Base de Dados

08 - SQL

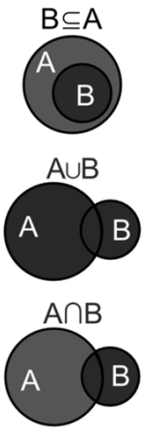
Vitor Vaz da Silva

Base de Dados - Índice

Base de Dados

Álgebra Relacional

- Subconjunto
- União
- Intersecção
- Cardinalidade
  - Número de elementos
- Produto Cartesiano
  - AxBxCxD {(a_1,b_1,c_1,d_1), (a_2,b_2,c_2,d_2), …}
- Grau
  - Dimensão do par ordenado (a,b,c) é 3



SQL

SQL (Structured Query Language)

É uma linguagem normalizada, utilizada por um largo conjunto de SGBD relacionais para:

- Definição de dados
- Manipulação de dados
- Interrogações
- Controlo transaccional
- Gestão de privilégios

É um norma da ISO e da ANSI

Existem duas formas da linguagem ser usada

- Interactivamente
- Embutida noutras linguagens de programação (C, C++, JAVA, etc)

SQL

SQL – História

1970: Codd define o Modelo Relacional

1974: A IBM desenvolve o SYSTEM/R com a linguagem SEQUEL (Structured English Query Language)

Lançamento de SGBDs comerciais

- 1979: Primeiro SGDB comercial (Relational software Inc. hoje ORACLE Corp.)
- 1981: SGBD INGRES
- 1983: IBM anuncia o DB2

Normas SQL

- 1986, 1987: Norma SQL-86 (ANSI X3.135-1986 e ISO 9075:1987)
- 1989: Norma SQL-89 (ANSI X3.135.1-1989)
- 1992: Norma SQL2 (ISO/IEC 9075:1992)
- 1996: Proposta Norma SQL3
- 2003: Integração de XML

DEZ – 2016 (ISO/IEC 9075:2016)

SQL

SQL (Structured Query Language) - continuação

É uma Linguagem não-Procedimental

- Especifica-se O QUÊ e não COMO

Existe uma clara abstracção perante a estrutura física dos dados

- não é necessário especificar caminhos de acesso, nem elaborar algoritmos de pesquisa física

As operações efectuam-se sobre:

- conjuntos de Dados (Tabelas)
- não é necessário (nem possível) manipular os Dados Linha-a-Linha.

Não é *CASE-SENSITIVE*

Podem existir SGBDs que não implementem todas as características da linguagem

SQL

SQL – As duas Componentes da linguagem

LDD (Linguagem de Definição de Dados) DDL
- Permite definir os Esquemas de Relação
- Permite definir os atributos dos Esquemas de Relação
- Permite definir restrições (chaves primárias, estrangeiras, etc.)

LMD (Linguagem de Manipulação de dados) DML
- Permite aceder à informação armazenada na base de dados
- Permite inserir, eliminar e alterar a informação presenta na base de dados

SQL

Comandos principais

Definição de Dados
- CREATE – Criar estruturas de dados (tabelas, vistas, índices)
- ALTER – Alterar estruturas de dados
- DROP – Remover estruturas de dados

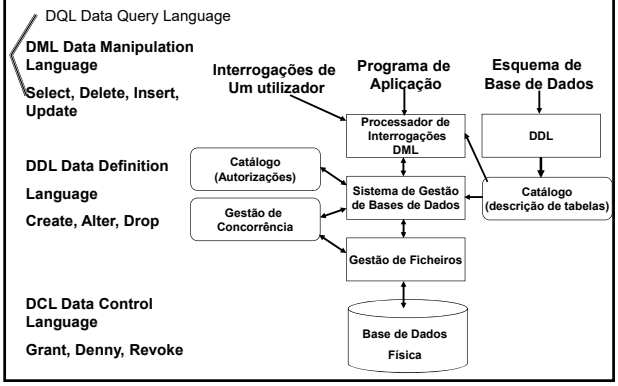
} **LDD**

Interrogação
- SELECT

Manipulação de Dados
- INSERT – Inserir novos registos
- UPDATE – Alterar registos existentes
- DELETE – Apagar registos

} **LMD**

O Ambiente - SQL



SQL

Comandos principais

Controlo de Transações
- COMMIT
- SAVEPOINT
- ROLLBACK

Segurança
- GRANT – Usado para atribuir direitos de acesso
- REVOKE – Usado para retirar direitos de acesso

SQL

LDD – Linguagem de definição de dados

- Existem três comandos pertencentes á LDD:
  - CREATE (criar estruturas de dados)
  - ALTER (alterar estruturas de dados)
  - DROP (remover estruturas de dados)
- A criação de tabelas pode ser efectuada em qualquer momento de uma sessão SQL
- A estrutura de uma tabela pode ser alterada em qualquer momento de uma sessão SQL, podendo ser perdida a informação anteriormente armazenada nessa tabela
- Uma tabela não tem qualquer dimensão pré determinada

SQL

CREATE – para criar uma tabela

- A sintaxe do geral do CREATE, para este caso:

  – CREATE TABLE <nome tabela>
  ({<nome coluna> <tipo>
  [DEFAULT <valor | função | NULL>]
  [<restrição de coluna>]
  }+
  [<restrição de tabela>]
)
- Onde
  – <restrição de coluna> indica uma restrição a aplicar a uma coluna
  – <restrição de tabela> aplica-se a mais de uma coluna

SQL

Tipos

CHAR[ACTER] (n)	Cadeia de caracteres com, exactamente, ***n*** caracteres
CHAR[ACTER]	Abreviatura de CHARACTER (1)
CHAR[ACTER] VARYING (n) VARCHAR (n)	Cadeia de caracteres com comprimento variável até ***n***
BIT (n)	Cadeia de bits com, exactamente, ***n*** bits
BIT VARYING (n)	Cadeia de bits com comprimento variável

SQL

Tipos

NUMERIC (p, q)	Número decimal com, exactamente, ***p*** dígitos e sinal e ponto decimal assumido na posição ***q***, contada da direita para a esquerda
DECIMAL (p, q)	Número decimal (BCD) com, pelo menos, ***p*** dígitos e sinal e ponto decimal assumido na posição ***q***, contada da direita para a esquerda
INTEGER	Número inteiro com sinal (decimal ou binário)
SMALINT	Número inteiro com sinal (decimal ou binário) com precisão não superior a INTEGER

SQL

Tipos

FLOAT (p) FLOAT REAL DOUBLE PRECISION	Número de vírgula flutuante $N = f \times 10^{exp}$

SQL

Tipos de dados do standard SQL2

- CHARACTER(n) – cadeia de caracteres de dimensão n, fixa, n>0
  - Utiliza-se CHAR como abreviação
- CHARACTER VARYING(n) – cadeia de caracteres de dimensão variável, com um máximo de n caracteres, n>0
  - Utiliza-se VARCHAR como abreviação
- BIT(n) – cadeia de bits com dimensão fixa de n bits, n>0
- BIT VARYING(n) – cadeia de bits com dimensão variável com um máximo de n bits, n>0
- NUMERIC(p,q) – número decimal com (no máximo) p dígitos e sinal, com q casas decimais, a contar da direita, $0 \leq q \leq p$, p >0
  - NUMERIC(p) é uma abreviação de NUMERIC(p,0)
  - a precisão do número é exactamente de p dígitos

SQL

Tipos de dados do standard SQL2 - continuação

- DECIMAL(p, q) – número decimal com p dígitos e sinal, com q casas decimais, a contar da direita, $0 \leq q \leq p \leq m$, p >0
  - DEC é uma abreviação de DECIMAL
  - DECIMAL(p) é uma abreviação de DECIMAL(p,0)
  - A precisão do número pode não ser p, podendo ter uma precisão m maior
- INTEGER – número inteiro, com sinal, decimal ou binário
  - INT é uma abreviação de INTEGER
- SMALLINT – número inteiro, com sinal, decimal ou binário
  - SMALLINT terá sempre uma precisão nunca superior a INT

SQL

Tipos de dados do standard SQL2 - continuação

- FLOAT(p) – número de virgula flutuante
  - FLOAT é uma abreviação de FLOAT(p), onde p depende da implementação
- REAL é uma alternativa a FLOAT(s), onde s depende da implementação
- DOUBLE PRECISION é uma alternativa a FLOAT(d), onde d depende da implementação

SQL

Caso prático - Tipos de dados no SQL Server

- Binários: BINARY[(n)], VARBINARY[(n)]
- Carácter: CHAR[(n)], VARCHAR[(n)], TEXT
- Caracteres Unicode: NCHAR[(n)], NVARCHAR[(n)],NTEXT
- Data e Hora: DATETIME, SMALLDATETIME, TIMESTAMP
- Numérico exacto: DECIMAL[(p[,s])], NUMERIC[(p[,s])]
- Numérico aproximado: FLOAT[(n)], REAL
- Inteiro: BIGINT, INT, SMALLINT, TINYINT
- Monetário: MONEY, SMALLMONEY
- Outros: BIT, IMAGE

- Algumas considerações:
  - Quando n não é especificado, o valor assumido é 1; ex CHAR ≡ CHAR(1)
  - O n máximo para as cadeias de caracteres é de 8000 *(4000 para unicode)*

SQL

Caso prático - Tipos de dados no SQL Server - continuação

- Mais considerações:
  - O tipo BINARY consiste em dados hexadecimais; ex: O número decimal 245 é guardado como F5
  - É preferível usar o tipo VARBINARY sempre que a dimensão seja inferior a 8kb. Quando esse valor for excedido é preferível usar o tipo IMAGE
  - O tipo IMAGE é utilizado preferencialmente para guardar documentos; ex: imagens, documentos postscript, etc.
  - Para os tipos de dados VAR... apenas é alocado o espaço necessário para guardar os dados, até ao máximo especificado (n). Para o correspondente tipo sem o prefixo VAR, é alocado todo o espaço definido (n), independentemente da dimensão efectiva dos dados. É de utilizar o tipo VAR... quando se prevê, nessa coluna, existência de valores nulos ou grandes variações na dimensão dos dados

SQL

Caso prático - Tipos de dados no SQL Server - continuação

A dimensão suportada pelos tipos IMAGE e TEXT de $2^{31} – 1$ bytes/caracteres e $2^{30} – 1$ para NTEXT

O tipo de dados DATETIME permite armazenar datas desde 1 de Janeiro de 1753 até 31 Dezembro 9999. Tem precisão de 3.33 millisegundos

O tipo de dados SMALLDATETIME permite armazenar datas desde 1 de Janeiro de 1900 até 6 de Junho de 2079, com precisão ao minuto

Os tipos DECIMAL e NUMERIC são equivalentes, sendo suportados os dois por questões de compatibilidade

A precisão p máxima é 38

Quando é utilizada a precisão máxima, podem representar-se números de $-10^{38} +1$ até $10^{38} +1$

Para $1 \leq p \leq 9$ são necessários 5 bytes para armazenar os dados

Para $29 \leq p \leq 38$ são necessários 17 bytes para armazenar os dados

SQL

Caso prático - Tipos de dados no SQL Server - continuação

- O tipo FLOAT pode armazenar valores compreendidos entre - 1.79E + 308 e 1.79E + 308, quando é especificado um n de 53 (máximo)
  - Quando $1 \leq n \leq 24$, tem-se uma precisão de 7 dígitos e ocupa 4 bytes
  - Quando $25 \leq n \leq 53$, tem-se uma precisão de 15 dígitos e ocupa 8 bytes
- FLOAT(24) é sinónimo de REAL
- FLOAT(53) é sinónimo de DOUBLE PRECISION
- <u>É de evitar a referência a colunas "floating-point" em cláusulas WHERE</u>
- O tipo BIGINT utiliza 8 bytes para armazenar inteiros compreendidos entre -2^63 e -2^63 -1
- O tipo INT utiliza 4 bytes (números entre -2^31 e -2^31 -1)
- O tipo SMALLINT utiliza 2 bytes (números entre -2^15 e -2^15 -1)
- O tipo TINY utiliza 1 bytes (números 0 e 255)
- O tipo BIT armazena 0,1 ou NULL

SQL

CREATE – para criar uma tabela – continuação

- As restrições de tabela podem ser
  - **[CONSTRAINT nome_restrição]**
    [{PRIMARY KEY | UNIQUE]} (coluna, …)] |
    [FOREIGN KEY (coluna, …) {REFERENCES tabela [(coluna, …)]
    [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}] | [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}}] |
    [CHECK (condição)]
- As restrições definidas na criação das tabelas são asseguradas pelo SGDB durante a manipulação dos dados (INSERT, UPDATE, DELETE)
- Quando alguma restrição for violada, o comando em execução é abortado

SQL

CREATE – para criar uma tabela – continuação

- Definição de uma tabela com uma chave primária, uma chave candidata (alternativa) e restrição de valor NULL:
  - CREATE TABLE ALUNO
    (numAluno int CONSTRAINT pk_ALUNO PRIMARY KEY,
    numBI char(8) NOT NULL CONSTRAINT ak1_ALUNO UNIQUE,
    nome varchar(100)
    CONSTRAINT ck1_ALUNO CHECK (nome IS NOT NULL))
- Definição de uma tabela com chave primária composta e restrição de valor NULL:
  - CREATE TABLE LINHAFACTURA
    (numFactura int,
    numLinha int,
    quantidade int NOT NULL,
    CONSTRAINT pk_LINHAFACTURA PRIMARY KEY (numFactura, numLinha))

SQL

CREATE – para criar uma tabela – continuação

- Definição de uma tabela com chave estrangeira composta:
  - CREATE TABLE ENTREGA
    (numEntrega int
      dataEntrega datetime NOT NULL,
      numEnc int NOT NULL,
      numProduto int NOT NULL,
      CONSTRAINT pk_ENTREGA PRIMARY KEY (numEntrega),
      CONSTRAINT fk1_ENTREGA FOREIGN KEY (numEnc,numProduto)
            REFERENCES FACTURA (numFactura, numLinha)
            ON DELETE CASCADE
)

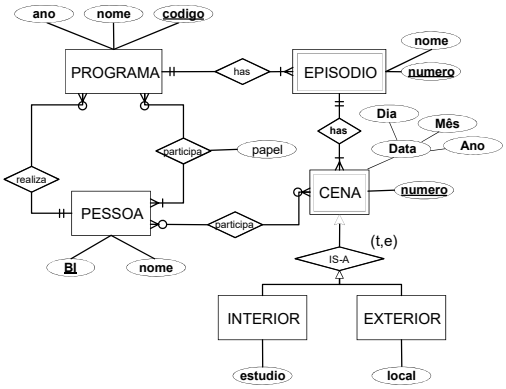
Cascade: Quando remover um tuplo factura são removidos todos os tuplos de entrega a ele ligados.

SQL

CREATE – para criar uma tabela – continuação

- Definição de uma tabela com uma regra de verificação:
  - CREATE TABLE ENCOMENDA
    (numEnc int CONSTRAINT pk_ENCOMENDA PRIMARY KEY,
      dataEnc datetime NOT NULL,
      codCliente int NOT NULL CONSTRAINT fk1_ENCOMENDA
      FOREIGN KEY REFERENCES CLIENTE (codCliente),
      dataEntrega datetime,
      CONSTRAINT ck1_ENCOMENDA CHECK (dataEntrega > dataEnc)
)

EXEMPLO



EXEMPLO

PESSOA(BI, Nome)

```
CREATE Table PESSOA (
     BI VARCHAR(8) CONSTRAINT pk_PESSOA Primary Key,
      Nome VARCHAR(50)
)
```

PROGRAMA(Código, ano, nome, realizador)

```
CREATE TABLE PROGRAMA (
     Código CHAR(8) CONSTRAINT pk_PROGRAMA Primary Key,
     ano INTEGER,
     nome VARCHAR(50),
     realizador VARCHAR(8) NOT NULL
               CONSTRAINT fk_PROGRAMA Foreign Key
               REFERENCES PESSOA(BI)
)
```

EXEMPLO

EPISÓDIO(Codigo, Num_Ep, nome)

```
CREATE TABLE EPISODIO (
      código CHAR(8)
      CONSTRAINT fk_EPISÓDIO Foreign Key
            REFERENCES PROGRAMA(Código),
      num_ep INTEGER,
      nome VARCHAR(80) NOT NULL,
       CONSTRAINT pk_EPISODIO Primary Key (código, num_ep)
)
```

EXEMPLO

CENA(Codigo,Num_Ep,Num_Cen, Dia,Mês,Ano)

```
CREATE TABLE CENA (
      código CHAR(8) CONSTRAINT fk1_CENA Foreign Key
            REFERENCES PROGRAMA(Código),
      num_ep INTEGER,
       num_cena INTEGER,
       dia INTEGER NOT NULL CONSTRAINT ck1_CENA
                  CHECK(dia BETWEEN 1 and 31),
       mes INTEGER NOT NULL CONSTRAINT ck2_CENA
                  CHECK(mes BETWEEN 1 and 12),
      ano INTEGER NOT NULL CONSTRAINT ck3_CENA
                  CHECK(ano > 1990),
      CONSTRAINT pk_CENA Primary Key (código, num_ep,Num_cena),
      CONSTRAINT fk2_CENA Foreign Key (código,num_ep)
      REFERENCES EPISODIO(Código,num_ep)
 )
```

EXEMPLO

INTERIOR(Codigo, Num_Ep, Num_Cen, estudio)

```
CREATE TABLE INTERIOR (
     código CHAR(8) CONSTRAINT fk1_INTERIOR Foreign Key
              REFERENCES PROGRAMA(Código),
      num_ep INTEGER,
      num_cena INTEGER,
      Estudio Varchar(10) CONSTRAINT ck_INTERIOR
                  check(Estudio in ('K01', 'TOBIS 27', 'NBP32')),
      CONSTRAINT pk_INTERIOR Primary Key (código,Num_ep,Num_cena),
      CONSTRAINT fk2_INTERIOR Foreign Key (código,num_ep)
          REFERENCES EPISODIO(Código,num_ep)
)
```

EXEMPLO

EXTERIOR(Codigo, Num_Ep, Num_Cen, local)

```
CREATE TABLE EXTERIOR(
     código CHAR(8) CONSTRAINT fk1_Exterior Foreign Key
          REFERENCES PROGRAMA(Código),
     num_ep INTEGER,
     num_cena INTEGER,
     Local Varchar(50),
     CONSTRAINT pk_Exterior Primary Key (código, num_ep, Num_cena),
     CONSTRAINT fk2_Exterior Foreign Key (código, num_ep)
          REFERENCES EPISODIO(Código, num_ep)
)
```

EXEMPLO

PARTICIPA_PROG(BI, Codigo, papel)

```
CREATE TABLE PARTICIPA_PROG (
     BI VARCHAR(8) CONSTRAINT fk1_PART_PROG Foreign key
                REFERENCES PESSOA(BI),
     CÓDIGO CHAR(8) CONSTRAINT fk2_PART_PROG Foreign key
                REFERENCES PROGRAMA(Código),
      PAPEL VARCHAR(50),
      CONSTRAINT pk_PART_PROG Primary Key (BI, código)
)
```

SQL

PARTICIPA_CEN(BI, Codigo, Num_EP, Num_Cen)

```
CREATE TABLE PARTICIPA_CENA(
     BI VARCHAR(8) CONSTRAINT fk1_PART_CENA Foreign key
                REFERENCES PESSOA(BI),
     CÓDIGO CHAR(8) CONSTRAINT fk2_PART_CENA Foreign key
                REFERENCES PROGRAMA(Código),
     NUM_EP INTEGER,
     NUM_CENA INTEGER,
     CONSTRAINT pk_PART_CENA Primary Key
                                (BI,código,NUM_EP,NUM_CENA),
     CONSTRAINT fk3_PART_CENA Foreign Key (código,num_ep)
                                REFERENCES EPISODIO(Código,num_ep),
     CONSTRAINT fk4_PART_CENA Foreign Key
                                (código,num_ep,num_cena)
          REFERENCES CENA(Código,num_ep,num_cena)
)
```

SQL

ALTER – para alterar uma tabela

- **Sintaxe do comando ALTER, para alteração de tabelas:**
  - **ALTER TABLE nome_tabela**
    [ADD {[COLUMN] novas colunas | CONSTRAINT <novas restrições_coluna>}]
    [ALTER [COLUMN] coluna]
    [DROP {[COLUMN] coluna} | {CONSTRAINT restrição_coluna} [RESTRICT |CASCADE]]
- **As alterações possíveis a uma tabela são:**
  - **acrescentar colunas, eventualmente acompanhadas de restrições**
  - **alterar a definição de colunas existentes**
  - **acrescentar restrições de integridade à tabela**
  - **remover uma restrição da tabela**
- **Não é possível, no entanto:**
  - **modificar uma coluna com valores NULL para NOT NULL.**
  - **Só se pode adicionar uma coluna NOT NULL a uma tabela que não contenha nenhuma linha.**
  - **Remover uma coluna se essa for a única existente na tabela**

SQL

ALTER – para alterar uma tabela – continuação

Alguns exemplos

- Adição de uma coluna:
  - ALTER TABLE EMPREGADO ADD comissao int NOT NULL
- Modificação da definição de uma coluna:
  - ALTER TABLE EMPREGADO ALTER column comissao smallint NOT NULL
- Remoção de uma coluna:
  - ALTER TABLE EMPREGADO DROP comissao
- Eliminar uma restrição de integridade:
  - ALTER TABLE ENCOMENDA DROP CONSTRAINT ck1_ENCOMENDA
- Acrescentar uma restrição de integridade (chave estrangeira):
  - ALTER TABLE ENCOMENDA ADD CONSTRAINT fk1_ENCOMENDA FOREIGN KEY (codCliente) REFERENCES CLIENTE (codCliente)

SQL

DROP – para remover uma tabela

- A sintaxe do DROP, para remover tabelas
  - DROP TABLE <nome-tabela> [RESTRICT|CASCADE]
- Exemplo de remoção da tabela EMPREGADO
  - DROP TABLE EMPREGADO
  - DROP TABLE EMPREGADO RESTRICT
- Algumas características da acção de remoção de uma tabela:
  - a remoção de uma tabela causa a perda de todos os dados nela existentes, assim como de todos os índices associados
  - uma tabela só pode ser removida por quem a criou ou pelo administrador da Base de Dados
  - se a tabela estiver a ser referenciada em VIEWS (abordadas mais adiante) ou em restrições de integridade, o comando DROP falha
  - no entanto se for especificada a palavra CASCADE, tanto a tabela como quem a referencia (VIEWS e restrições de integridade) são removidas

SQL

Sintaxe do comando SELECT

- A sintaxe geral de uma interrogação SQL é a seguinte:
  - SELECT [DISTINCT] <colunas> | *
    FROM <tabelas>
    [WHERE <condição>]
- <colunas> especifica a lista de atributos cujos valores interessa conhecer
- <tabelas> especifica quais as tabelas envolvidas no processamento da interrogação
- <condição> é uma expressão lógica que define a condição a verificar
- DISTINCT indica que se quer remover os duplicados no resultado final
- O símbolo * é utilizado quando se pretendem seleccionar todos os atributos das tabelas especificadas na cláusula FROM

SQL

Operações Algébricas

- Será com o comando SELECT que as operações algébricas serão implementadas
- Recordando quais são essas operações:
- Operadores Unários
  - Selecção
  - Projecção
- Operadores Binários
  - União
  - Intersecção
  - Diferença
  - Produto Cartesiano
  - Junção
  - Divisão

SQL

Selecção

Operação de selecção escolhe um conjunto de tuplos de um relação que verificam uma determinada condição

Sendo r uma relação, esta operação representa-se da seguinte forma:

$\sigma_{\text{<condição boolena>}}(r)$

Quais as pessoas com idade superior a 25 anos?

$\sigma_{\text{idade> 25}}(\text{PESSOA})$

PESSOA

BI	Nome	Idade
123	José	25
456	José	70
789	Joana	15
909	Pedro	30

$\sigma_{\text{idade> 25}}$ **(PESSOA)**

BI	Nome	Idade
456	José	70
909	Pedro	30

SQL

Selecção

Cada condição booleana consiste numa sequências de cláusulas da forma:

Atributo op valor, em que valor pertence ao domínio do atributo

Atributo op Atributo

As operações op podem ser:

<, >, =, ≥, ≤, <>

As cláusulas estão ligadas entre si por operadores lógicos:

∧ (e), ∨ (ou), ¬ (negação)

BI	Nome	Idade
456	José	70

Quais as pessoas com idade superior a 25 anos e com nome José? $\sigma_{\text{idade > 25} \wedge \text{nome = José}}(\text{PESSOA})$

SQL

Selecção– características

A operação de selecção é uma operação unária, que tem como operando uma única relação

O grau da relação resultante da aplicação da operação é igual ao grau da relação operando

SQL

Selecção– características

Uma sequência de operações σ pode ser escrita numa só que tem como condição a conjunção das condições das várias operações σ

σcond1 (σ cond2 (...σ condN(r)…)) =
σσ (cond1 ∧ cond2 ∧ … ∧ condn) (r)

A composição de operações σ é comutativa, i.e.:
σ cond1 (σ cond2 (r))= σ cond2 (σ cond1 (r))

SQL

Selecção– características

Quais as pessoas com idade superior a 25 anos e com nome José?

$\sigma_{idade > 25}(\sigma_{nome = José}(PESSOA)) =$
$\sigma_{idade > 25 \wedge nome = José}(PESSOA)$

$\sigma_{idade > 25}(\sigma_{nome = José}(PESSOA)) =$
$\sigma_{nome = José}(\sigma_{idade > 25}(PESSOA))$

SQL

Exemplo

- Considere-se os seguintes Esquemas de Relação e respectivas Relações

DEPARTAMENTO(codDept, nomeDept, localizacao)

CATEGORIA(codCat, designacao, salarioBase)

EMPREGADO(codEmp, nomeEmp, dataAdmissao, codCat, codDept, codEmpChefe)

SQL

Exemplo

Departamento

codDepart	nomeDepart	localizacao
1	Contabilidade	Lisboa
2	Vendas	Porto
3	Investigação	Coimbra

Categoria

codCat	designacao	salarioBase
1	CategoriaA	300
2	CategoriaB	250
3	CategoriaC	160

Empregado

codEmp	nomeEmp	dataAdmissão	codCat	codDept	codEmpChefe
1	António	20-Mar-01	1	1	1
2	João	20-Mar-01	1	2	1
3	Nuno	20-Mar-01	3	3	1
4	Carlos	6-Abr-98	3	2	2

SQL

Operações Algébricas - Selecção ou Restrição

- *Questão:* Quais as Categorias onde o salário base é inferior a 1200€?

codCat	designacao	salarioBase
2	CategoriaB	1.100,00 €
3	CategoriaC	750,00 €

Em álgebra relacional:

$\sigma_{salarioBase<1200}$ (CATEGORIA)

Em SQL:

SELECT * FROM CATEGORIA WHERE salarioBase<1200

SQL

Operações Algébricas - Selecção ou Restrição

Questão: Quais as Categorias onde o salário base é inferior a 1000€ ou que têm a descrição *'Categoria A'* ?

Em álgebra relacional:

$\sigma_{salarioBase<1000 \text{ v } designacao='categoriaA'}$ (CATEGORIA)

Em SQL:

- SELECT * FROM CATEGORIA WHERE salarioBase<1000 OR designacao='CategoriaA'

SQL

Projecção

Operação de projecção escolhe um determinado subconjunto de atributos da relação operando

Sendo r uma relação, esta operação representa-se da seguinte forma:

$\pi_{<\text{lista de atributos}>}(r)$

Quais os nomes e idades das pessoas?

$\pi_{\text{nome, idade}}(\text{PESSOA})$

SQL

Projecção

π<lista de atributos>(r)

Quais os nomes e idades das pessoas?

$\pi_{\text{nome, idade}}(\text{PESSOA})$

PESSOA

BI	Nome	Idade
123	José	25
456	José	70
789	Joana	15
909	Pedro	30

$\pi_{\text{nome,idade}}(\text{PESSOA})$

Nome	Idade
José	25
José	70
Joana	15
Pedro	30

SQL

Projecção – características

Quando no resultado de uma operação de projecção existirem vários tuplos iguais, apenas um é considerado:

Este procedimento é conhecido como *"Eliminação de duplicados"*

Com este procedimento é garantido que o resultado é uma relação, ou seja, um conjunto

SQL

Projecção – características

Se na lista de atributos da projecção estiver contida a chave da relação, a cardinalidade do resultado é a igual à cardinalidade da relação operando

Na composição de várias projecções

- Se lista1 está contida em lista2
  - $\pi_{\text{lista1}}(\pi_{\text{lista2}}(\text{PESSOA})) = \pi_{\text{lista1}}(\text{PESSOA})$
- Se lista1 não está contida em lista2
  - $\pi_{\text{lista2}}(\pi_{\text{lista1}}(\text{PESSOA}))$ = operação inválida

SQL

Operações Algébricas - Projecção

- *Questão:* Qual o nome e data de admissão de cada empregado?
- Em álgebra relacional:
  - $\pi_{\text{nomeEmp, dataAdmissão}}(\text{EMPREGADO})$
- Em SQL:
  - SELECT nomeEmp, dataAdmissão FROM EMPREGADO

SQL

Operações Algébricas - Projecção

- *Questão:* Quais os códigos das categorias de cada um dos empregados?

nomeEmp	dataAdmissão
António	20-Mar-01
João	20-Mar-01
Nuno	20-Mar-01
Carlos	6-Abr-98

- Em álgebra relacional:
  - $\pi_{\text{codCat}}(\text{EMPREGADO})$

codCat
1
3

codCat
1
1
3
3

- Em SQL:
  - SELECT codCat FROM EMPREGADO

SQL

Operações Algébricas – Projecção (continuação)

- Para não existirem duplicados é necessário utilizar a palavra reservada DISTINCT

SQL

Operações Algébricas – Projecção (continuação)

Questão: Qual a data de admissão e código da categoria de cada empregado?

- *Em álgebra relacional:*
  - $\pi_{\text{dataAdmissão,codCat}}$ (EMPREGADO)

dataAdmissão	codCat
20-Mar-01	1
20-Mar-01	3
6-Abr-98	3

- *Em SQL:*

```
SELECT DISTINCT dataAdmissão,codCat
FROM EMPREGADO
```

```
SELECT dataAdmissão,codCat
FROM EMPREGADO
```

dataAdmissão	codCat
20-Mar-01	1
20-Mar-01	1
20-Mar-01	3
6-Abr-98	3

SQL

Encadeamento das operações Projecção e Selecção

Escrever as operações numa única expressão

As operações são aplicadas uma de cada vez a resultados intermédios

Para o exemplo anterior

Nome	Idade
José	70
Pedro	30

Quais os nomes e idades das pessoas cuja idade é superior a 25 anos?

- $\pi_{\text{nome,idade}}$ ($\sigma_{\text{idade>25}}$ (PESSOA))
- Raux = $\sigma_{\text{idade> 25}}$ (PESSOA) ;
- $\pi_{\text{nome,idade}}$ (Raux)

SQL

Operações Algébricas – Composição de Projecção e Selecções

Questão: Qual o nome e a data de admissão dos empregados com categoria igual a 3 e admitidos antes de '20-02-2000'?

Em álgebra relacional:

$\pi_{\text{nomeEmp,dataAdmissão}}$ ($\sigma_{\text{dataAdmissão < '20-02-2000'} \wedge \text{codCat=3}}$ (EMPREGADO))

nomeEmp	dataAdmissão
Carlos	6-Abr-98

SQL

Operações Algébricas – Composição de Projecção e Selecções

$\pi_{\text{nomeEmp,dataAdmissão}}$ ($\sigma_{\text{dataAdmissão < '20-02-2000'} \wedge \text{codCat=3}}$ (EMPREGADO))

- *Em SQL:*

nomeEmp	dataAdmissão
Carlos	6-Abr-98

```
SELECT DISTINCT emp.nomeEmp,
                emp.dataAdmissão
    FROM (SELECT * FROM EMPREGADO
          WHERE dataAdmissão<'20-02-2000'
    AND codCat=3) as emp
```

SQL

Operações Algébricas – Composição de Projecção e Selecções

```
SELECT DISTINCT emp.nomeEmp,
                emp.dataAdmissão
    FROM (SELECT * FROM EMPREGADO
          WHERE dataAdmissão<'20-02-2000'
    AND codCat=3) as emp
```

ou

nomeEmp	dataAdmissão
Carlos	6-Abr-98

```
SELECT DISTINCT nomeEmp,dataAdmissão
                FROM EMPREGADO
WHERE dataAdmissão<'20-02-2000' AND
                codCat=3
```

SQL

Considere os seguintes Esquemas de Relação e as respectivas Relações

ALUNO (numeroAluno, nomeAluno, dataNascimentoAluno)

DOCENTE (numeroDocente, nomeDocente, dataNascimentoDocente)

ALUNO

numeroAluno	nomeAluno	dataNascAluno
12345	Antonio	10-10-1970
43321	Nuno	21-09-1970
12231	Maria	12-05-1976

DOCENTE

numeroDocente	nomeDocente	dataNascDocente
12345	Antonio	10-10-1970
11223	Felizberta	21-09-1961

SQL

União

A operação de união de duas Relações R1 e R2 tem como resultado uma Relação R, contendo os tuplos de R1 e R2

Os tuplos repetidos são removidos

Esta operação só é válida se R1 e R2 tiverem Esquemas compatíveis

SQL

União

É uma operação comutativa: $R_1 \cup R_2 = R_2 \cup R_1$

Formalmente escreve-se

$$R_1 \cup R_2 = \{t: t \in R_1 \vee t \in R_2\}$$

Quais o conjunto de todos os alunos e docentes?

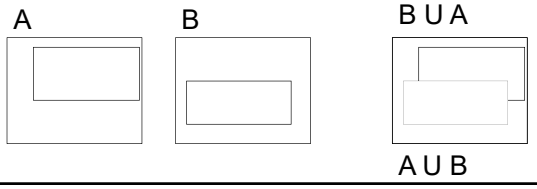
ALUNO ∪ DOCENTE

12345	Antonio	10-10-1970
43321	Nuno	21-09-1970
12231	Maria	12-05-1976
11223	Felizberta	21-09-1961

SQL

Operações Algébricas – União

Na operação União, os Esquemas de Relação têm de ser compatíveis, isto é, têm que ter o mesmo grau e os atributos terem o mesmo domínio



SQL

Operações Algébricas – União

Para os Esquemas de Relação $A(A_1,A_2)$, $B(B_1,B_2)$

- *Em álgebra relacional:*
  - $A \cup B$
- *Em SQL:*
  - SELECT * FROM A UNION SELECT * FROM B

- É garantido que o resultado é um conjunto: *não existem duplicados*
- Se os duplicados são desejados, então usar UNION ALL

SQL

Operações Algébricas – União (continuação)

- *Questão:* pretende-se saber não só os nomes dos empregados do departamento com o código 2, mas também os nomes dos empregados que entraram ao serviço depois de 1998.

```
SELECT nome FROM EMPREGADO
  WHERE codDep=2
  UNION ALL
  SELECT nome FROM EMPREGADO
  WHERE dataAdmissão >= '01-01-1999'
```

SQL

Operações Algébricas – União (continuação)

- SELECT nome FROM EMPREGADO
  WHERE codDep=2
  UNION ALL
  SELECT nome FROM EMPREGADO
  WHERE dataAdmissão >= '01-01-1999'

OU

- SELECT nome FROM EMPREGADO
  WHERE codDep=2 **OR** dataAdmissão>='01-01-1999'

Atenção ao texto!

pretende-se saber não só os nomes dos empregados do departamento com o código 2, mas também os nomes dos empregados que entraram ao serviço depois de 1998.

pretende-se saber os nomes dos empregados do departamento com o código 2, e os nomes dos empregados que entraram ao serviço depois de 1998.

e ou

SQL

Diferença

A operação de diferença entre duas Relações R_1 e R_2 tem como resultado uma Relação R, contendo os tuplos de R_1 que não constam de R_2

Esta operação só é válida se R_1 e R_2 tiverem Esquemas compatíveis

SQL

Diferença

Formalmente escreve-se

$$R_1 - R_2 = \{t: t \in R_1 \wedge t \notin R_2\}$$

Qual o conjunto de todos os alunos que não são docentes?

ALUNO - DOCENTE

43321	Nuno	21-09-1970
12231	Maria	12-05-1976

SQL

Diferença - características

Ao contrário da operação união, a operação diferença não é comutativa: $R_1 - R_2 \neq R_2 - R_1$

Para o exemplo:

ALUNO - DOCENTE tem como resultado todos os alunos que não são docentes

43321	Nuno	21-09-1970
12231	Maria	12-05-1976

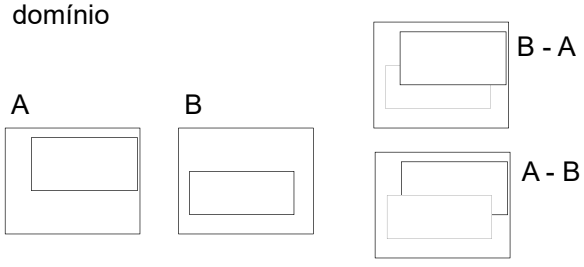
DOCENTE -ALUNO tem como resultado todos os docentes que não são alunos

11223	Felizberta	21-09-1961

SQL

Operações Algébricas – Diferença

- Na operação de Diferença, os Esquemas de Relação têm de ser compatíveis, ou seja, terem o mesmo grau e os atributos terem o mesmo domínio



SQL

Operações Algébricas – Diferença

Para os Esquemas de Relação $A(A_1,A_2)$, $B(B_1,B_2)$

Questão: O que pertence a A mas não pertence a B

Em álgebra relacional: A - B

Em SQL:

SELECT * FROM A EXCEPT SELECT * FROM B

*SELECT * FROM A MINUS SELECT * FROM B*

– (MINUS para versões anteriores à SQL92)

O resultado é um conjunto – não existem duplicados

SQL

Intersecção

A operação intersecção entre duas Relações R_1 e R_2 tem como resultado uma Relação R constituída pelos tuplos comuns a R_1 e R_2

Esta operação só é válida se R_1 E R_2 tiverem Esquemas compatíveis

SQL

Intersecção

Formalmente escreve-se

$$R_1 \cap R_2 = \{t: t \in R_1 \wedge t \in R_2\}$$

A intersecção pode ser feita recorrendo à diferença:

$$R1 – (R1 – R2)$$

Qual o conjunto de todos os alunos que também são docentes?

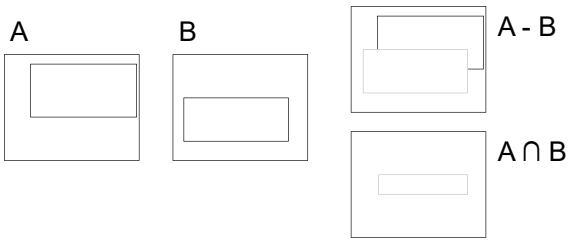
ALUNO ∩ DOCENTE

12345	Antonio	10-10-1970

SQL

Operações Algébricas – Intersecção

Na operação de Intersecção, os Esquemas de Relação têm de ser compatíveis, ou seja, terem o mesmo grau e os atributos terem o mesmo domínio



SQL

Operações Algébricas – Intersecção

Para os Esquemas de Relação $A(A_1,A_2)$, $B(B_1,B_2)$

- Questão: Quais as partes comuns entre A e B
- *Em álgebra relacional:* A ∩ B
- *Em SQL:*

SELECT * FROM A

INTERSECT

SELECT * FROM B

É garantido que o resultado é um conjunto:

não existem duplicados

SQL

União, diferença, intersecção: características comuns

São operações binárias, ou seja, são operações efectuadas sobre duas Relações

Os atributos da Relação resultante podem ser identificados pelo seu índice

Para que as operações possam ser aplicadas, as Relações operando têm de ter Esquemas compatíveis

SQL

União, diferença, intersecção: características comuns

Os Esquemas são compatíveis quando:

Têm o mesmo grau

Os atributos têm o mesmo domínio

O Esquema da Relação obtida é igual ao das relações operando

SQL

Operações Algébricas – União, Intersecção, Diferença

- Nestas três operações, por omissão, não são admitidos duplicados no resultado
- Num SELECT, pelo contrário, o comportamento por omissão é o de admitir duplicados

SQL

Operações Algébricas – União, Intersecção, Diferença

- É necessário ter em atenção que os SGBD fazem por vezes conversões implícitas de dados, para torná-los compatíveis
- Nesses casos é necessário consultar as tabelas de conversões e respectivas prioridades

SQL

Operações Algébricas – União, Intersecção, Diferença

- Tipicamente:
  - as cadeias de caracteres são convertidas na que tiver maior dimensão
  - os tipos numéricos são convertidos no que tiver maior precisão

SQL

Operações Algébricas – União, Intersecção, Diferença

- Dependendo do SGBD e da sua implementação da norma, algumas destas operações podem não estar disponíveis (ex: *Intersect*)

SQL

Considere os seguinte Esquemas de Relação e as respectivas Relações

BANDA (codigo, nome, anoFormacao, genero)
GENERO (codigo, designacao)

BANDA

Codigo	Nome	AnoFormacao	Genero
1	Metallica	1981	1
2	Madredeus	1991	2
3	Iron Maiden	1976	1
4	The Platters	1953	3

GENERO

Codigo	Designacao
1	Metal
2	Fado
3	Rock

SQL

Produto cartesiano

A operação produto cartesiano tem como argumentos duas Relações R_1 e R_2, tendo como resultado uma Relação R constituída pelas combinações possíveis dos tuplos de R_1 com R_2 *(por esta ordem)*

Formalmente escreve-se

$$R_1 \text{ X } R_2 = \{t_1, t_2: t_1 \in R_1 \wedge t_2 \in R_2\}$$

SQL

Produto cartesiano

Se o grau de R_1 for G_1 e o de R_2 for G_2, então

O grau da Relação resultante de R_1 X R_2 será igual a $G_1 + G_2$

Se a cardinalidade de R_1 for C_1 e a de R_2 for C_2, então

A cardinalidade da Relação resultante de R_1 X R_2 será igual a $C_1 * C_2$

SQL

- Questão: Combinar os tuplos de BANDA com os de GENERO
- *Em álgebra relacional:* BANDA x GENERO
- *Em SQL:* SELECT * FROM BANDA,GENERO

ou

SELECT * FROM BANDA CROSS JOIN GENERO

Codigo	Nome	AnoFormacao	Genero	Codigo	Designacao
1	Metallica	1981	1	1	Metal
1	Metallica	1981	1	2	Fado
1	Metallica	1981	1	3	Rock
2	Madredeus	1991	2	1	Metal
2	Madredeus	1991	2	2	Fado
2	Madredeus	1991	2	3	Rock
3	Iron Maiden	1976	1	1	Metal
3	Iron Maiden	1976	1	2	Fado
3	Iron Maiden	1976	1	3	Rock
4	The Platters	1953	3	1	Metal
4	The Platters	1953	3	2	Fado
4	The Platters	1953	3	3	Rock

SQL

Operações Algébricas – Produto Cartesiano (continuação)

- Com foi visto anteriormente, o uso do Produto Cartesiano pode não ter muito interesse do ponto de vista prático, visto a combinação da informação ser feita sem nenhum critério.
- No entanto, quando aplicada uma Selecção sobre um produto cartesiano, o interesse é obvio

SQL

Operações Algébricas – Produto Cartesiano (continuação)

- Questão: Qual o género musical de cada banda

Em álgebra relacional:

– $\sigma_{\text{genero=codigo}}$ **(BANDA x GENERO)**

Em SQL:

– SELECT * FROM BANDA,GENERO
WHERE BANDA.genero=GENERO.codigo

Codigo	Nome	AnoFormacao	Genero	Codigo	Designacao
1	Metallica	1981	1	1	Metal
2	Madredeus	1991	2	2	Fado
3	Iron Maiden	1976	1	1	Metal
4	The Platters	1953	3	3	Rock

SQL

Junção

A operação de junção entre duas Relações R_1 e R_2 é definida da seguinte forma:

$$R_1 \text{ <condição junção> } R_2$$

A condição de junção pode ser constituída pela conjunção de várias condições, cada uma da forma

$$i \; \theta \; j$$

sendo i o i-ésimo atributo de R_1
e j o j-ésimo de R_2

<u>i e j têm o mesmo domínio</u>

SQL

Junção

i θ j

O operador θ pode ser qualquer um dos seguintes operadores de comparação:

<, >, =, ≥, ≤, <>

A operação junção definida desta forma designa-se de Junção Teta

SQL

Junção

A operação de junção tem uma analogia directa com o encadeamento de duas operações:

(uma Seleção sobre um Produto cartesiano)

$$R_1 \bowtie_{i\ \theta\ j} R_2 = \sigma_{A_i\ \theta\ A_{(N+j)}} (R_1 x R_2),$$

onde N é o grau de R_1

Os tuplos que tiverem o valor NULL nos atributos de junção (i e j) não são contabilizados, ou seja, não pertencem à Relação resultante

SQL

Junção

Do exemplo anterior

BANDA $\bowtie_{\text{genero=codigo}}$ GENERO

Codigo	Nome	AnoFormacao	Genero	Codigo	Designacao
1	Metallica	1981	1	1	Metal
2	Madredeus	1991	2	2	Fado
3	Iron Maiden	1976	1	1	Metal
4	The Platters	1953	3	3	Rock

SQL

Operações Algébricas – Junção

Questão: Qual o género musical de cada banda?

Em álgebra relacional:

BANDA $\bowtie_{\text{genero=codigo}}$ GENERO

Em SQL:

– SELECT * FROM BANDA **INNER JOIN** GENERO **ON** (BANDA.genero=GENERO.codigo)

SQL

Operações Algébricas – Junção

SELECT * FROM BANDA **INNER JOIN** GENERO **ON** (BANDA.genero = GENERO.codigo)

ou *por omissão é inner*

SELECT * FROM BANDA **JOIN** GENERO **ON** (BANDA.genero = GENERO.codigo)

Codigo	Nome	AnoFormacao	Genero	Codigo	Designacao
1	Metallica	1981	1	1	Metal
2	Madredeus	1991	2	2	Fado
3	Iron Maiden	1976	1	1	Metal
4	The Platters	1953	3	3	Rock

SQL

Operações Algébricas – Junção (continuação)

- A operação de Join tem o formato geral:

SELECT <atributos> | *

FROM <tabela1> [NATURAL] [<tipoJunção>]

JOIN <tabela2> [ON <condição>]

- *tipoJunção* pode ter os seguintes valores:
  - INNER
  - LEFT [OUTER]
  - RIGHT [OUTER]
  - FULL [OUTER]
- LEFT, RIGHT, FULL são Junções Externas
- Quando omitida é considerada INNER

SQL
Ambiguidade na identificação de Atributos

Quando são especificadas mais do que uma tabela num comando SELECT, por vezes existe ambiguidade na identificação dos atributos

SQL
Ambiguidade na identificação de Atributos

- Pretende-se saber o código do empregado e o nome do departamento onde ele trabalha:
  - Empregado(codigo, nome, codigoDepart)
  - Departamento(codigo, nome)
  - $\pi_{1,2}$ (Empregado $\bowtie_{3=1}$ Departamento)

SELECT codigo, nome
FROM Empregado INNER JOIN Departamento On (codigoDepart=codigo)

Será que a implementação em SQL está correcta?

SQL
Ambiguidade na identificação de Atributos (continuação)

- A resposta é não!!

Existem várias ambiguidades:
- *Codigo* é um atributo de que tabela?
- *Nome* é um atributo de que tabela?
- Quando se compara *codigo* a *codigoDepart*, *codigo* é um atributo de que tabela?

SQL
Ambiguidade na identificação de Atributos (continuação)

- Correctamente
  - Empregado(codigo, nome, codigoDepart)
  - Departamento(codigo, nome)

```
SELECT Empregado.codigo, Empregado.nome
FROM Empregado INNER JOIN Departamento
On (Empregado.codigoDepart =
                    Departamento.codigo)
```

SQL
Ambiguidade na identificação de Atributos (continuação)

- Correctamente
  - Empregado(codigo, nome, codigoDepart)
  - Departamento(codigo, nome)

ou

```
SELECT Emp.codigo, Emp.nome
FROM Empregado as Emp INNER JOIN
Departamento as Dep
 On (Emp.codigoDepart = Dep.codigo)
```

SQL
Ambiguidade na identificação de Atributos (continuação)

- Mesmo quando apenas uma única tabela está envolvida num SELECT podem existir ambiguidades

SQL

Ambiguidade na identificação de Atributos (continuação)

- Considere-se o seguinte Esquema Relacional

EMPREGADO(<u>numBI</u>, primNome, ultNome, numBIChefe)

Questão: Para cada empregado, pretende-se saber o seu primeiro e último nome e também o primeiro e último nome do seu chefe

SQL

Ambiguidade na identificação de Atributos (continuação)

- Questão: Para cada empregado, pretende-se saber o seu primeiro e último nome e também o primeiro e último nome do seu chefe
  - $\pi_{2,3,6,7}$ (Empregado $\bowtie_{1=4}$ Empregado)

```
SELECT Emp.primNome, Emp.ultNome,
       Chefe.primNome, Chefe.ultNome
FROM Empregado as Emp, Empregado as Chefe
   WHERE Emp.numBIChefe = Chefe.numBI
```

SQL

Combinação de operações – Junção, Projecção, Selecção

Por vezes, a interrogação que se pretende leva a que sejam combinadas um conjunto de operações

- Material(nome, codigoMaterial)
- Fornece(codigoFornecedor, codigoMaterial)

Questão: Qual o nome dos materiais fornecidos pelo fornecedor de código 123?

SQL

Combinação de operações – Junção, Projecção, Selecção

Questão: Qual o nome dos materiais fornecidos pelo fornecedor de código 123?

Material(nome, codigoMaterial)
Fornece(codigoFornecedor, codigoMaterial)

Em Álgebra Relacional:

- $\pi_{[1]}$ ($\sigma_{[3]=123}$ (Material $\bowtie_{[2]=[2]}$ Fornece))

SQL

Combinação de operações – Junção, Projecção, Selecção

Material(nome, codigoMaterial)
Fornece(codigoFornecedor, codigoMaterial)

Em SQL

```
SELECT Material.nome FROM Fornece INNER
JOIN Material
ON(Fornece.codigoMaterial=Material.codigoMaterial)
   WHERE Fornece.codigoFornecedor=123
```

SQL

Combinação de operações – Junção, Projecção, Selecção

- Outra solução para a mesma questão pode ser:

Em Álgebra Relacional

- $\pi_{[1]}$ (Material $\bowtie_{[2]=[2]}$ ($\sigma_{[1]=123}$ (Fornece))

Em Sql

```
SELECT Material.nome FROM Material
INNER JOIN (SELECT * FROM Fornece
WHERE Fornece.codigoFornecedor=123) as F
   ON(F.codigoMaterial=Material.codigoMaterial)
```

Nesta resolução, os tuplos usados na junção são minimizados, pois a selecção é efectuada antes da junção!

SQL

Operadores de comparação

- Quando na construção de um junção, não é necessário que o operador de comparação usado seja o operador '='
- Podem ser usados quaisquer operadores de comparação e predicados (referidos mais adiante)
- Operadores de comparação
  - '=','>','<','>=','<=','<>'

SQL

Operadores de comparação

Aluno(num, nome) Disciplina(codigo, nome)

Classificação(numAluno, codDisciplina, nota)

- Quais as notas positivas do aluno 123?
-

SELECT d.nome, c.nota FROM Classificação as c INNER JOIN disciplina AS d

- ON (d.codigo = c.codDisciplina AND c.nota > 10) inner join Aluno AS a ON (a.num = c.numAluno) WHERE a.num=123

SQL

Junções encadeadas

- Seja
  - Material(codigoMaterial, nome)
  - Fornecedor(codigoFornecedor, nome)
  - Fornece(codigoFornecedor, codigoMaterial)
- Qual o nome do fornecedor e os nomes dos materiais por ele fornecido?

SQL

Junções encadeadas

Material(codigoMaterial ,nome)

Fornecedor(codigoFornecedor, nome)

Fornece(codigoFornecedor, codigoMaterial)

SELECT Fornecedor.nome, Material.nome

FROM Fornecedor inner join Fornece

on(Fornecedor.codigoFornecedor = Fornece.codigoFornecedor) inner join Material on(Fornece.codigoMaterial=Material.codigoMaterial)

SQL

Junção Externa

Nas operações de junção apresentadas, os tuplos de uma Relação R_1 que não tenham associação noutra Relação R_2 não constavam da Relação resultante

Será que este comportamento é sempre desejado?

SQL

Junção Externa

- Para resolver esta particularidade foi criada uma operação de junção que aplicada a duas relações R_1 e R_2 onde:
- Os tuplos que têm associação são incluídos no resultados, como o resultado de uma junção
- Os outros são igualmente incluídos na Relação resultante, sendo completados com NULL nos atributos inexistentes
- Esta operação designa-se de Junção Externa:

$$R_1 ⟕_{I\ \theta\ J} R_2$$

SQL

Operações Algébricas – Junção Externa

Questão: Quais os códigos de todos os fornecedores e o de todos os produtos, por eles fornecidos ou não

Em Álgebra Relacional (Material $\bowtie_{[2]=[2]}$ Fornece)

Em Sql

```
SELECT Material.*, Fornece.codFornecedor
    FROM Material FULL OUTER JOIN Fornece
ON(Fornece.codigoMaterial=Material.codigoMaterial)
```

SQL

Para as relações R1 e R2

R1

Codigo	Marca	Categoria
12345	Toyota Corolla	2
18904	Honda Accord	3
14566	Mercedes E3500	4

R2

Codigo	Descricao
1	1000CC
2	1500CC
3	2000CC
4	2500CC

Quais as marcas de automóveis e categorias existentes, e para cada automóvel as categorias superiores à sua?

$R1 \bowtie_{[3]<[1]} R2$

Codigo	Marca	Categoria	Codigo	Descricao
12345	Toyota	2	3	2000CC
12345	Toyota	2	4	2500CC
18904	Honda	3	4	2500CC
14566	Mercedes	4	NULL	NULL
NULL	NULL	NULL	1	1000CC
NULL	NULL	NULL	2	1500CC

SQL

Em SQL:

```
SELECT * FROM Carro FULL OUTER JOIN
Categoria ON(Carro.categoria<Categoria.codigo)
```

12345	Toyota	2	3	2000
12345	Toyota	2	4	2500
14566	Mercedes	4	NULL	NULL
18904	Honda	3	4	2500
NULL	NULL	NULL	1	1000
NULL	NULL	NULL	2	1500

```
SELECT * FROM Carro INNER JOIN
Categoria ON(Carro.categoria<Categoria.Codigo)
```

12345	Toyota	2	3	2000
12345	Toyota	2	4	2500
18904	Honda	3	4	2500

SQL

Junção Externa à Esquerda e Direita

Por vezes apenas os tuplos de uma das Relações se querem manter, numa Junção Externa

Na operação de Junção à Esquerda (*Left Outer Join*) apenas os tuplos da Relação R_1, à esquerda do operador, são considerados no resultado

$$R_1 \bowtie_{i\,\theta\,j} R_2$$

SQL

Junção Externa à Esquerda e Direita

Na operação de Junção à Direita (*Right Outer Join*) apenas os tuplos da Relação R_2, à direita do operador, são considerados no resultado

$$R_1 \bowtie_{i\,\theta\,j} R_2$$

SQL

Junção Externa à Esquerda e Direita

Para as relações anteriores, pretende-se saber quais as categorias existentes e para essas quais os veículos existentes

Ou seja

$R_1 \bowtie_{3=1} R_2$

R1

Codigo	Marca	Categoria
12345	Toyota Corolla	2
18904	Honda Accord	3
14566	Mercedes E3500	4

R2

Codigo	Descricao
1	1000CC
2	1500CC
3	2000CC
4	2500CC

Codigo	Marca	Categoria	Codigo	Descricao
12345	Toyota Corolla	2	2	1500CC
12345	Honda Accord	3	3	2000CC
12345	Mercedes E3500	4	4	2500CC
NULL	NULL	NULL	1	1000CC

SQL

SELECT * FROM Carro RIGHT OUTER JOIN Categoria ON(carro.categoria = categoria.codigo)

NULL	NULL	NULL	1	1000
12345	Toyota	2	2	1500
18904	Honda	3	3	2000
14566	Mercedes	4	4	2500

SQL

Para as relações anteriores, pretende-se saber quais os veículos existentes e para estes quais as categorias maiores que as deles

Ou seja

$$R_1 \overset{\triangleright}{\bowtie}_{3<1} R_2$$

SELECT * FROM Carro LEFT OUTER JOIN Categoria ON (carro.categoria<categoria.codigo)

12345	Toyota	2	3	2000
12345	Toyota	2	4	2500
14566	Mercedes	4	NULL	NULL
18904	Honda	3	4	2500

SQL

SELECT * FROM Carro FULL OUTER JOIN Categoria ON(Carro.categoria<Categoria.codigo)

Pode também ser escrito

SELECT * FROM carro LEFT OUTER JOIN Categoria ON(Carro.categoria<Categoria.codigo)

UNION

SELECT * from Carro RIGHT OUTER JOIN Categoria ON(Carro.categoria<Categoria.codigo)

SQL

Operações Algébricas – Junção Externa (continuação)

EMPREGADO(codEmp, nome, codCategoria)

CATEGORIA(codCategoria, designacao, ordenado)

Questão: Quais os empregados e categorias existentes e para cada empregado qual as categorias superiores à sua?

SQL

Operações Algébricas – Junção Externa (continuação)

EMPREGADO(codEmp, nome, codCategoria)

CATEGORIA(codCategoria, designacao, ordenado)

Em Álgebra Relacional

– Empregado $\overset{\triangleright}{\bowtie}_{[3]<[1]}$ Categoria

Em SQL

SELECT Empregado.*, Categoria.*
FROM Empregado FULL JOIN Categoria
ON(Empregado.codCategoria < Categoria.codCategoria)

SQL

Operações Algébricas – Junção Externa à Esquerda

Questão: Quais os empregados e as categorias inferiores á sua?

Em Álgebra Relacional

Empregado $\overset{\triangleright}{\bowtie}_{[3]<[1]}$ Categoria

Em SQL

– SELECT Empregado.*,Categoria.*
FROM Empregado LEFT JOIN Categoria
ON (Empregado.codCategoria > Categoria.codCategoria)

SQL

Operações Algébricas – Junção Externa à Esquerda

Questão: Quais categorias existentes e seus empregados ?

Em Álgebra Relacional

– Empregado $\overset{\bowtie}{\circ}_{[3]=[1]}$ Categoria

Em SQL

– SELECT Categoria.*, Empregado.*
FROM Empregado RIGHT JOIN Categoria
ON (Empregado.codCategoria =
Categoria.codCategoria)

SQL

1. Considere os seguintes Esquemas de Relação

Cliente(cod_cliente, nome, profissao, localidade)

Agencia(cod_agencia, nome, localidade)

Conta(num_conta, tipo, cod_cliente, cod_agencia, saldo)

Emprestimo(num_emprestimo, cod_cliente, cod_agencia, valor)

cod_cliente é uma chave estrangeira para cliente.cod_cliente
cod_agencia é uma chave estrangeira para agencia.cod_agencia

SQL

1.1 Implemente em SQL a criação das tabelas

1.2. Implemente cada uma das seguintes alíneas em Álgebra Relacional e SQL

a) Qual o código e o nome de todos os clientes do banco?
b) Quais os clientes que residem em Lisboa?
c) Quais os clientes (nome e código) que têm conta na agência com código 1?
d) Quais os clientes que residem na mesma localidade da agência onde têm contas?
e) Quais os clientes com empréstimos superiores a 5000€
f) Quais os clientes com as mesma profissão do cliente com o código 1234?

SQL

```
CREATE TABLE Cliente (
   cod_cliente int Primary key,
   nome varchar(50),
   profissão varchar(50),
   localidade varchar(20)
)

CREATE TABLE Agencia (
   cod_agencia int Primary key,
    nome varchar(50),
    localidade varchar(20)
)
```

SQL

```
CREATE TABLE Conta(
   num_conta int Primary Key,
   tipo int check (tipo between 1 and 3),
   cod_cliente int Foreign key REFERENCES
                              Cliente(cod_cliente),
   cod_agencia int Foreign key REFERENCES
                           Agencia(cod_agencia),
   saldo real
)
```

SQL

```
CREATE TABLE Emprestimo (
   num_emprestimo int Primary Key,
   cod_cliente int Foreign key REFERENCES
                              Cliente(cod_cliente),
   cod_agencia int Foreign key REFERENCES
                           Agencia(cod_agencia),
   valor real
)
```

SQL

a) Qual o código e o nome de todos os clientes do banco?

```
Select cod_cliente, nome
                From Cliente
```

7777	Carlos
9876	Maria
12345	Pedro
13579	António
65432	Raquel
67890	Luís
123987	Júlio
878787	Margarida

b) Quais os clientes que residem em Lisboa?

```
select cod_cliente,nome from
  cliente where localidade = 'lisboa'
```

7777	Carlos
13579	António
67890	Luís

SQL

c) Quais os clientes (nome e código) que têm conta na agência com código 1?

12345	Pedro
67890	Luis
67890	Luis

```
SELECT cli.cod_cliente,cli.nome
FROM cliente AS cli, conta AS co
WHERE co.cod_agencia = 1 AND
                cli.cod_cliente = co.cod_cliente
UNION
SELECT cli.cod_cliente,cli.nome
FROM cliente AS cli, emprestimo AS e
WHERE e.cod_agencia = 1 AND
                cli.cod_cliente = e.cod_cliente
```

SQL

c) Quais os clientes (nome e código) que têm conta na agência com código 1?

OU

12345	Pedro
67890	Luis
67890	Luis

```
SELECT cli.cod_CLIENTE, cli.nome
FROM cliente AS cli
INNER JOIN conta AS co
            ON (cli.cod_cliente=co.cod_cliente)
INNER JOIN EMPRESTIMO AS e
            ON (e.cod_cliente = cli.cod_cliente)
WHERE (co.cod_agencia=1 OR
                        e.cod_agencia=1)
```

SQL

d) Quais os clientes que residem na mesma localidade da agência onde têm contas?

```
SELECT DISTINCT cliente.cod_cliente,
            cliente.nome, cliente.localidade,
                        agencia.localidade
FROM cliente,conta,agencia
WHERE conta.cod_agencia =
                    agencia.cod_agencia AND
        conta.cod_cliente = cliente.cod_cliente AND
            agencia.localidade = cliente.localidade
```

SQL

d) Quais os clientes que residem na mesma localidade da agência onde têm contas?

OU

```
SELECT DISTINCT cliente.cod_cliente,
        cliente.nome, cliente.localidade as c_loc,
                    agencia.localidade as a_loc
FROM cliente INNER JOIN conta
      ON(conta.cod_cliente = cliente.cod_cliente)
INNER JOIN agencia
  ON (conta.cod_agencia = agencia.cod_agencia)
WHERE agencia.localidade = cliente.localidade
```

SQL

Cod_Cliente	Nome	C_LOC	A_LOC
7777	Carlos	Lisboa	Lisboa
9876	Maria	Porto	Porto
12345	Pedro	Sacavem	Sacavem
13579	Antonio	Lisboa	Lisboa
65432	Raquel	Coimbra	Coimbra
67890	Luis	Lisboa	Lisboa
123987	Julio	Sacavem	Sacavem
878787	margarida	Coimbra	Coimbra

SQL

e) Quais os clientes com empréstimos superiores a 5000€

```
SELECT  cliente.cod_cliente, cliente.nome,
                                    emprestimo.valor
FROM cliente, emprestimo
WHERE emprestimo.cod_cliente =
                          cliente.cod_cliente AND
                          emprestimo.valor>5000
```

12345 Pedro 6789
67890 Luis 8888

SQL

e) Quais os clientes com empréstimos superiores a 5000€

OU

```
SELECT  cliente.cod_cliente, cliente.nome,
                                    emprestimo.valor
FROM cliente INNER JOIN emprestimo
                ON (emprestimo.cod_cliente =
                                    cliente.cod_cliente)
WHERE emprestimo.valor>5000
```

12345 Pedro 6789
67890 Luis 8888

SQL

f) Quais os clientes com as mesma profissão do cliente com o código 12345?

```
SELECT  cli2.cod_cliente, cli2.nome,
                                    cli2.profissão
FROM cliente AS cli1, cliente AS cli2
WHERE (cli1.cod_cliente = 12345 AND
                cli2.profissão = cli1.profissão)
```

12345 Pedro Electricista
123987 Julio Electricista

SQL

f) Quais os clientes com as mesma profissão do cliente com o código 12345?

```
SELECT  cli2.cod_cliente, cli2.nome,
                                    cli2.profissão
FROM cliente AS cli1, cliente AS cli2
WHERE (cli1.cod_cliente=12345 AND
                cli2.profissão = cli1.profissão)
EXCEPT SELECT  cli.cod_cliente, cli.nome,
cli.profissão FROM cliente AS cli
WHERE (cli.cod_cliente=12345)
```

12345 Pedro Electricista
123987 Julio Electricista

SQL

OU

```
SELECT  cli2.cod_cliente, cli2.nome,
                                    cli2.profissão
FROM cliente AS cli1, cliente AS cli2
WHERE (cli1.cod_cliente=12345 AND
                cli2.profissão = cli1.profissão AND
                cli2.cod_cliente <>12345)
```

123987 Julio Electricista

SQL

OU

```
SELECT  cod_cliente, nome, profissão
FROM cliente
WHERE (cod_cliente <> 12345 AND
    profissão = (SELECT profissão FROM cliente
                WHERE cod_cliente=12345
    )
)
```

123987 Julio Electricista

SQL

Funções de agregação

Questão: Quantos empregados existem na empresa?

- Não é possível responder a esta questão com o que foi apresentado até aqui!

SQL

Funções de agregação

- Existem um conjunto de funções que efectuam operações sobre um conjunto de linhas
  - COUNT – conta o número de linhas
  - SUM – efectua o somatório de valores
  - AVG – encontra a média de valores
  - MAX – determina o maior valor
  - MIN – determina o menor valor

Estas funções designam-se funções de agregação

SQL

Funções de agregação - continuação

COUNT(*)

Conta o número total de linhas (incluindo os valores NULL)

SQL

Funções de agregação - continuação

COUNT([DISTINCT] | [ALL] <coluna>)

Conta o número de linhas excluindo as que, para a coluna indicada, têm valor NULL.

Caso se use DISTINCT, não se consideram valores duplicados. Se apenas for indicado o nome da coluna, por omissão é considerado o ALL (os duplicados são incluídos)

SQL

Funções de agregação - continuação

- Para responder à questão:

```
SELECT COUNT (codEmpregado) FROM
                                Empregado
```

De notar que neste caso não é necessário utilizar o DISTINCT, pois a coluna sobre a qual é feita a contagem é chave primária da tabela, ou seja, não admite valores iguais nem NULLs

SQL

Funções de agregação - continuação

- SUM([DISTINCT] | [ALL] <expressão escalar>)

Efectua o somatório dos valores da expressão.

Quando especificado DISTINCT, apenas os valores diferentes são tidos em conta.

Por omissão os duplicados são incluídos (ALL)

SQL

Funções de agregação - continuação

- AVG([DISTINCT] | [ALL] <expressão escalar>)

Efectua a média dos valores da expressão.

Quando especificado DISTINCT, apenas os valores diferentes são tidos em conta.

Por omissão os duplicados são incluídos (ALL).

É equivalente a SUM/COUNT

SQL

Funções de agregação - continuação

- MAX(<expressão escalar>)

  Determina o valor máximo na expressão.

- MIN(<expressão escalar>)

  Determina o valor mínimo na expressão.

SQL

Funções de agregação - continuação

- Estas funções de agregação apenas podem aparecer na cláusula SELECT ou na cláusula HAVING (apresentada mais à frente)
- O argumento (expressão escalar) das funções SUM e AVG tem de ser numérico
- Se o resultado da expressão escalar é vazio, o resultado da função COUNT é zero, o das outras é NULL

SQL

Funções de agregação - continuação

- A expressão escalar não pode ser ela própria um resultado de uma função de agregação:

  SELECT AVG (SUM (QTY))) as QT FROM ..... *A cláusula acima é ilegal!*

- Quando no SELECT aparecem misturados atributos resultantes de funções agregadoras com outros, essa expressão é ilegal!

SQL

GROUP BY

- *A seguinte expressão é ilegal:*

  SELECT Atr1, AVG (Atr2) FROM TAB1
- O resultado de uma selecção de um atributo sobre uma tabela, possivelmente terá vários valores
- O resultado de uma função agregadora é, sempre, um único valor!
- É necessário utilizar um mecanismo de agrupamento
- Existe em SQL forma de fazer isso, utilizando a cláusula GROUP BY

SQL

GROUP BY - continuação

- A cláusula GROUP BY tem a forma:
  - GROUP BY <lista colunas>
  - Onde <lista de colunas> é uma lista de colunas separadas por virgula, sobre as quais será feito o agrupamento

SQL

GROUP BY - continuação

- Questão: qual o maior dos ordenados, para cada departamento?

  – SELECT codigo as Departamento ,
    MAX(ordenado) as MaiorOrdenado
    FROM DEPARTAMENTO
    GROUP BY Departamento

SQL

GROUP BY - continuação

No problema anterior, qual a soma dos depósitos de cada cliente

– SELECT Cod_cliente as Cliente ,
– SUM(saldo) as Saldo_total
  FROM CONTA
  GROUP BY Cliente

SQL

No exemplo anterior a resposta à última questão daria

Cod_cliente	Saldo_total
7777	87654
9876	888
12345	1000,32
13579	7779,99
65432	78965
67890	5356,89
123987	15673
878787	99999

SQL

GROUP BY - continuação

- Quando é especificada a cláusula GROUP BY, na cláusula SELECT apenas podem aparecer:
  - As colunas que são especificadas na cláusula GROUP BY
  - Resultados de funções agregadores
- Qualquer outra coluna especificada na cláusula SELECT dá origem a instruções ilegais

SQL

HAVING

- A cláusula WHERE é verificada para cada linha da tabela, ficando essa linha no resultado final se verificar a condição
- Por vezes, apenas se querem obter resultados sobre grupos quando estes verificam uma determinada condição
- Com a cláusula WHERE não se consegue isso

SQL

HAVING

- Como responder no exemplo anterior à questão: a soma dos depósitos dos clientes com saldo em todas as suas contas maior que 5000 €
- É necessário aplicar uma condição a cada grupo de contas de um dado cliente; só os clientes cujo saldo da soma das contas seja maior que 5000€ deverão aparecer.

SQL

HAVING

- SELECT cod_cliente, sum(saldo) FROM conta AS conta1
- WHERE (
  SELECT sum(saldo) FROM conta as conta2
  WHERE conta1.cod_cliente = conta2.cod_cliente
) > 5000
  GROUP BY cod_cliente

SQL

HAVING - continuação

- Outra forma de indicar a condição é usando a cláusula HAVING
- A cláusula HAVING tem a forma:
  – HAVING <condição>

Onde <condição> é uma expressão cujo resultado é *booleano,* com um <u>único</u> valor por grupo – contem sempre uma função agregadora

SQL

HAVING - continuação

- Para responder à questão colocada:
- SELECT cod_cliente, sum(saldo) as saldo_total
- FROM conta
- GROUP BY cod_cliente
- HAVING sum(saldo) > 5000

cod_cliente	saldo_total
7777	87654
13579	7779,99
65432	78965
67890	5356,89
123987	15673
878787	99999

SQL

ORDER BY

- Por vezes deseja-se que o resultado de uma interrogação venha ordenado por um determinado critério.
- Essa ordenação é feita utilizando a cláusula ORDER BY.
- A cláusula ORDER BY tem a forma:

ORDER BY {<coluna | número da coluna} [ASC|DESC] >

SQL

ORDER BY

•ORDER BY {<coluna | número da coluna} [ASC|DESC] >

–coluna indica o nome da coluna sobre qual a ordenação vai ser feita

–número da coluna indica qual a coluna (posicionalmente) sobre a qual a ordenação irá ser efectuada. Esse número corresponde à ordem que essa coluna tem na cláusula SELECT

–ASC indica uma ordenação ascendente

–DESC indica uma ordenação descendente

- <u>Por omissão, o tipo de ordenação é ascendente</u>

SQL

ORDER BY - continuação

- *Questão:* Qual o nome e o departamento dos funcionários existentes, ordenados alfabeticamente?

  – SELECT nome, codDepartamento as Departamento
  FROM FUNCIONARIO
  ORDER BY nome ASC

SQL

ORDER BY - continuação

- *Questão:* Quais os códigos dos departamentos e o maior dos salários, onde a média seja maior que 1000€, ordenados por ordem decrescente de salários?
  - SELECT codDepartamento, MAX(ordenado)
    FROM DEPARTAMENTO
    GROUP BY codDepartamento
    HAVING AVG(ordenado)>1000
    ORDER BY 2 DESC

SQL

SELECT – A sintaxe com GROUP BY, HAVING e ORDER BY

A sintaxe do SELECT, com a inclusão das cláusulas GROUP BY, HAVING e ORDER BY fica então:

SELECT [DISTINCT] <colunas> | *
FROM <lista tabelas>
[WHERE <condição>]
[GROUP BY <lista colunas>]
[HAVING <condição>]
[ORDER BY {<coluna>|<numCol>}[ASC|DESC]]

SQL

EXEMPLO continuação:

g) Listar as contas (num_conta, saldo) da agencia com o código 1, por ordem crescente do saldo

h) Quantas contas existem no banco?

i) Quantos clientes possuem contas na agencia com o código 1?

j) Listar quantas contas existem em cada agência?

k) Para as agencias com pelo menos 2 contas, listar os valores máximos ,mínimos e médios das contas dessa agência

SQL

g) SELECT num_conta,saldo FROM conta
WHERE cod_agencia=1
ORDER BY saldo ASC

h) SELECT COUNT(num_conta) FROM conta

i) SELECT COUNT(num_conta) FROM conta
WHERE cod_agencia=1

SQL

j) SELECT cod_agencia, COUNT(num_conta) AS contas_agencia FROM conta GROUP BY cod_agencia

k) SELECT cod_agencia , MAX(saldo) AS max , MIN(saldo) AS min, AVG(saldo) AS AVG
FROM conta GROUP BY cod_agencia
HAVING COUNT(num_conta) >=2

SQL

Sub-Interrogação

- Seja:

CATEGORIA(codCat, nome, salarioBase)
DEPARTAMENTO(codDep, nome, localizacao)
EMPREGADO(codEmp, nome, salarioEfectivo, codCat, codDep)

- Qual o nome dos empregados que trabalham no mesmo departamento que o(s) empregado(s) com nome 'João Maria' ?

SQL

Sub-Interrogação

- Qual o nome dos empregados que trabalham no mesmo departamento que o(s) empregado(s) com nome 'João Maria' ?

SELECT E2.nome FROM empregado AS E1
INNER JOIN empregado AS E2
ON (E1.nome='João Maria' AND
E1.codDep=E2.codDep AND
E2.nome <> 'João Maria')

Existe no entanto outra solução possível:
Separar a interrogação em duas partes

SQL

Sub-Interrogação (continuação)

É necessário responder então a duas sub-questões:

- Qual o código do departamento do(s) empregado(s) 'João Maria' ?

  • SELECT DISTINCT E2.codDep
  FROM EMPREGADO AS E2
  WHERE E2.nome = 'João Maria'

SQL

Sub-Interrogação (continuação)

- Qual o nome dos empregados do(s) departamento(s) com o código obtido na interrogação anterior, mas que não são 'João Maria' ?
  - *assumindo que existem cinco empregados com o nome 'João Maria' e que que três deles estão no departamento 4, um está no 7 e outro no 11*
  - SELECT DISTINCT E1.nome
    FROM EMPREGADO AS E1
    WHERE E1.nome <> 'João Maria' AND
    E1.codDep IN (4,7,11)

SQL

Sub-Interrogação (continuação)

- Para responder à questão inicial:
  - SELECT DISTINCT E1.nome
    FROM EMPREGADO AS E1
    WHERE E1.nome <> 'João Maria' AND
    E1.codDep IN
    (SELECT DISTINCT E2.codDep
    FROM EMPREGADO AS E2
    WHERE E2.nome = 'João Maria'
)

SQL

Sub-Interrogação (continuação)

- Para responder à questão inicial:
- Foi utilizada uma sub interrogação (SELECT interior) para responder à questão.
- Foi utilizado o predicado IN (abordado mais adiante) para verificar se um valor pertence ao conjunto.

SQL

Sub-Interrogação não correlacionada

- Numa sub-interrogação não correlacionada, a interrogação interior não necessita de valores da interrogação exterior
- Questão: Quais os códigos e nomes das categorias com menor salário base?
  - SELECT C.codCat, C.nome FROM CATEGORIA AS C
    WHERE C.salarioBase = (SELECT MIN(C.salarioBase) FROM CATEGORIA AS C)

SQL

Sub-Interrogação não correlacionada

```
SELECT C.codCat, C.nome
FROM CATEGORIA AS C
WHERE C.salarioBase = (
  SELECT MIN(C.salarioBase)
  FROM CATEGORIA AS C
)
```

A interrogação interior não depende da exterior

- a interrogação interior é executada em primeiro lugar e apenas uma vez
- a relação devolvida na interrogação interior permite resolver a exterior

SQL

Sub-Interrogação correlacionada

- Numa sub-interrogação correlacionada a interrogação interior necessita de valores da interrogação exterior
- Questão: Quais as categorias cujo salário base é inferior à média dos salários efectivos dos empregados dessas categorias ?

SQL

Sub-Interrogação correlacionada

- Questão: Quais as categorias cujo salário base é inferior à média dos salários efectivos dos empregados dessas categorias ?

```
SELECT C.* FROM CATEGORIA AS C
   WHERE C.salarioBase < (
      SELECT AVG( E.salarioEfectivo)
      FROM EMPREGADO As E
      WHERE E.codCat=C.codCat
   )
```

SQL

Sub-Interrogação correlacionada

- A interrogação interior (SELECT AVG…) depende da exterior
  - a informação da interrogação exterior é passada à interior
  - para cada linha da interrogação exterior é executada a interior

SQL

Predicados

- Quando foi introduzida a sintaxe geral de uma interrogação SQL, uma das cláusulas existentes era o WHERE
- Esta cláusula foi apresentada no contexto de um SELECT, tendo o formato
  - [WHERE <condição>]
- Onde
  - <condição> é uma expressão lógica que define a condição que cada linha da tabela resultante verifica

SQL

Predicados

- Essa condição pode ser
  - Um conjunto de comparações combinadas entre si, e/ou
  - Uma colecção de Predicados combinados entre si
  - A combinação é feita recorrendo aos operadores lógicos AND, OR e NOT
- Cada Predicado quando avaliado produz um valor lógico verdadeiro ou falso

SQL

Predicados - continuação

- Os Predicados podem ser utilizados num contexto estático, sendo avaliados com base em valores constantes.
  - ...WHERE E1.codDep IN (4, 7, 11) ...
- Podem, no entanto, ser usados com base em valores dinâmicos, a retirar da base de dados
  - ...WHERE E1.codDep IN (SELECT E2.codDep FROM EMPREGADO)...

SQL

Os predicados existentes são:

1 - de Comparação

BETWEEN - *WHERE ATR1 BETWEEN 1 AND 5*

LIKE - *WHERE ATR1 LIKE '%123'*

IN – *WHERE ATR1 IN (1,2,3,10)*

ALL, ANY - *WHERE Atr1 > ANY(SELECT …)*

EXISTS - *WHERE EXISTS(SELECT …)*

SQL

Os predicados existentes são:

2 -Teste de valor nulo

IS NULL - *WHERE Atr1 IS NULL*

SQL

Predicados - BETWEEN

- O predicado BETWEEN não é mais que uma forma simplificada de escrever algumas condições
- A sintaxe do BETWEEN
  - <construtor de linha> [NOT] BETWEEN <construtor de linha> AND <construtor de linha>
- Semanticamente:
  - Y BETWEEN X AND Z é equivalente a
    - X <= Y AND Y <=Z

SQL

Predicados - BETWEEN

- Questão: Qual o nome e o código dos empregados que têm o salário efectivo entre 1000€ e 2000€?
  - SELECT nome, codEmp
    FROM EMPREGADO
    WHERE salário_Efectivo BETWEEN 1000 AND 2000

SQL

Construtor de linha

- Um construtor de linha é usado no predicado BETWEEN e noutros abordados de seguida
- Um construtor de linha pode ser:
  - um átomo (por ex. uma coluna)
  - uma expressão que origina uma tabela, entre parêntesis curvos. Nesse caso, o resultado dessa expressão deve ser uma tabela com pelo menos uma linha

SQL

Construtor de linha

- Se forem efectuadas comparações entre construtores de linha que sejam avaliados em tabelas, estas têm de ter o mesmo grau
- A comparação faz-se linha a linha e coluna a coluna

SQL

Construtor de linha

- Seja E e D os construtores de linha Esquerdo e direito e N o seu grau
  - E = D é verdade se Ei=Di, para cada linha e para todo o i, i ∈{1...N}
  - E < D é verdade se Ei<Di, para cada linha e para todo o i , i ∈{1...N}
- Este raciocínio aplica-se aos restantes operadores de comparação

SQL

Predicados – LIKE

- O predicado LIKE é usado para encontrar padrões em cadeias de caracteres
- A sintaxe do LIKE
  - <expressão> [NOT] LIKE <padrão> [ESCAPE <caracterExcepção>]
- <expressão> tem de ser uma cadeia de caracteres

SQL

Predicados – LIKE

- <padrão> é constituído pelo padrão que se quer encontrar na cadeia de caracteres, podendo incluir meta-caracteres, que não sendo precedidos do carácter de excepção, têm o seguinte significado:
  - O símbolo '%', indica qualquer sequência de 0 ou mais caracteres
  - O símbolo '_' indica um carácter qualquer. Pode ser usado várias vezes

SQL

Predicados – LIKE

- caracterExcepção permite colocar na string os caracteres especiais sem que eles adquiram esse significado

SQL

Predicados - LIKE (continuação)

- *Questão:* Qual o nome e código dos empregados que tem 'João' no nome?
  - SELECT nome, codEmp
    FROM EMPREGADO
    WHERE nome LIKE '%João%'

SQL

Predicados - LIKE (continuação)

- *Questão:* Qual o nome e código dos empregados cujo nome começa por 'João'?

  – SELECT nome, codEmp
    FROM EMPREGADO
    WHERE nome LIKE 'João%'

SQL

Predicados - LIKE (continuação)

- *Questão:* Qual o nome e código dos empregados que tem '%' no nome?

- SELECT nome, salário_efectivo
- FROM empregado
- WHERE nome like '%\%% 'ESCAPE '\'

SQL

Predicados – IN

- O predicado IN é usado para verificar se um determinado valor está contido numa determinada lista de valores

- A sintaxe do IN
  – <construtor de linha> [NOT] IN (<sub-interrogação> | <lista de expressões escalares>)

SQL

Predicados – IN

- *Questão:* Qual o nome e código dos empregados que pertencem aos departamentos 2 e 3?

  – SELECT nome, codEmp FROM EMPREGADO AS E1
    WHERE E1.codDep IN (2,3)

SQL

Predicados – IN

- *Questão:* Quais as categorias às quais pertencem empregados com maior salário efectivo?

```
SELECT C.* FROM CATEGORIA AS C WHERE C.codCat IN (
   SELECT E.codCat FROM EMPREGADO AS E
      WHERE E.salarioEfectivo =
           ( SELECT MAX( E.salarioEfectivo )
      FROM EMPREGADO AS E )
)
```

SQL

Predicados – ANY, ALL

- Estes predicados verificam se alguma ou todas as linhas têm um atributo que obedece a uma expressão envolvendo operadores relacionais

- A sintaxe do ANY, ALL

  – <construtor de linha> <operador de comparação> ANY | ALL
    (sub-interrogação)

SQL

Predicados – ANY, ALL

- *Questão:* Qual o nome dos empregados cujo salário efectivo é superior ao de alguns empregados da mesma categoria ?

```
– SELECT E.nome
    FROM EMPREGADO AS E
    WHERE E.salário_Efectivo > ANY
     ( SELECT E1.salário_Efectivo
       FROM EMPREGADO AS E1
       WHERE E.codCat = E1.codCat )
```

SQL

Predicados – ANY, ALL (continuação)

- Qual o nome dos empregados cujo salário efectivo é superior ao de todos os empregados de departamentos localizados em 'Lisboa' ?

```
SELECT E.nome FROM EMPREGADO AS E
WHERE E.salário_Efectivo > ALL (
   SELECT E1.salário_Efectivo
      FROM EMPREGADO AS E1 INNER JOIN
         DEPARTAMENTO AS D
         ON (E1.codDep = D.codDep AND
         D.localização='Lisboa'
  )
```

SQL

Predicados – ANY, ALL (continuação)

- Qual o nome dos empregados cujo salário efectivo é superior ao de todos os empregados de departamentos localizados em 'Lisboa' ?

Ou

```
SELECT E.nome FROM EMPREGADO AS E
WHERE E.salário_Efectivo > (
   SELECT MAX(E1.salário_Efectivo)
   FROM EMPREGADO AS E1 INNER JOIN
   DEPARTAMENTO AS D
   ON (E1.codDep = D.codDep AND
D.localização='Lisboa')          )
```

SQL

Listar o código de Departamento e o número de empregados dos maiores departamentos da empresa.

SQL

Solução 1

```
SELECT D.CodDep, COUNT(E.CodEmp)
FROM Departamento AS D INNER JOIN
Empregado AS E  ON (D.CodDep=E.CodDep)
GROUP BY D.CodDep HAVING
Count(E.CodEmp) = (
   SELECT MAX(K.Cemp) FROM (
      Select D1.CodDep, Count(E1.Codemp) AS
         Cemp FROM Deparatamento AS D1
         INNER JOIN Empregado AS E1 ON
         (D1.CodDep=E1.CodDep)
         GROUP BY D1.CodDep) AS K)
```

SQL

Solução 2

```
SELECT D.CodDep, count(E.CodEmp)
FROM Deparatamento AS D INNER JOIN
Empregado AS E ON (D.CodDep = E.CodDep)
GROUP BY D.CodDep
HAVING COUNT(E.CodEmp)>= ALL (
   SELECT COUNT(E1.codEmp)
    FROM  Deparatamento AS D1
   INNER JOIN Empregado AS E1
    ON (D1.CodDep=E1.CodDep)
    GROUP BY D1.CodDep
)
```

SQL

Predicados – EXISTS

- Este predicado é utilizado para testar se uma determinada tabela tem pelo menos uma linha
- A sintaxe do EXISTS
  - [NOT] EXISTS (sub-interrogação)

SQL

Predicados – EXISTS

- *Questão:* Quais os departamentos que têm pelo menos um empregado?

```
SELECT D.* FROM Departamento as D
   WHERE EXISTS (
   SELECT E.codEmp
   FROM EMPREGADO as E
   WHERE E.codDep=D.codDep
)
```

- Se o resultado da sub-interrogação não for vazio, o predicado EXISTS retorna *true*

SQL

Predicados – EXISTS (continuação)

Responder à mesma questão sem recorrer ao EXISTS

- *Questão:* Quais os departamentos que têm pelo menos um empregado?

```
SELECT D.* FROM DEPARTAMENTO as D
  WHERE 1<=(
     SELECT COUNT(E.codEmp)
     FROM EMPREGADO
     WHERE E.codDep = D.codDep
   )
```

SQL

Predicados – EXISTS (continuação)

- *Questão:* Qual o código e nome das categorias que não têm empregados ?

```
SELECT C.codCat, C.nome
  FROM categoria1 AS C
  WHERE NOT EXISTS (
    SELECT * FROM Empregado AS E
    WHERE E.CodCat = C.CodCat
  )
```

SQL

EXEMPLOS III

m) Quais os clientes da agência com código 1?

n) Quais os clientes cujo saldo total das suas contas é superior a qualquer um dos outros clientes?

o) Quais os clientes com pelo menos um empréstimo no banco e respectivo valor total do empréstimo?

p) Qual o cliente que deve mais ao banco?

q) Quais os clientes que são, simultaneamente, depositantes e devedores na agência com código 1?

SQL

m) Quais os clientes da agência com código 1?

```
SELECT C.*
  FROM CLIENTE as C
  WHERE C.cod_Cliente IN (
     SELECT Cod_Cliente
        FROM Conta WHERE Cod_Agencia =1)
        UNION (
           SELECT Cod_Cliente
           FROM Emprestimo WHERE
           cod_agencia=1)
  )
```

SQL

n) Quais os clientes cujo saldo total das suas contas é superior a qualquer um dos outros clientes?

```
SELECT C.* FROM Cliente AS C
INNER JOIN CONTA  AS CO
ON (C.Cod_Cliente = Co.Cod_Cliente)
GROUP BY C.Cod_Cliente, C.nome, C.profissão,
C.localidade Having SUM(Co.Saldo) >= ALL (
    SELECT SUM(Saldo) FROM Conta
    GROUP BY Cod_CLIENTE
)
```

SQL

n) Quais os clientes cujo saldo total das suas contas é superior a qualquer um dos outros clientes? **ou**

```
SELECT C.* FROM Cliente AS C
INNER JOIN CONTA AS Co
ON (C.Cod_Cliente = Co.Cod_Cliente)
GROUP BY C.Cod_Cliente, C.nome, C.profissão,
                C.localidade, Co.cod_Cliente
Having SUM(Co.Saldo)> ALL (
    SELECT SUM(Co1.Saldo)  FROM Conta as Co1
    Where Co1.Cod_Cliente <> Co.Cod_cliente
    GROUP BY Co1.Cod_CLIENTE
)
```

SQL

o) Quais os clientes com pelo menos um empréstimo no banco e respectivo valor total do empréstimo?

```
SELECT DISTINCT C.Cod_Cliente,SUM(E.valor)
FROM Cliente AS C
INNER JOIN Emprestimo as E
ON(E.Cod_Cliente = C.Cod_Cliente)
GROUP BY C.Cod_Cliente
```

SQL

p) Qual o cliente que deve mais ao banco?

```
Select C.Cod_Cliente, SUM(E.valor)
From Cliente as C JOIN Emprestimo as E
ON(C.Cod_Cliente = E.Cod_Cliente)
Group By C.Cod_cliente
Having SUM(E.valor) >= ALL (
    SELECT SUM(E1.valor)
    FROM Cliente AS C1
    JOIN Emprestimo AS E1
    ON(C1.Cod_Cliente = E1.Cod_Cliente)
    GROUP BY C1.Cod_cliente
   )
```

SQL

q) Quais os clientes que são, simultaneamente, depositantes e devedores na agência com o código 1?

```
SELECT Co.Cod_Cliente FROM Conta as Co
WHERE (Co.Cod_Agencia=1 AND EXISTS (
        SELECT * FROM Emprestimo as E
        WHERE E.Cod_Agencia=1 AND
            E.cod_Cliente=Co.Cod_Cliente
        )
)
```

SQL

Valores NULL – o conceito

- Quando um determinado valor é desconhecido ou indefinido, existe um valor especial para representar isso – o NULL

SQL

Valores NULL – o conceito

- Algumas situações onde o NULL é aplicado:
  - o atributo não é aplicável para determinado tuplo
  - o atributo tem um valor desconhecido para determinado tuplo
  - o atributo tem valor conhecido mas o valor está ausente nesse instante, ou seja, ainda não foi registado na Base de Dados
  - pode ser o valor por omissão para uma determinada coluna

SQL

Valores NULL – o conceito

- Algumas características
  - É independente do domínio - inteiro, real, carácter, data, etc
  - Uma expressão com um operador de comparação será avaliada como FALSE, se algum dos seus operandos tiver o valor NULL
  - Existem funções para determinar se o valor é NULL e alterá-lo

SQL

Manipulação de NULL

- O predicado IS NULL permite determinar se um valor é NULL
- A sintaxe do IS NULL
  - <construtor linha> IS [NOT] NULL

SQL

Manipulação de NULL

- *Questão:* Quais os clientes que têm telefone? (admitindo que esse atributo é opcional)
  - SELECT * FROM CLIENTE WHERE telefone IS NOT NULL

SQL

Manipulação de NULL

- A função NULLIF(X,Y) devolve NULL se X e Y forem iguais. Caso contrário devolve X
- A função COALESCE(X,Y) devolve X se este for diferente de NULL, devolvendo Y caso contrário
- Listar os clientes sem apresentar NULL
  SELECT cliente.nome, COALESCE(cliente.telefone,'Indisponível') FROM CLIENTE

SQL

Atributo Discriminante

- Por vezes, é necessário efectuar um conjunto de interrogações, e ser necessário ter um atributo discriminante que identifique a origem desses dados
- *Seja:*
  - FUNCIONARIO(num,nome,idade)
  - MOTORISTA(num, nCarta, tipoCarta)
  - PROFESSOR(num,grauAcademico)
- *Questão:* Quais os funcionários existentes com indicação da sua profissão

SQL

Atributo Discriminante

- *Questão:* Quais os funcionários existentes com indicação da sua profissão

- SELECT f.num, f.nome, 'motorista' AS profissão
  FROM funcionário AS f
  INNER JOIN motorista AS m
  ON (f.num = m.num)
  UNION
  SELECT f.num, f.nome, 'professor‘
  AS profissão FROM funcionário
  AS f INNER JOIN professor AS p
  ON (f.num = p.num)

SQL

Conversão de valores

- Existem casos em que é necessário apresentar os valores existentes num determinado domínio convertidos para outro:
  - Um determinado atributo é do tipo real mas para um determinado caso é necessário passar esse valor para um conjunto de valores discretos
  - Pode utilizar-se o CASE para efectuar esse mapeamento

SQL

Conversão de valores

- A Sintaxe do CASE:
  - CASE {WHEN <condição> THEN <valor>}+ [ELSE <valor>] END

SQL

Conversão de valores

- *Questão:* Qual o numero, o nome e o tipo de carta dos motoristas?

```
SELECT f.num, f.nome, 'motorista' AS profissão,
CASE WHEN m.tipo='a' THEN 'ligeiros‘
   WHEN m.tipo='b' THEN 'ligeiros com reboque’
   WHEN m.tipo='c' THEN 'pesados mercadorias‘
   WHEN m.tipo='d' THEN 'pesados passageiros'
   ELSE 'pesados com reboque‘
END AS tipo
FROM funcionário AS f INNER JOIN motorista
AS m ON(f.num=m.num)
```

SQL

LMD – comandos de manipulação de dados

- A LMD, não só permite aceder à informação (SELECT), como também permite alterá-la e actualizá-la

- Existem mais três comandos que permitem manipular a informação:
  - INSERT (insere novas linhas numa tabela)
  - UPDATE (actualiza linhas de uma tabela)
  - DELETE (remove linhas de uma tabela)

SQL

LMD – comandos de manipulação de dados

- INSERT (insere novas linhas numa tabela)
  - INSERT INTO <nome da tabela> [(coluna1, coluna2, ...)]
    VALUES (valor1, valor2, ...) | <comando SELECT>

SQL

LMD – comandos de manipulação de dados

– UPDATE (actualiza linhas de uma tabela)

• UPDATE <nome da tabela>
SET coluna = valor | expressão, coluna = valor | expressão, ...
[WHERE <condição>]

SQL

LMD – comandos de manipulação de dados

– DELETE (remove linhas de uma tabela)

• DELETE FROM <nome da tabela>
[WHERE <condição>]

SQL

INSERT

- O comando INSERT é usado para inserir dados numa determinada tabela
- Pretende-se inserir informação na tabela CLIENTE(BI,nome), introduzindo a informação do cliente José Maria com BI 123

– INSERT INTO CLIENTE(NOME,BI)
VALUES('José Maria', 123)

SQL

INSERT

- Nos casos em que o comando INSERT é usado desta forma, apenas é inserido uma linha de cada vez na tabela

– Quando é omitida a lista dos nomes das colunas, os valores tem de ser dados segundo a ordem das colunas definidas na tabela

SQL

INSERT

– Quando é especificada a lista de nomes, os valores têm de estar de acordo com essa lista, embora esta não esteja, necessariamente, pela ordem definida na tabela

– Quando são omitidos valores, é assumido que têm o valor NULL

SQL

INSERT – continuação

- Pretende-se copiar para a tabela com informação histórica dos empregados, todos os empregados dos departamentos de Lisboa.
- O Esquema de Relação que irá conter a informação histórica dos empregados é
- HISTORICO_EMPREGADO(codEmp, nome)

SQL

INSERT – continuação

```
INSERT INTO HISTORICO_EMPREGADO(
  codEmp, nome )
     SELECT codEmp, E.nome
     FROM EMPREGADO AS E INNER JOIN
     DEPARTAMENTO AS D
     ON (E.codDep = D.codDep )
     WHERE localizacao = 'Lisboa'
```

SQL

INSERT – continuação

- Nos casos em que o comando INSERT é usado desta forma
  - o número de colunas na lista da cláusula SELECT tem que ser igual ao número de colunas referidas no comando INSERT e os domínios de colunas correspondentes têm que ser compatíveis

SQL

Com as relações anteriores, pretende-se criar uma relação de profissionais

Profissionais(num, nome, profissão, Ncarta, Grau_academico)

```
INSERT INTO
Profissionais(num, nome, profissão, Ncarta)
SELECT f.num, f.nome, 'motorista' AS profissão,
   m.num_CartaCondução
FROM funcionário AS f INNER JOIN motorista
   AS m ON (f.num = m.num)
```

SQL

ou

```
INSERT INTO Profissionais(num, nome,
                    profissão, Grau_academico)
SELECT f.num, f.nome, 'professor' AS profissão,
                    p.grau_acad
FROM funcionário AS f
INNER JOIN professor AS p
ON (f.num = p.num)
```

SQL

50	pedro	motorista	12434	NULL
100	joao	professor	NULL	licenciado
150	maria	motorista	54321	NULL
200	helena	professor	NULL	mestre
250	joana maria	motorista	09876	NULL
300	miguel	professor	NULL	doutor
350	jose	motorista	34567	NULL
450	manuel	motorista	45678	NULL
550	Antonio	motorista	55555	NULL

SQL

UPDATE

- O comando UPDATE é usado para alterar os dados nas linhas de uma determinada tabela
- Pretende-se registar o facto do empregado com código 123 ter mudado para o departamento 444 cujo chefe tem o código 654
- UPDATE EMPREGADO SET codDep = 444, codEmpChefe = 654
- WHERE codEmp = 123

SQL

UPDATE

- Características do comando UPDATE:
  - se a cláusula WHERE for omitida, todas as linhas da tabela são actualizadas
  - os valores a actualizar podem ser o resultado de expressões ou interrogações à Base de Dados

SQL

DELETE

- O comando DELETE é usado para remover linhas de uma determinada tabela
- Pretende-se remover os empregados do departamento com o código 444
  - **DELETE FROM EMPREGADO WHERE codDep=444**
- Pretende-se remover toda a informação relativa ao histórico dos empregados
  - **DELETE FROM HISTORICO_EMPREGADO**
- Características do comando DELETE:
  - se a cláusula WHERE for omitida, todas as linhas da tabela são removidas

SQL

VIEWS – Vistas sobre os dados

- Por vezes, é necessário, por razões de segurança ou de simplicidade, criar *"tabelas virtuais"* que apresentam os dados numa forma diferente daquela segundo a qual estes estão armazenados
- Usando a terminologia SQL, uma Vista (View)

SQL

VIEWS – Vistas sobre os dados

Uma Vista (View) consiste numa única tabela construída a partir de:

- tabelas
- Vistas anteriormente definidas

Uma Vista apesar de poder ser manipulada como uma tabela, não tem existência física, o que:

- origina algumas limitações às operações de actualização (update)
- mas não limita as operações de interrogação (select)

SQL

VIEWS – continuação

- Vistas simples:
  - construídas com base numa única tabela, não contêm funções nem grupos de dados
- Vistas complexas:
  - construídas com base em várias tabelas, contêm funções ou grupos de dados

SQL

VIEWS – continuação

- Utilidade das Vistas:
  - mostrar apenas parte dos dados (segurança)
  - permitir que os mesmos dados sejam visualizados de diferentes maneiras por diferentes utilizadores (segurança)
  - simplificar a consulta dos dados, substituindo consultas elaboradas envolvendo várias tabelas, por fáceis consultas sobre a Vista
  - reduzir a possibilidade de incoerências (WITH CHECK OPTION)

SQL

VIEWS – continuação

Sintaxe do CREATE, para criação de vistas:

```
CREATE VIEW <nome_vista>
  [ (nome_coluna_1, nome_coluna_2, ...) ]
  AS SELECT comando [WITH CHECK
  OPTION]
```

nome_coluna_i - nome da coluna usado na vista.

Se não for especificado é assumido o mesmo nome das colunas definidas na cláusula SELECT

SQL

VIEWS – continuação

Sintaxe do CREATE, para criação de vistas:

```
CREATE VIEW <nome_vista>
  [ (nome_coluna_1, nome_coluna_2, ...) ]
  AS SELECT comando [WITH CHECK
  OPTION]
```

- A directiva SELECT tem alguma limitações
  - não pode incluir as cláusulas ORDER BY

SQL

VIEWS – continuação

- Seja:

DEPARATAMENTO (codDep, nome, localizacao)

EMPREGADO (codEmp, nome, salário_efectivo, codCat, codDep)

- Pretende-se criar uma vista que permita saber:
  - Para cada departamento quantos empregados existem e qual o montante total de salários

SQL

VIEWS – continuação

```
CREATE VIEW
  INFORMACAO_DEPARTAMENTO
        (CodDep, numEmpregados , totalSalarios )
```

- AS SELECT D.CodDep, COUNT(*), SUM(salário_efectivo)
- FROM DEPARATAMENTO AS D INNER JOIN EMPREGADO AS E
  - ON (D.CodDep = E.CodDep)
- GROUP BY D.CodDep

SQL

EMPREGADO

1	joão Maria	2100	1	3
2	Maria	1200	3	3
3	josé	1600	1	3
4	pedro	2100	2	3
5	anabela	1200	3	1
6	josefina	1300	4	1
7	antónio	900	4	2
8	miguel	800	4	1
9	João Maria	1000	3	2
10	João%Maria	500	3	1
11	maria	600	4	3
12	carlos	1800	5	3
13	antonio	1900	6	3

DEPARATAMENTO

1	Secretaria	Lisboa
2	Tesouraria	Lisboa
3	Oficina	Loures
4	loja	Loures

```
SELECT * FROM
INFORMACAO_DEPARTAMENTO
```

1	4	3800
2	2	1900
3	7	11300

SQL

VIEWS – continuação

- Algumas considerações:
  - A vista não é concretizada no momento em que é criada, mas sempre que é especificada uma interrogação sobre essa vista, mantendo-se assim sempre actualizada
  - As alterações das tabelas originais reflectem-se nas diversas vistas onde essa tabelas são referenciadas

SQL

VIEWS – continuação

INFORMACAO_DEPARTAMENTO, para responder à questão: Para o departamento Oficina, quantos empregados existem e qual o montante total de salários

```
SELECT numEmpregados , totalSalarios
  FROM INFORMACAO_DEPARTAMENTO
  WHERE CodDep= 3
```

7	11300

SQL

```
DELETE FROM EMPREGADO
WHERE CodEmp=4

SELECT numEmpregados , totalSalarios
   FROM INFORMACAO_DEPARTAMENTO
   WHERE CodDep= 3
```

6	9200

SQL

Pretende-se saber qual a soma das diferenças salariais por departamento:

SQL

```
CREATE VIEW EmpregadoCategoria (CodEmp,
CodDep,Diferença_salário) AS SELECT E.CodEmp,
E.CodDep, E.Salário_Efectivo - C.SalarioBase FROM
EMPREGADO AS E JOIN Categoria1 as C
ON(E.CodCat=C.CodCat)

CREATE VIEW ECDepartamento (CodDep,
Soma_Dif_salarial) AS SELECT D.CodDep,
SUM(EC.Diferença_Salário) FROM
EmpregadoCategoria as EC JOIN Deparatamento as
   ON(D.CodDep=EC.CodDep)  GROUP BY D.CodDep

Select * FROM ECDepartamento
```

1	-800
2	-300
3	-800

SQL

Remoção de vistas

- A remoção de vistas é feita utilizando o comando DROP, com a sintaxe:
  - DROP VIEW <nome vista> [RESTRICT | CASCADE]
- Exemplo de remoção da vista INFORMACAO_DEPARTAMENTO
  - DROP VIEW INFORMACAO_DEPARTAMENTO

SQL

Remoção de vistas

- Algumas considerações:
  - a remoção de uma vista não tem qualquer influência nos dados das tabelas que lhe serviam de base
  - se existirem outras vistas que dependam da vista removida
    - A acção falha (dependendo da implementação de cada SGBD)
    - É especificado CASCADE e essas vistas são igualmente removidas

SQL

Remoção de vistas

- Algumas considerações:
  - uma vista só pode ser removida por um utilizador com permissões para efectuar essa operação (ou pelo administrador da BD)

SQL

Actualização de dados sobre vistas

- Se a vista for definida sobre uma única tabela e sem funções de agregação de dados:
  - é uma operação simples que se traduz na actualização da tabela que lhe serve de base
- No entanto se a vista envolver múltiplas tabelas e funções de agregação de dados:
  - é uma operação complicada e que pode ser ambígua

SQL

Actualização de dados sobre vistas

- De uma forma geral, não são actualizáveis as vistas:
  - definidas sobre múltiplas tabelas utilizando junções (join)
  - que utilizam agrupamento de dados e funções de agregação

SQL

Actualização de dados sobre vistas - continuação

- O comando de DELETE não é permitido se a vista incluir:
  - condições de junção (join)
  - funções de agrupamento
  - o comando DISTINCT
  - sub-interrogações correlacionadas

SQL

Actualização de dados sobre vistas - continuação

- O comando de UPDATE não é permitido se a vista incluir:
  - qualquer das limitações do comando de DELETE
  - colunas definidas por expressões (ex: salarioAno = 14 * salario)

SQL

Actualização de dados sobre vistas - continuação

- O comando de INSERT não é permitido se a vista incluir:
  - qualquer das limitações do comando UPDATE;
  - colunas com possibilidade de terem valores NOT NULL que não tenham valores de omissão nas tabelas base e que não estejam incluídas na vista através da qual se pretende inserir novas colunas

SQL

Actualização de dados sobre vistas - continuação

- Os comandos de INSERT e UPDATE são permitidos em vistas contendo várias tabelas base se:
  - o comando afectar apenas uma das tabelas que serve de base à vista

SQL

Actualização de dados sobre vistas - continuação

- Se as vistas forem criadas com a opção WITH CHECK OPTION, algumas das alterações podem não ser possíveis
- Esta opção só permite INSERTs ou UPDATEs sobre vistas se, finalizadas essas acções, o resultado seja visível na vista
- Por outras palavras, as alterações têm de ser compatíveis com as condições especificadas na cláusula WHERE
- Se tal não acontecer, as alterações não são permitidas e o comando é abortado

SQL

```
CREATE VIEW EmpregadoCategoria (CodEmp,
CodDep, Diferença_salário)
AS SELECT E.CodEmp, E.CodDep,
          E.Salário_Efectivo - C.SalarioBase
FROM EMPREGADO AS E
JOIN Categoria1 as C
ON(E.CodCat=C.CodCat)


INSERT INTO EmpregadoCategoria (CodEmp,
CodDep)     VALUES (16, 2)
```

SQL

Actualização de dados sobre vistas - continuação

- Pretende-se criar uma vista que permita saber:
  - Quais os empregados que têm um salário inferior a 2500.
  - Também se pretende que qualquer acção de alteração sobre essa vista apenas afecte os empregados cujo salário seja inferior a 2500

SQL

Actualização de dados sobre vistas - continuação

- Ou seja
  - CREATE VIEW VISTA_EMPREGADO
    AS SELECT cod, nome FROM EMPREGADO
    WHERE salario < 2500
    WITH CHECK OPTION

- As instruções de INSERT e UPDATE sobre esta vista têm que verificar sempre a condição definida na cláusula WHERE

SQL

```
CREATE VIEW EMPREGADO1 (CodEmp, CodCat, CodDep) AS
    SELECT CodEmp, CodCat, CodDep FROM Empregado
    WHERE (CodDep =3)

UPDATE EMPREGADO1 Set CodDep=2 WHERE CodEmp=1

INSERT INTO EMPREGADO1 (CodEmp, CodCat, CodDep) Values (15, 2, 2)

CREATE VIEW EMPREGADO2   (CodEmp, CodCat, CodDep) AS
    SELECT CodEmp, CodCat, CodDep FROM Empregado
    WHERE (CodDep =3) WITH CHECK OPTION

INSERT INTO EMPREGADO2 (CodEmp, CodCat, CodDep) Values (18, 2, 3)

UPDATE EMPREGADO2 Set CodDep=2 WHERE CodEmp=2

INSERT INTO EMPREGADO2 (CodEmp, CodCat, CodDep) Values (17, 2, 2)
```

ERRO

SQL

Tratamento da vistas no SGDB

- O comando CREATE VIEW não origina a execução do comando SELECT a ele associado
- O comando CREATE VIEW apenas origina o armazenamento da definição da vista (directiva SELECT) no dicionário de dados

SQL

Tratamento da vistas no SGDB

- Ao aceder aos dados através de uma vista, o sistema:
  - extrai do dicionário de dados, a definição da vista
  - verifica as permissões de acesso à vista
  - converte a operação sobre a vista numa operação equivalente na tabela ou tabelas que servem de base à vista

Base de Dados

Referências

Modern Database Management – Hoffer JA, Prescott MB, McFadden FR – ISBN 0-13-145320-3

Fundamentals of Database Systems, Elmasri e Navathe, ISBN 0-8053-1755-4

Database Management Systems - Ramakrishnan, R. - McGraw-Hill

http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_60_sqlref#RSf26134

http://www.htmlgoodies.com

http://www.w3schools.com/sql/sql_datatypes.asp