Aplicações - SQL

Banco de Dados: Teoria e Prática

André Santanchè e Luiz Celso Gomes Jr
Instituto de Computação – UNICAMP
Agosto de 2013

Picture by Steve Kelley 2008

SQL

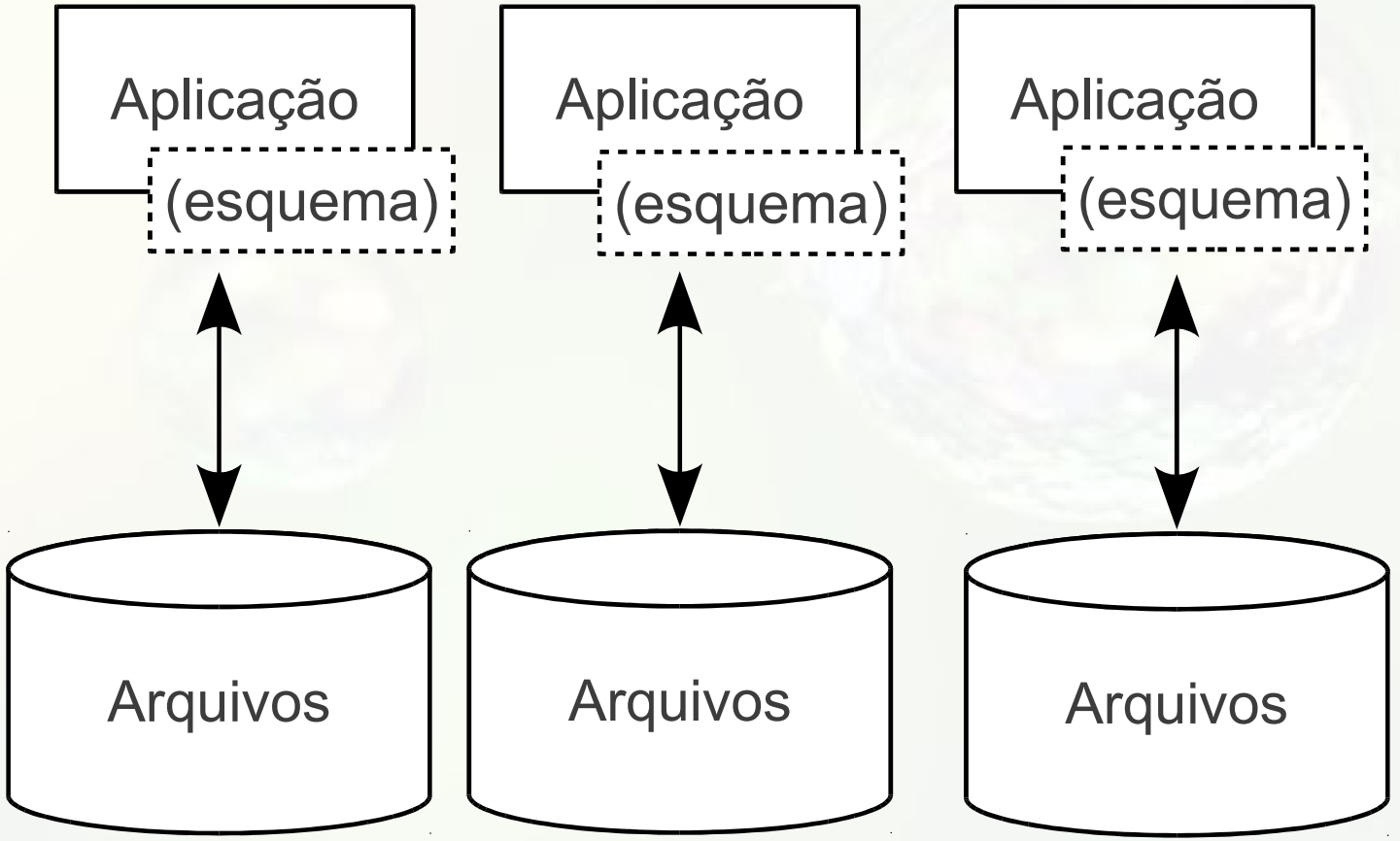
- SQL – Structured Query Language
- Originalmente: SEQUEL – Structured English QUEry Language
- Criada pela IBM Research
  - Interface BD Relacional → SYSTEM R

SQL
Padronização

- ANSI + ISO
- SQL-86 ou SQL1
- SQL-92 ou SQL2
- SQL:1999 ou SQL3
- SQL:2003
- SQL:2006

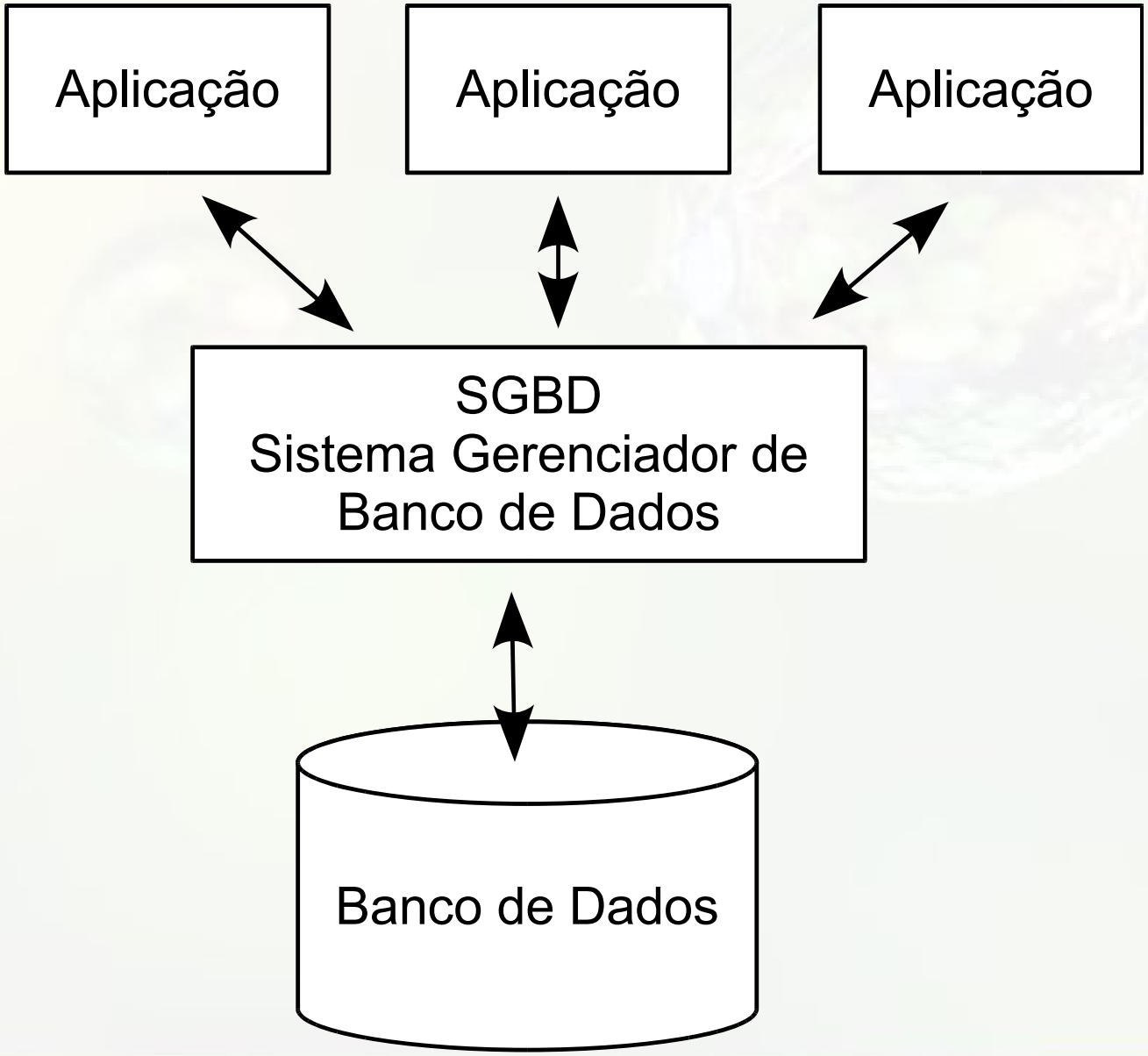
Aplicações e Armazenamento

Arquivos

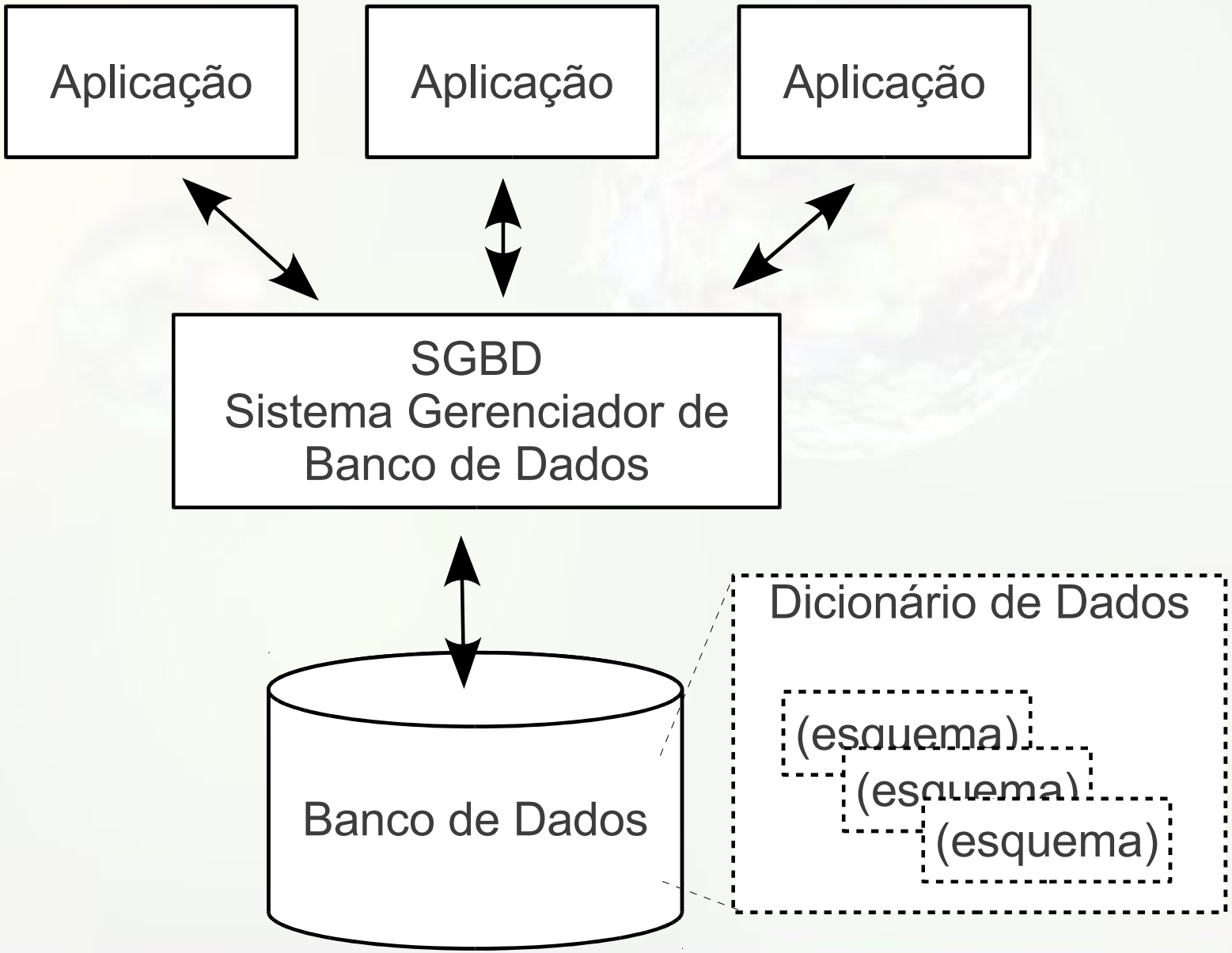


Aplicações e Armazenamento

SGBD

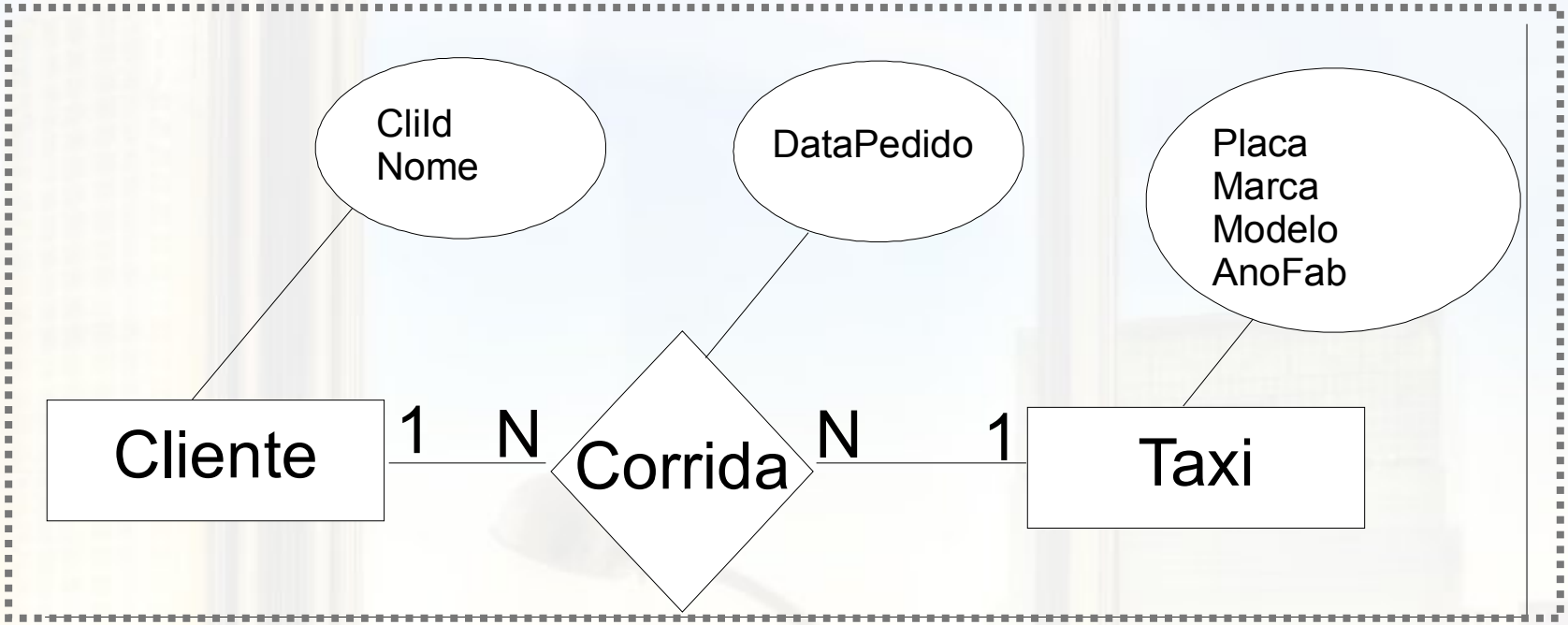


Dicionário de Dados



Caso Prático - Taxis

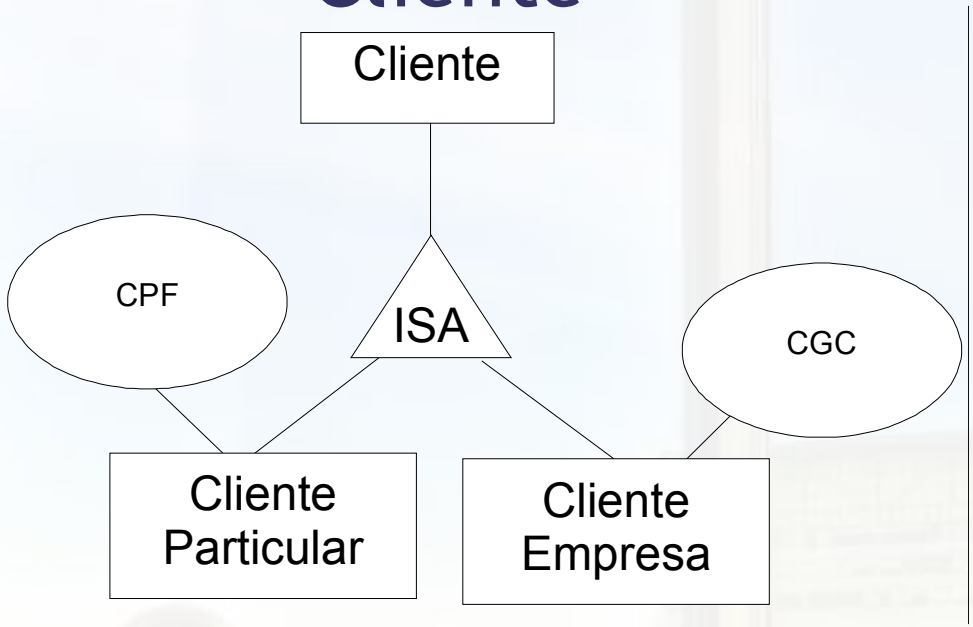
Esquema Conceitual – Exemplo
Táxis



Este é um subconjunto do Estudo de Caso proposto “Despacho e controle de Táxis via terminais móveis ligados on-line com um sistema multi-usuário” por prof. Geovane Cayres Magalhães

Esquema Conceitual – Exemplo

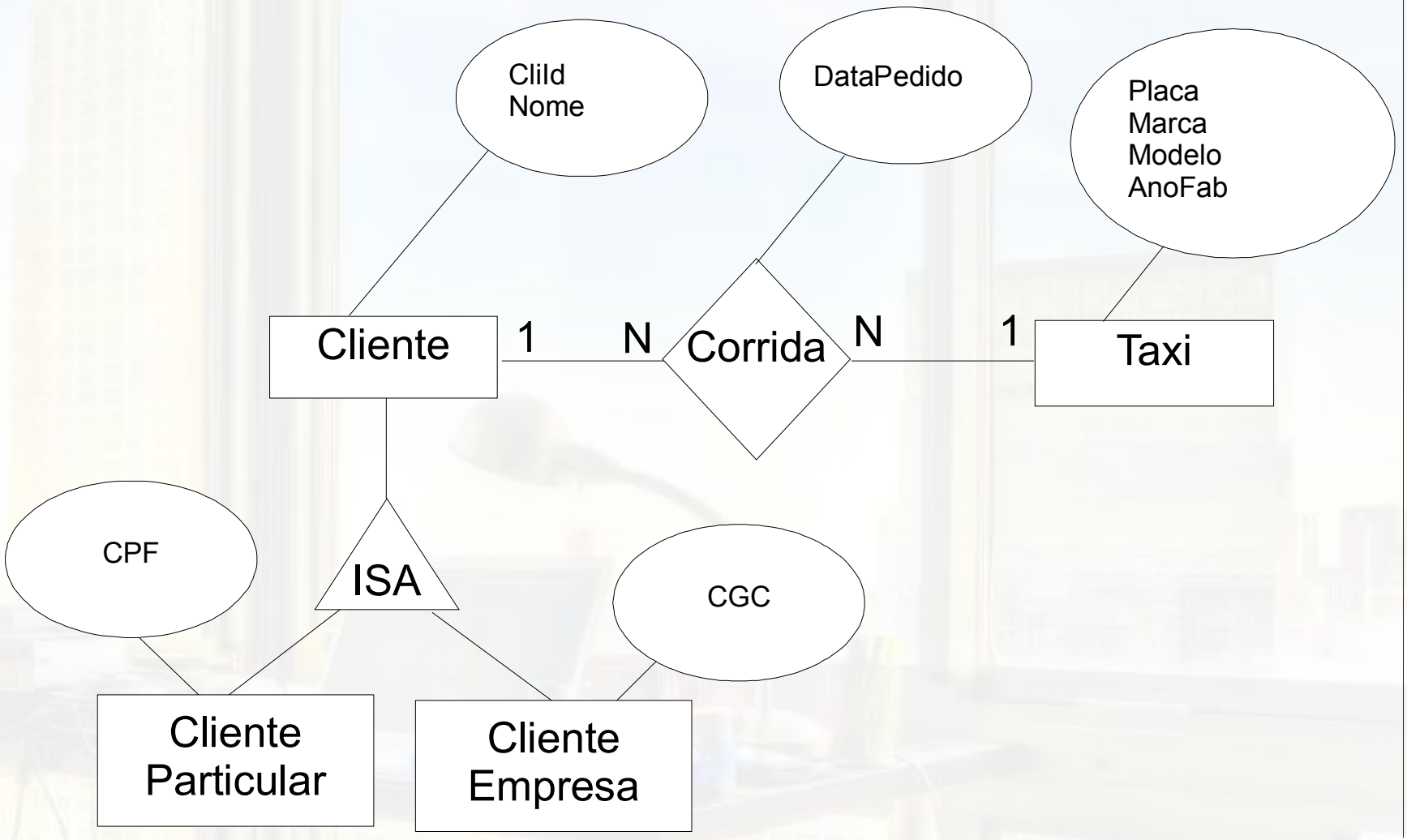
Cliente



Para ilustrar o tema apresentado, foram acrescentadas duas entidades que são especialização de Cliente. A primeira representa um indivíduo que irá pagar a conta, a segunda representa um funcionário de uma empresa conveniada, para a qual a conta será enviada. Um cliente pode pertencer a ambas especializações.

Esquema Conceitual completo

Táxis



Tabelas para exemplo - Táxis

Cliente Particular (CP)

CliId	Nome	CPF
1532	Asdrúbal	448.754.253-65
1755	Doriana	567.387.387-44
1780	Quincas	546.373.762-02

Cliente Empresa (CE)

CliId	Nome	CGC
1532	Asdrúbal	754.856.965/0001-54
1644	Jepeto	478.652.635/0001-75
1780	Quincas	554.663.996/0001-87
1982	Zandor	736.952.369/0001-23

Tabelas para exemplo - Táxis

Táxi (TX)

Placa	Marca	Modelo	AnoFab
DAE6534	Ford	Fiesta	1999
DKL4598	Wolksvagen	Gol	2001
DKL7878	Ford	Fiesta	2001
JDM8776	Wolksvagen	Santana	2002
JJM3692	Chevrolet	Corsa	1999

Corrida (R1)

ClId	Placa	DataPedido
1755	DAE6534	15/02/2003
1982	JDM8776	18/02/2003

CREATE SCHEMA

- CREATE SCHEMA <esquema>
  AUTHORIZATION <id_autorizado>

- Java: executeUpdate(...)

CREATE TABLE

- CREATE TABLE <tabela>
  (<campo$_1$> <tipo> [NULL|NOT NULL] [restrição],
  [...,
  <campo$_n$> <tipo> [NULL|NOT NULL] [restrição],
  PRIMARY KEY <chave_primaria>])

- Java: executeUpdate(...)

CREATE TABLE

```
CREATE TABLE Taxi (
   Placa VARCHAR(7) NOT NULL,
   Marca VARCHAR(30) NOT NULL,
   Modelo VARCHAR(30) NOT NULL,
   AnoFab INTEGER,
   Licenca VARCHAR(9),
   PRIMARY KEY(Placa)
);

CREATE TABLE Cliente (
   CliId VARCHAR(4) NOT NULL,
   Nome VARCHAR(80) NOT NULL,
   CPF VARCHAR(14) NOT NULL,
   PRIMARY KEY(CliId)
);
```

CREATE TABLE
FOREIGN KEY

- CREATE TABLE <tabela>

  ...
  FOREIGN KEY (<coluna_estr>$_1$[,...,<coluna_estr>$_n$])
  REFERENCES <tabela_ref>([<coluna_ref>[,...,<coluna_ref]])
  [ON DELETE <ação_ref>]
  [ON UPDATE <ação_ref>]

- <ação_ref>
  - NO ACTION → impede a ação na tabela mestre <tabela_ref>
  - CASCADE → propaga a ação da tabela mestre
  - SET NULL → valores de referências alterados para nulo
  - SET DEFAULT → valores de referências alterados para default

CREATE TABLE
FOREIGN KEY

```
CREATE TABLE Corrida (
   CliId VARCHAR(4) NOT NULL,
   Placa VARCHAR(7) NOT NULL,
   DataPedido DATE NOT NULL,
   PRIMARY KEY(CliId, Placa, DataPedido),
   FOREIGN KEY(CliId)
      REFERENCES Cliente(CliId)
         ON DELETE NO ACTION
         ON UPDATE NO ACTION,
   FOREIGN KEY(Placa)
      REFERENCES Taxi(Placa)
         ON DELETE NO ACTION
         ON UPDATE NO ACTION
);
```

Exercício 1

- Escreva uma comando SQL para criar os esquemas:
  - Pessoa(<u>nome</u>, nome_da_mãe, ano_nascimento, nome_cidade_natal)
    - nome_cidade_natal → CHE Cidade
  - Cidade(<u>nome_cidade</u>, sigla_estado)

INSERT

- INSERT INTO <tabela>
  [(<campo$_1$>[,..., <campo$_n$>])]
  VALUES (<valor$_1$>[,..., <valor$_n$>])

- executeUpdate(...)

Exercício 2

- Escreva um comando SQL para inserir uma tupla na tabela Pessoa com os seus dados e dados de familiares próximos (cerca de 2 linhas). Preencha a tabela Cidade com as cidades listadas na tabela Pessoa e suas respectivas siglas de estado. Use dados fictícios se preciso.

SELECT

- SELECT * | <campo$_1$>[,..., <campo$_n$>]
  FROM <tabela$_1$>[,..., <tabela$_n$>]
  WHERE <condição/junção>

- executeQuery(...)

Exercício 3

- Para a tabelas que você montou no exercício 1, escreva um comando SQL que retorne:

  a) nomes de todas as mães

  b) nomes de todas as mães com filhos maiores de 12 anos

SELECT
LIKE

- SELECT ...
  FROM <tabela$_1$>[,..., <tabela$_n$>]
  WHERE <condição/junção>

- % → qualquer cadeia com 0 a n caracteres
- _ → exatamente um caractere (qualquer)
- = → caractere de escape
  - e.g., serve para encontrar um caractere _

AS (alias)

- SELECT <campo$_1$> [AS] <alias$_1$>
  [,..., <campo$_n$> [AS] <alias$_n$>]
  ...

- SELECT ...
  FROM <tabela$_1$> [AS] <alias$_1$>
  [,..., <tabela$_n$> [AS] <alias$_n$>]
  ...

SELECT
DISTINCT e ALL

- SELECT DISTINCT ...
- SELECT ALL ...
- A cláusula ALL é implícita se não especificada

SELECT
ORDER BY

- SELECT ...
  ORDER BY <campo$_1$>[,..., <campo$_n$>]

Exercício 4

- Para a tabelas que você montou no exercício 1, escreva um comando SQL que retorne:
  - nomes de parentes que nasceram no mesmo estado que você
  - retorne todos os primos por parte de mãe, que você for capaz de inferir a partir da tabela

DELETE

- DELETE FROM <tabela$_1$>
  WHERE <condição>

- executeUpdate(...))

UPDATE

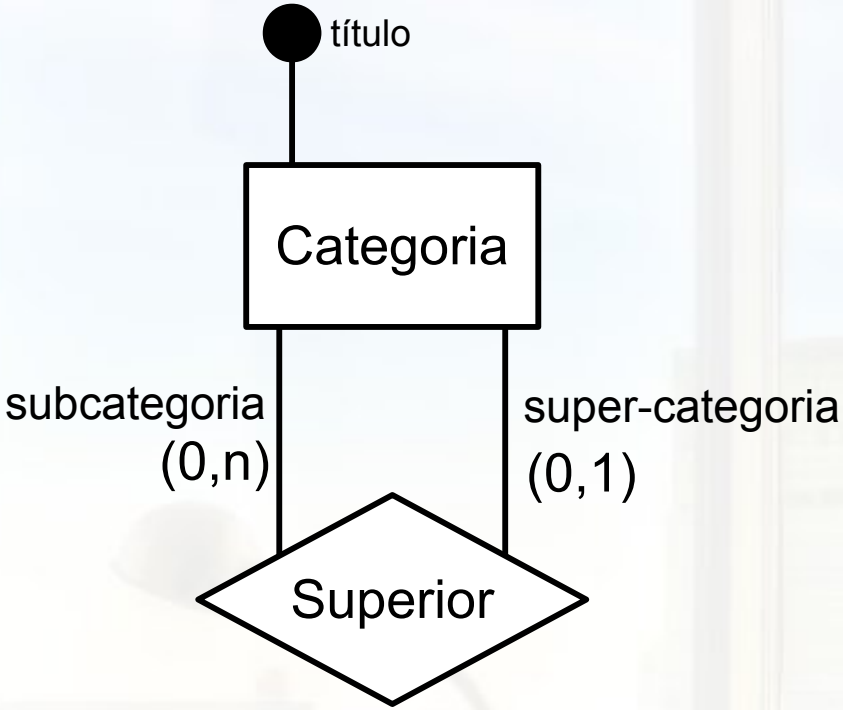
- UPDATE <tabela>
  SET <campo$_1$>=<valor$_1$>
  [,..., <campo$_n$>=<valor$_n$>]
  WHERE <condição>

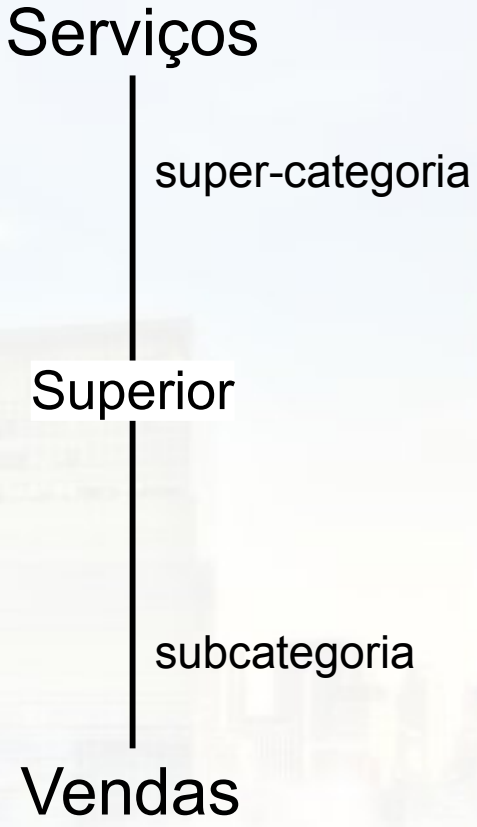
- executeUpdate(...)

Aplicando o UPDATE

Categorias de Marcadores

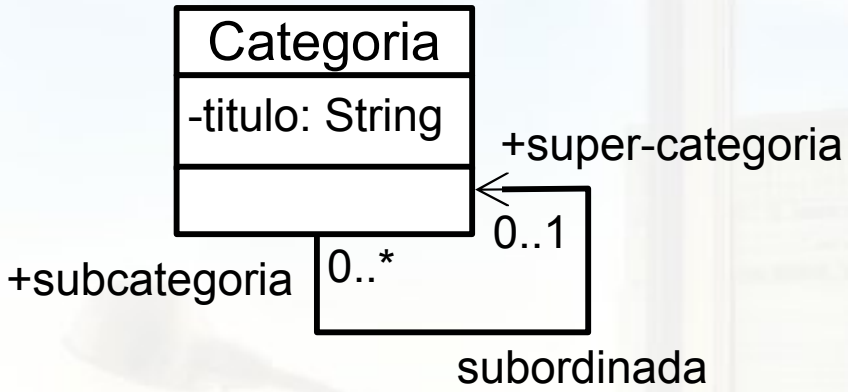
Modelo ER

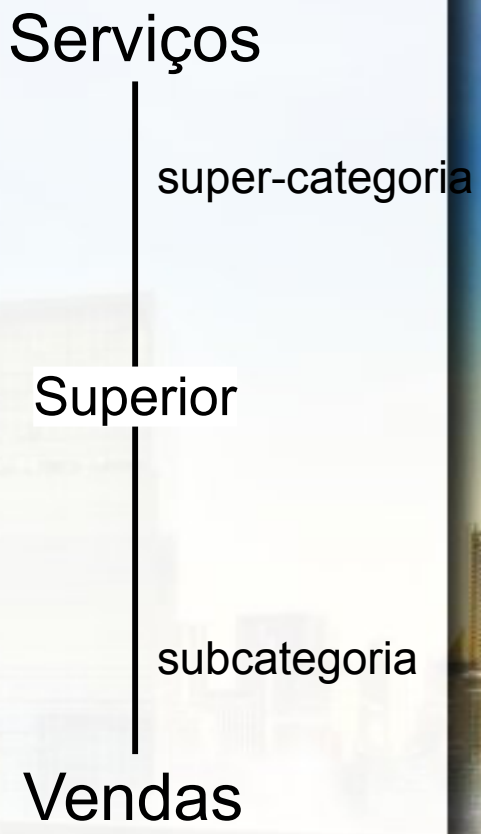




Categorias de Marcadores

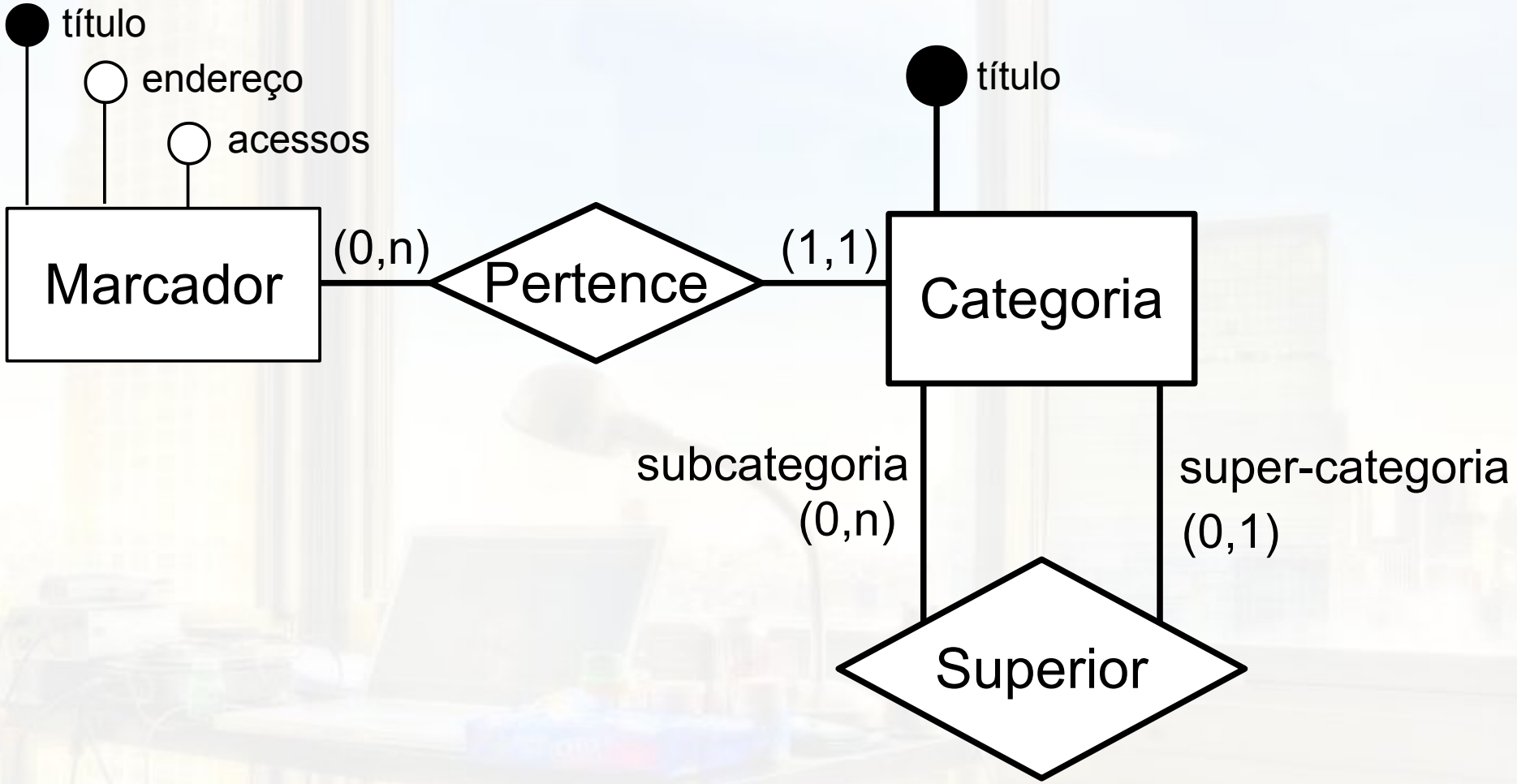
Modelo UML





Marcadores e Categorias

Modelo ER



Marcadores e Categorias

Modelo ER

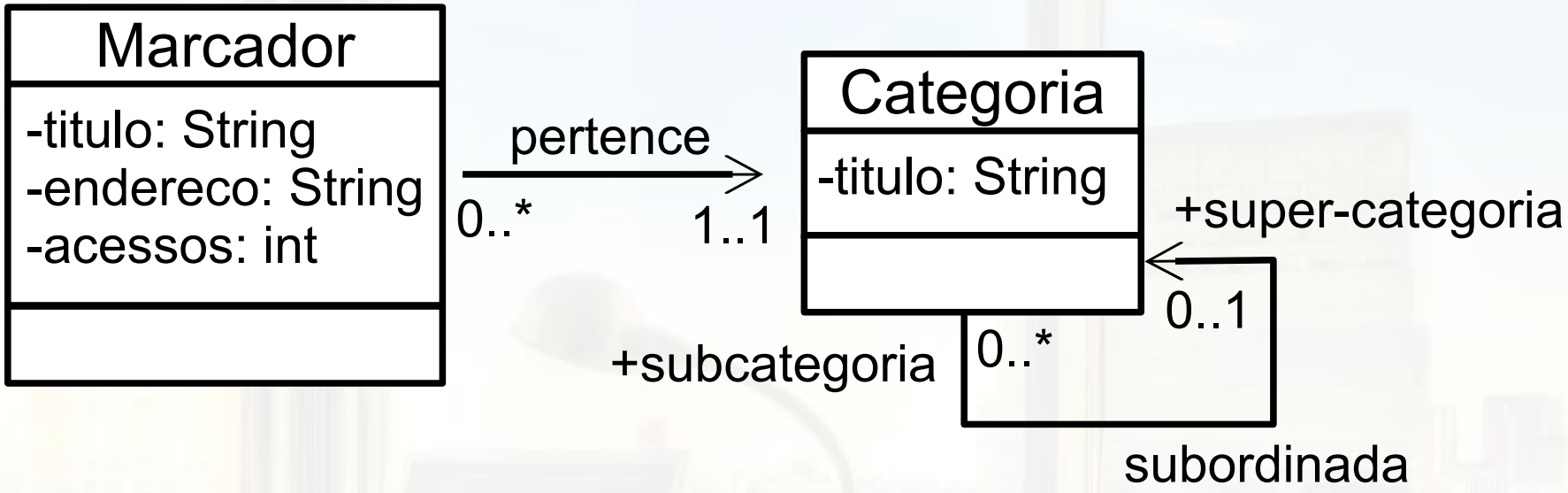
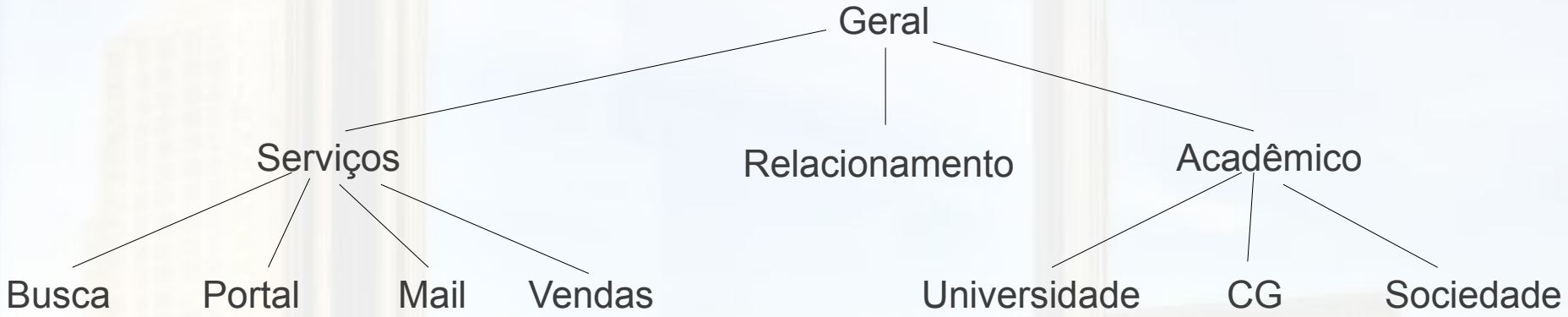


Tabela Taxonomia

Modelo Relacional



Categoria	Superior
Geral	
Serviços	Geral
Acadêmico	Geral
Relacionamento	Geral
Busca	Serviços
Portal	Serviços
Mail	Serviços
Vendas	Serviços
Universidade	Acadêmico
CG	Acadêmico
Sociedade	Acadêmico

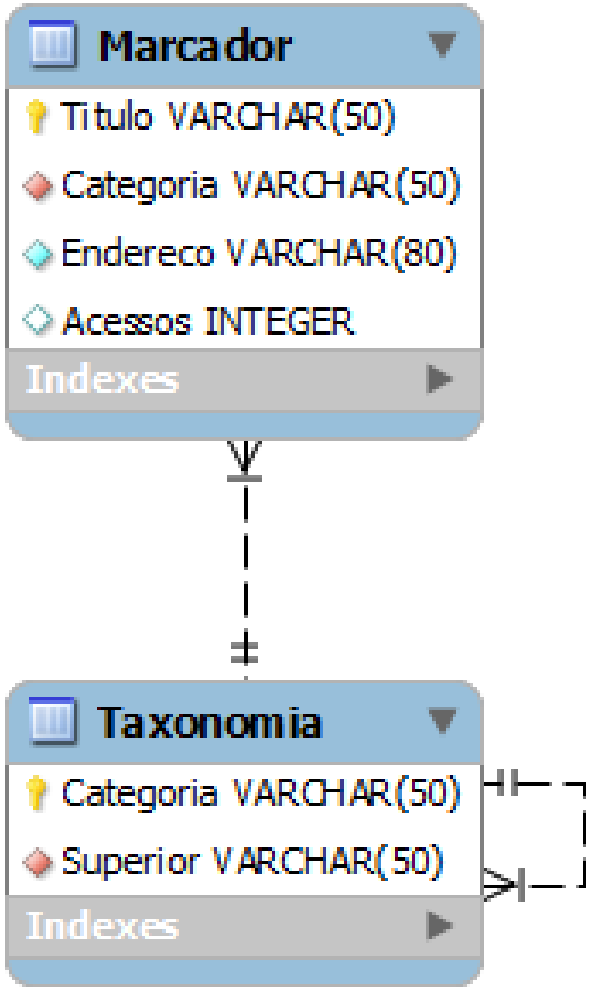
Marcadores e Categorias

Modelo Relacional

Marcador(Titulo, Categoria, Endereco, Acesso)

- Categoria: chave estrangeira para Taxonomia

Taxonomia(Categoria, Superior)



Estudo de Caso
SQL

- UPDATE Marcadores
  SET Categoria = <nova>
  WHERE Categoria = <antiga>
- UPDATE Taxonomia
  SET Categoria = <nova>
  WHERE Categoria = <antiga>
- UPDATE Taxonomia
  SET Superior = <nova>
  WHERE Superior = <antiga>

Exercício 5

- Retomando os seguintes esquemas:
  - Pessoa(<u>nome</u>, nome_da_mãe, ano_nascimento, nome_cidade_natal)
    - nome_cidade_natal → CHE Cidade
  - Cidade(<u>nome_cidade</u>, sigla_estado)
- É possível especificar um comando SQL de criação da tabela Pessoa que permita mudar o nome de uma cidade nas tabelas Pessoa e Cidade com um único comando SQL?
- Se sim, escreva o(s) comando(s) CREATE necessários para isso e a sentença SQL de mudança do nome da cidade.

Prepared Statement

Utilizando o PreparedStatement

- SELECT FROM Marcadores
  WHERE Titulo = ?

- <comando>.setString(<numero>, <valor>)

Utilizando o PreparedStatement

- INSERT INTO Marcadores
  VALUES (? , ? , ? , ?)

- <comando>.setString(<numero>, <valor>)
- <comando>.setInt(<numero>, <valor>)

Utilizando o PreparedStatement

- UPDATE Marcadores
  SET Categoria = ?
  WHERE Categoria = ?

- <comando>.setString(<numero>, <valor>)
- <comando>.setInt(<numero>, <valor>)

Agrupamento

GROUP BY

- SELECT * | <campo$_1$>[,..., <campo$_n$>]
  FROM <tabela$_1$>[,..., <tabela$_n$>]
  WHERE <condição/junção>
  GROUP BY <coluna_agrupar>
  HAVING <condição_grupo>

Exercício 6

- Escreva uma sentença SQL, baseada no esquema abaixo, que retorne o número de pessoas da família em cada estado:
  - Pessoa(nome, nome_da_mãe, ano_nascimento, nome_cidade_natal)
    - nome_cidade_natal → CHE Cidade
  - Cidade(nome_cidade, sigla_estado)

Funções de Agregação

- COUNT(*) ⇨ contagem
- SUM(<coluna>) ⇨ soma
- AVG(<coluna>) ⇨ média
- MAX(<coluna>) ⇨ maior valor
- MIN(<coluna>) ⇨ menor valor

Visões

VIEW

- CREATE VIEW <nome> AS SELECT ...

Consultas Aninhadas

SELECT
IN e NOT IN

- SELECT ...
  WHERE <campo> IN
  (SELECT <campo> ...)

- SELECT ...
  WHERE <campo> NOT IN
  (SELECT <campo> ...)

SELECT
EXISTS e NOT EXISTS

- SELECT ...
  WHERE EXISTS
  (SELECT <campo> …)

- SELECT ...
  WHERE NOT EXISTS
  (SELECT <campo> …)

SELECT
Comparação

- SELECT ...
  WHERE <campo> <comparação>
  (SELECT <campo> …)

Exercício 7

- Para a tabelas que você montou no exercício 1, escreva um comando SQL que retorne retorne todos os primos por parte de mãe, que você for capaz de inferir a partir da tabela. Considere que você tem como ponto de partida o nome da sua avó.
- Utilize duas estratégias:
  - VIEW
  - SELECT aninhado

SELECT aninhado também pode ser usado em operações de UPDATE e DELETE

Join

Join

- SELECT …
  FROM <tabela> JOIN <tabela>
  ON <condição> …

- Tipo clássico de join explicitado
- Também conhecido como INNER JOIN

Natural Join

- SELECT …
  FOM <tabela> NATURAL JOIN <tabela>

- Condição não especificada
- EQUIJOIN: Verifica igualdade de cada par de atributos com o mesmo nome

Outer Join

- SELECT …
  FROM <tabela> <join> <tabela>
  ON <condição> …
- <join>
  - LEFT JOIN – toda tupla à esquerda aparece
  - RIGT JOIN – toda tupla à direita aparece
  - FULL JOIN – toda tupla aparece

União, Interseção e Diferença

- SELECT …
  <operador>
  SELECT …
- <operador>
  - UNION
  - INTERSECT
  - EXCEPT

André Santanchè
http://www.ic.unicamp.br/~santanche

Licença

- Estes slides são concedidos sob uma Licença Creative Commons. Sob as seguintes condições: Atribuição, Uso Não-Comercial e Compartilhamento pela mesma Licença, com restrições adicionais:
  - Se você é estudante, você não está autorizado a utilizar estes slides (total ou parcialmente) em uma apresentação na qual você esteja sendo avaliado, a não ser que o professor que está lhe avaliando:
    - lhe peça explicitamente para utilizar estes slides;
    - ou seja informado explicitamente da origem destes slides e concorde com o seu uso.
- Mais detalhes sobre a referida licença Creative Commons veja no link: http://creativecommons.org/licenses/by-nc-sa/2.5/br/