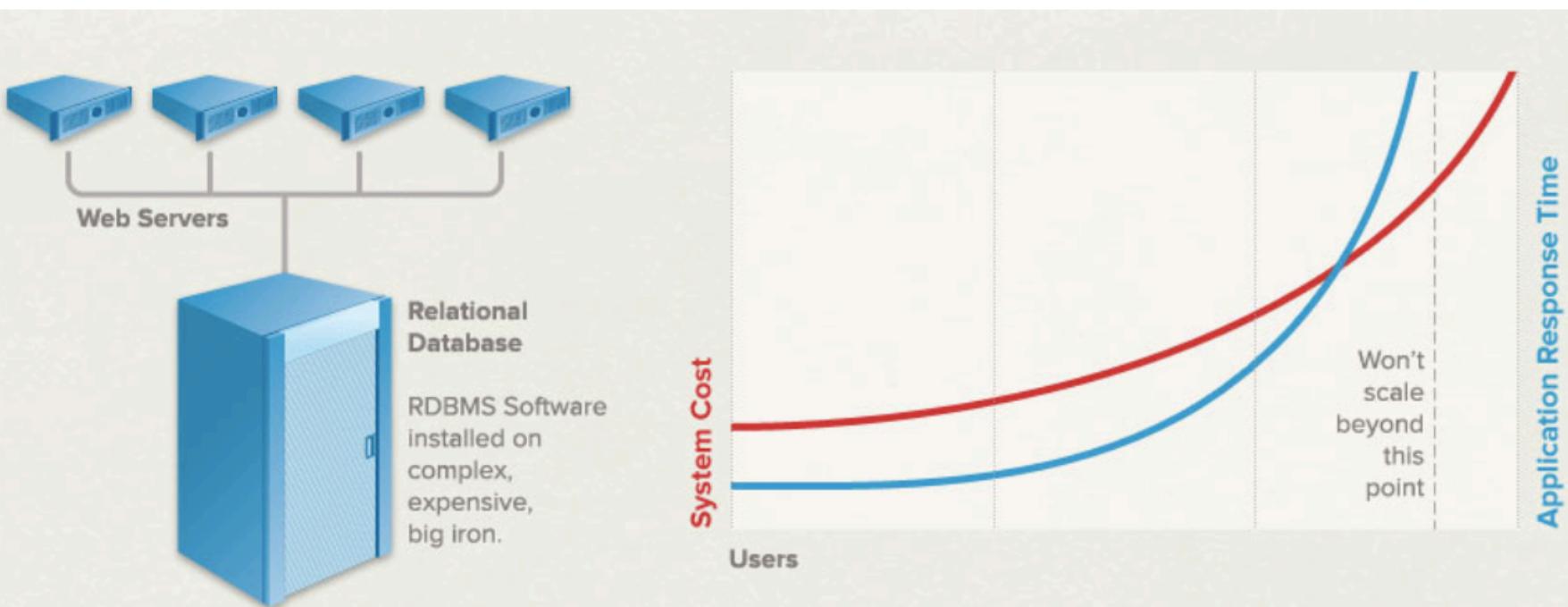
- 1998 – Carlo Strozzi menciona o termo mas com um objectivo bem diferente...
- 2003 – MemCached da LiveJournal
- 2004 – Paper do Google FileSystem
- 2006 – Paper do Google MapReduce
- 2006 – Paper do Google BigTable
- 2007 – Release do Hadoop no Apache
- 2009 – Paper do Amazon Dynamo
- 2010 – MongoDB (1a versão de produção)

Escalabilidade

- Capacidade de um sistema, rede ou processo para lidar com uma carga crescente de trabalho de uma forma eficaz e eficiente ou a sua elasticidade de forma a lidar com esse mesmo crescimento.
- Por exemplo, pode referir-se à capacidade de um sistema de aumentar o seu output face ao aumento de carga quando se adiciona recursos (geralmente hardware).

Escalabilidade BD Relacionais

- A certo ponto não escalam (elevados custos)



Web Application - RDBMS Scales Up. To support more users, you must get a bigger database server for your RDBMS. As a result, system cost grows exponentially with linear increases in users, and application response time degrades asymptotically.

Escalabilidade - opções

- Escalabilidade tradicional (vertical)
 - bases de dados na cloud escalam comprando servidores “maiores” (hardware) e mais caros
 - Nem sempre é possível
 - Em muitos casos não é fiável
- Paradigma de arquitectura
 - escalabilidade horizontal baseada na partição dos dados, i.e., dividir a base de dados em vários servidores “pequenos” e baratos.

Sharding

- Técnica
 - data sharding
 - partição horizontal de dados em diversas máquinas
- Consequências
 - Gerir o acesso paralelo da aplicação
 - Escalar tanto para as leituras como escritas
 - Não é transparente, aplicação tem que ser partition-aware

Escalabilidade NoSQL

- Escalabilidade horizontal é possível
- Lida bem com ‘picos’ na rede
- Possibilidade de scale up e scale down consoante a procura

Escalabilidade NoSQL

The diagram illustrates a system architecture for www.wellsfargo.com. It shows three user silhouettes at the top, connected to a Load Balancer. The Load Balancer connects to a pool of Web Servers, which in turn connect to a pool of NoSQL Database Servers. This architecture is designed to handle increasing numbers of users.

Application Scales Out
Just add more commodity web servers

System Cost

Users

Application Response Time

Database Scales Out
Just add more commodity data servers

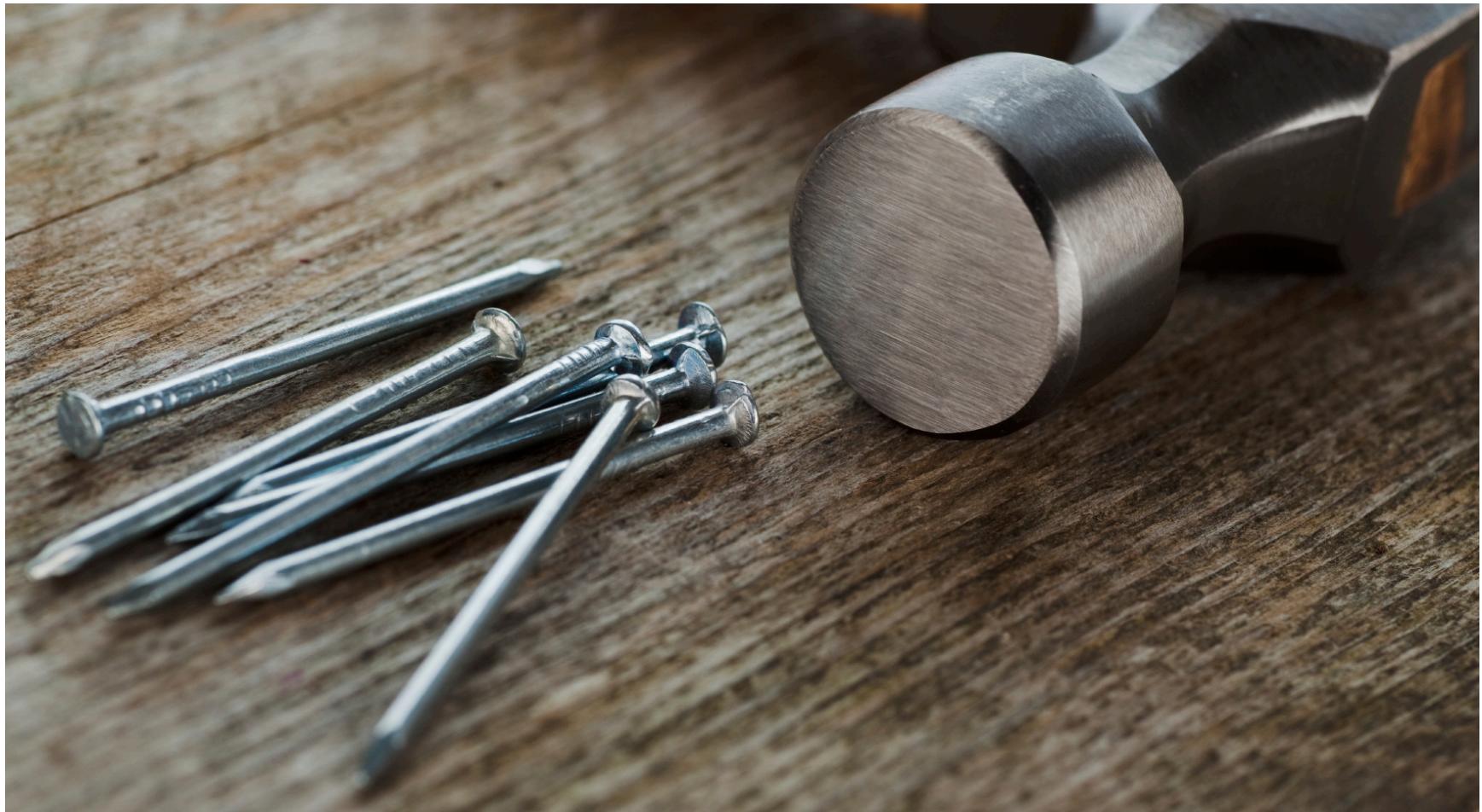
System Cost

Users

Application Response Time

In contrast to the non-linear increase in total system cost and asymptotic degradation of performance previously seen with RDBMS technology, NoSQL database technology flattens both curves.

Nem todos os problemas são pregos



ACID

- **Atomicidade**
 - ou todo o trabalho na transação é realizado ou nada
- **Consistência**
 - As transações alteram o estado da base de dados de um estado consistente para outro estado consistente (implica restrições)
- **Isolamento**
 - Os resultados de uma transação apenas são disponibilizados após commit.
- **Durabilidade**
 - Os resultados de uma transação após commit sobrevivem a falhas de sistema.

Bases de dados Relacionais

- Economia mundial é relacional
- SQL é uma query language rica, declarativa
- Bases de dados forçam integridade
- ACID
- Developers estão bem familiarizados
- Grande suporte de ferramentas (JDBC, ...)

Desafios das BD relacionais

- Desalinhamento
 - Dificuldade em mapear modelos de domínios complexos num esquema relacional
- Schema relacional é rígido
 - Dificuldade em lidar com dados semi-estruturados (alteração de atributos)
 - Alterações ao schema = downtime / prejuízo
- Extremamente difícil escalar escrita
 - Escalabilidade vertical é limitada / cara
 - Escalabilidade horizontal é limitada / cara

Teorema CAP

- Eric Brewer, 1998
- **Consistência** (*consistency*)
- **Disponibilidade** (*availability*)
- **Tolerância particional** (*partitioning tolerance*)

Consistência

- “Consistência Atómica”
- Diferente de consistência ACID
- Todos os nós do cluster vêm os mesmos dados a cada momento
- Cliente percebe que um grupo de operações ocorreu todo ao mesmo tempo

Disponibilidade

- Todas as operações resultam numa resposta de sucesso ou falha
- Falha(s) de um ou mais nós não impedem a operação dos restantes nós

Tolerância particional

- Operações completam-se mesmo quando componentes individuais não estão disponíveis
- Sistema continua funcional apesar de perda arbitrária de mensagens (erros de comunicação)

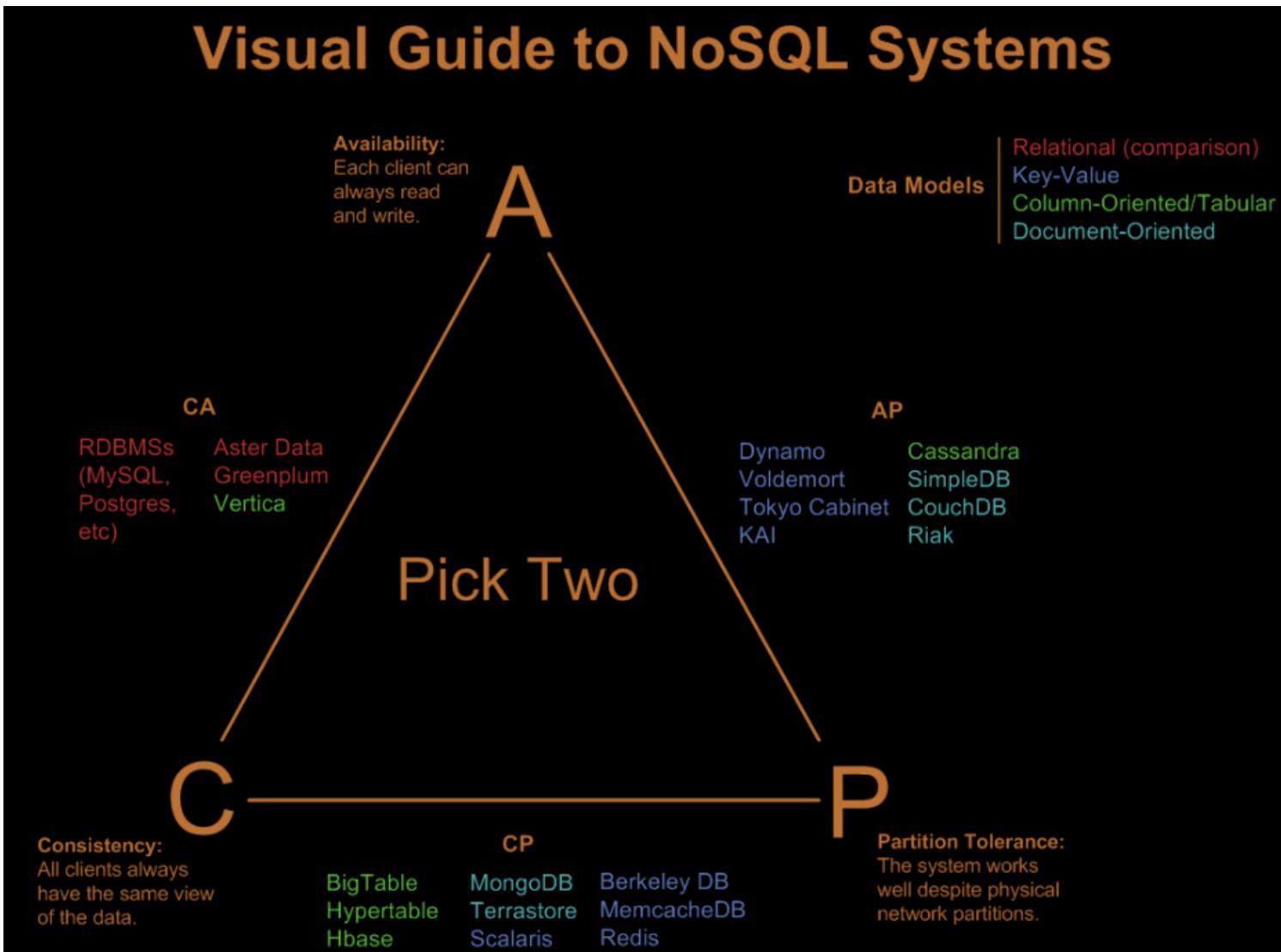
Teorema CAP

- Definição: qualquer sistema com dados distribuidos não é capaz de garantir as 3 propriedades em simultâneo

Teorema CAP e ACID

- Em NoSQL queremos garantir P
- Abandonar A or C do ACID
 - Relaxar C facilita a replicação e tolerância a falhas
 - Relaxar A reduz (ou elimina) a necessidade de controlo

CAP e sistemas NoSQL



Transações BASE

- **Basically Available**
 - não garante a disponibilidade dos dados
 - teorema de CAP
 - mas todos os pedidos são respondidos
 - mesmo que seja mensagem de erro
- **Soft State**
 - o estado do sistema pode mudar ao longo do tempo através de operações em background para eventual consistência
- **Eventual consistency**
 - o sistema não verifica a consistência depois de cada operação e continua a responder a pedidos sem verificar consistência. Eventualmente tratará de garantir consistência no futuro.

Transações BASE

- Características:
 - Fraca consistência
 - Prioridade à disponibilidade
 - Melhor esforço
 - Respostas aproximadas permitidas
 - Simples e rápido

NoSQL

- Definição (www.nosql-database.org):

Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontally scalable**. The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly.

Often more characteristics apply such as: **schema-free, easy replication support, simple API, eventually consistent** / BASE (not ACID), a huge amount of data and more. So the misleading term "nosql" (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

HOW TO WRITE A CV



Leverage the NoSQL boom

Bases de dados NoSQL

- Grandes volumes de dados ('Big Data')
- Replicação e distribuição escaláveis
 - Potencialmente milhares de máquinas
 - Potencialmente distribuídas no mundo inteiro
- As pesquisas exigem resposta rápida
- Sobretudo operações de pesquisa, poucos updates
- Inserts e updates assíncronos
- Não existe esquema (schema-less)
- BASE - Não são obrigatórias as propriedades ACID
- Desenvolvimento open source

Tipos de bases de dados NoSQL

- Wide column
- Key Value / Tuple stores
- Document
- Graph
- Multimodal
- Object
- Outros...

O ‘foco’ actualmente

- Key Value store
 - hash tables de keys
- Column oriented
 - cada bloco de storage contém apenas dados de uma coluna
- Document Store
 - armazena documentos constituídos por links para elementos
- Graph
 - focam-se em relações e não só em entidades

Key Value Store

- Tabela com 2 colunas contendo chave e valor associado a essa mesma chave
- A chave funciona como índice e o valor pode ser apontado como um look up.
- Exemplo
 - projecto Voldemort
 - Linkedin
 - Eventual consistência de chave valor, auto escalável

Column Oriented

- Armazena dados ordenados por coluna
- Permitem pares key value num sistema paralelo
 - Modelo de dados
 - define-se atributos num esquema e permite actualizações
 - Princípio de armazenamento
 - grandes hash tables distribuídas
 - Propriedades
 - particionamento vertical/horizontal, grande disponibilidade
 - Completamente transparente para a aplicação
 - Permite compressão à coluna

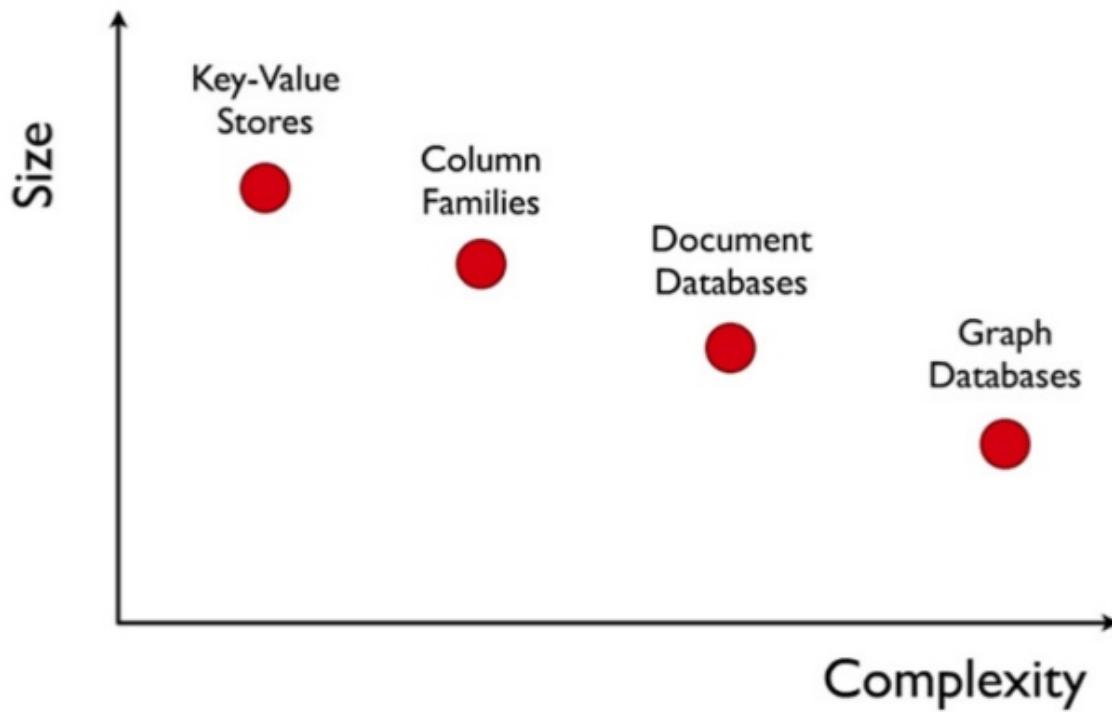
Document

- Coleção de documentos
- Não existe qualquer schema
- Baseadas em JSON
 - permite listas, datas, booleanos, etc...
- Indexa documentos semi estruturados

Graphs

- São criadas com nós, relações entre nós e propriedades dos nós
 - Nós representam entidades (CR7, Real Madrid)
 - Natureza similar aos objectos de programação OOP
 - Propriedades são informação sobre nós (idade)
 - Arestas representam relações (joga em)
- Escalabilidade em bases de dados de grafos é problemática (sharding...)

Complexidade



MongoDB

- Base de dados orientado ao documento
 - JSON (BSON)
 - Documentos organizados em collections (tabelas)
- Updates totais ou parciais de documentos
 - Update transacional em cada documento
 - Modificadores atómicos
- Linguagem de pesquisa rica e flexível
- Suporta índices
 - primários e compound
- GridFS para armazenar grande ficheiros
- MAP / REDUCE

MongoDB – casos de uso

- Real time analytics
- Sistemas de gestão de conteúdos
- Update parcial de um único documento
- Caching
- Grande volume de escritas

Quem usa?

- Foursquare
- Bit.ly
- NY Times
- SourceForge
- Shutterfly

BSON

- Binary JSON

```
{"hello": "world"}
```

→ \x16\x00\x00\x00
 \x02
 hello\x00
 \x06\x00\x00\x00world\x00
 \x00

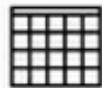
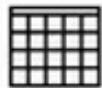
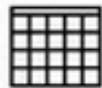
MongoDB vs RDBMS



Database



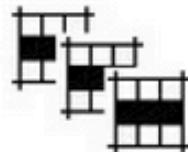
Database



Table

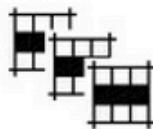


Collection



Document

MongoDB vs RDBMS



Rows

Id	Username
1	Ridho



Document

Id user	Street Name	House Number
1	Hirschbergerstrasse	62
1	Roemerstrasse	118

```
{  
  Username: "Ridho",  
  Adresses: [{  
    StreetName: "Hirschbergerstrasse",  
    HouseNumber: "62"  
  }, {  
    StreetName: "Roemerstrasse",  
    HouseNumber: "118"  
  }]  
}
```

MongoDB vs RDBMS

Books

Book_id	Title	ISBN	Year
1	Dragon Ball	120-99277	1989
2	Doraemon	220-99187	1995
3	Karate Kid	339-33111	1980

Authors

Author_id	Name
1	Akira Toriyama
2	Fujiko Fujio

Books Authors relation

id	Book	Author
1	1	1
2	2	2
3	3	1
4	3	2

MongoDB vs RDBMS

```
{  
    title:"Dragon Ball",  
    ISBN:"120-99277",  
    authors:[ "Akira Toriyama" ], year: 1989  
}  
  
{  
    title:"Doraemon",  
    ISBN:"220-99187",  
    authors:[ "Fujiko Fujio" ] , year: 1995  
}  
  
{  
    title:"Karate Kid",  
    ISBN:"339-33111",  
    authors:[ "Akira Toriyama", "Fujiko Fujio" ] , year: 1980  
}
```

ObjectID (`_id`)

- Id único para todos os documentos
- 12 bytes
- Consiste em
 - 0-3 timestamp
 - 4-6 machine
 - 7,8 PID
 - 9-11 increment

MongoDB vs SQL

```
CREATE TABLE users (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```

```
db.users.insert( {
    user_id: "abc123",
    age: 55,
    status: "A"
} )
```

MongoDB vs SQL

```
ALTER TABLE users  
ADD join_date DATETIME
```

```
db.users.update(  
    { },  
    { $set: { join_date: new Date() } },  
    { multi: true }  
)
```

MongoDB vs SQL

```
CREATE INDEX idx_user_id_asc  
ON users(user_id)
```

```
db.users.createIndex( { user_id: 1 } )
```

MongoDB vs SQL

```
INSERT INTO users(user_id,  
                  age,  
                  status)
```

```
VALUES ("bcd001",  
       45,  
       "A")
```

```
db.users.insert(  
    { user_id: "bcd001", age: 45, status: "A" }  
)
```

MongoDB vs SQL

```
SELECT *  
FROM users
```

```
db.users.find()
```

MongoDB vs SQL

```
SELECT *
```

```
FROM users
```

```
WHERE status = "A"
```

```
db.users.find(
```

```
  { status: "A" }
```

```
)
```

MongoDB vs SQL

```
SELECT *
FROM users
WHERE status != "A"

db.users.find(
    { status: { $ne: "A" } }
)
```

MongoDB vs SQL

```
SELECT *
FROM users
WHERE age > 25
```

```
db.users.find(
  { age: { $gt: 25 } }
)
```

MongoDB vs SQL

```
SELECT *
FROM users
WHERE status = "A"
ORDER BY user_id ASC
```

```
db.users.find( { status: "A" } ).sort( { user_id: 1 } )
```

MongoDB vs SQL

```
SELECT COUNT(*)  
FROM users  
WHERE age > 30
```

```
db.users.find( { age: { $gt: 30 } } ).count()
```

MongoDB vs SQL

```
UPDATE users  
SET status = "C"  
WHERE age > 25
```

```
db.users.update(  
    { age: { $gt: 25 } },  
    { $set: { status: "C" } },  
    { multi: true }  
)
```

Desvantagens do NoSQL

- Nova tecnologia logo buggy
- Dados são geralmente duplicados, potencial inconsistência
- Não existe schema standard
- Não existe standard de formato de queries
- Não existe linguagem standard
- Depende da camada de aplicação para garantir integridade dos dados
- Demasiadas opções e soluções

