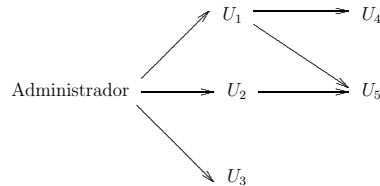


Atribuição de Privilégios

- ▶ A passagem de privilégios de um utilizador para outro pode ser representado por um grafo de autorizações.
- ▶ Os nós do grafo são utilizadores.
- ▶ A raiz é o administrador da base de dados.

Por exemplo: no grafo abaixo um arco $U_i \rightarrow U_j$ indica que o utilizador U_i atribuiu ao utilizador U_j um determinado privilégio.



- ▶ Todos os arcos têm que fazer parte de algum caminho com origem no administrador.
- ▶ Um dado utilizador, que não o administrador, só pode atribuir privilégios a outro utilizador se para isso estiver mandatado, isto é, se tiver o privilégio de atribuir privilégios.
- ▶ O comando GRANT é o comando usado para atribuir privilégios.

2014/09/29 (v96)
173 / 311

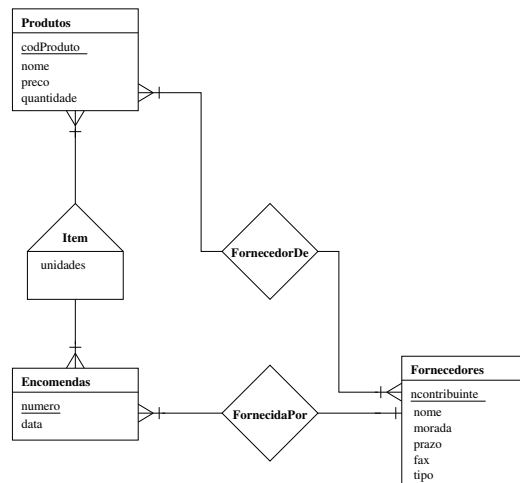
DML — SQL

Linguagem a que o utilizador recorre para obter informação a partir da base de dados.

- ▶ Estrutura básica
- ▶ Operações com conjuntos
- ▶ Funções de agregação
- ▶ Valores nulos
- ▶ Junções
- ▶ Sub-consultas embutidas
- ▶ Relações derivadas
- ▶ Vistas
- ▶ Modificação da Base de Dados
- ▶ União, Diferença, Divisão em SQL

2014/09/29 (v96)
174 / 311

Esquema Utilizado em Exemplos



2014/09/29 (v96)
175 / 311

Pesquisa — Estrutura Básica

- ▶ SQL é baseada em operações de conjuntos e de álgebra relacional com algumas modificações e extensões.
- ▶ Uma consulta SQL básica tem a forma:

```

SELECT  A1, A2, ..., An
FROM    r1, r2, ..., rm
WHERE   P
  
```

- ▶ os A_i s representam atributos;
- ▶ os r_i s representam relações;
- ▶ P é um predicado, nos atributos dos r_i s.
- ▶ A leitura desta instrução é a seguinte: Seleccionar (SELECT) os atributos A_1, A_2, \dots, A_n de entre (FROM) os atributos do produto cartesiano $r_1 \times r_2 \times \dots \times r_m$, sujeitos (WHERE) ao predicado P .
- ▶ Corresponde à expressão da álgebra relacional:

$$\Pi_{A_1, A_2, \dots, A_n}(\sigma_P(r_1 \times r_2 \times \dots \times r_m))$$

2014/09/29 (v96)
176 / 311

A componente SELECT

- ▶ É utilizado para listar os atributos pretendidos no resultado da consulta. Por exemplo, listar o nome e o endereço dos Fornecedores:

```
mysql> SELECT nome,morada FROM Fornecedores;
+-----+-----+
| nome      | morada      |
+-----+-----+
| Futaba    | MARYLAND    |
| Great Planes | Champaign, Illinois, USA |
| Tower Hobbies | Champaign, Illinois, USA |
+-----+-----+
```

- ▶ Um asterisco na cláusula SELECT denota «todos os atributos», por exemplo:

```
SELECT * FROM Loan
```

- ▶ NOTA: O SQL não permite o carácter '-' (hífen) nos identificadores, portanto, por exemplo, balcaoNome em vez de balcao-nome.
- ▶ NOTA: As maiúsculas e minúsculas não são distinguidas em palavras reservadas da linguagem SQL (e.g. SELECT é o mesmo que select).

2014/09/29 (v06)
177/311

A Componente SELECT (cont.)

- ▶ A cláusula SELECT pode conter expressões aritméticas envolvendo as operações, +, -, *, / com argumentos constantes e/ou atributos.
- ▶ Existem funções de agregação que permitem, entre outras, achar o valor mínimo contido num dado atributo (ver-se-à mais à frente).
- ▶ Dependendo das implementações, encontram-se normalmente definidas bibliotecas de funções.

Por exemplo:

```
mysql> SELECT nome,preco*1.23 FROM Produtos;
+-----+-----+
| nome      | preco*1.23  |
+-----+-----+
| Tower Hobbies .46 ABC BB | 90.737897415161 |
| Tower Hobbies .61ABC BB | 102.83789741516 |
| Futaba 9CAFS          | 417.43788818359 |
| Piper J-3 Cub 40      | 145.18789741516 |
+-----+-----+
```

dá-nos o preço dos produtos já com o IVA aplicado.

2014/09/29 (v06)
179/311

A Componente SELECT (cont.)

- ▶ O SQL permite duplicados nas relações e nos resultados de consultas (que não chaves primárias).
- ▶ Para forçar a eliminação de duplicados, inserir a palavra-chave DISTINCT após o SELECT.
Por exemplo: apresentar os nomes de todos os fornecedores sem repetições (o mesmo fornecedor pode fornecer vários tipos de artigos).

```
SELECT DISTINCT nome
FROM Fornecedores
```

- ▶ A palavra-chave ALL (valor por omissão) indica que os duplicados não devem ser removidos.

```
SELECT ALL nome
FROM Fornecedores
```

2014/09/29 (v06)
178/311

A componente FROM

- ▶ A cláusula FROM corresponde à operação de produto cartesiano da álgebra relacional. Indica as relações a consultar na avaliação da expressão.

Por exemplo: listar as encomendas existentes e (produto cartesiano) os fornecedores.

```
mysql> SELECT numero,data,nome,Fornecedores.nContribuinte
FROM Encomendas,Fornecedores;
+-----+-----+-----+-----+
| numero | data          | nome      | nContribuinte |
+-----+-----+-----+-----+
| 1      | 2007-07-04 23:42:00 | Futaba    | 123456780 |
| 2      | 2007-07-04 23:43:00 | Futaba    | 123456780 |
| 1      | 2007-07-04 23:42:00 | Great Planes | 123456781 |
| 2      | 2007-07-04 23:43:00 | Great Planes | 123456781 |
+-----+-----+-----+-----+
```

Ou mais útil, listar as encomendas e respectivos fornecedores

```
mysql> SELECT numero,data,nome,Fornecedores.nContribuinte
FROM Encomendas,Fornecedores
WHERE Encomendas.nContribuinte=Fornecedores.nContribuinte;
+-----+-----+-----+-----+
| numero | data          | nome      | nContribuinte |
+-----+-----+-----+-----+
| 2      | 2007-07-04 23:43:00 | Futaba    | 123456780 |
| 1      | 2007-07-04 23:42:00 | Great Planes | 123456781 |
+-----+-----+-----+-----+
```

2014/09/29 (v06)
180/311

A componente WHERE

- ▶ A cláusula `WHERE` corresponde ao predicado de selecção da álgebra relacional. É formada por um predicado envolvendo atributos de relações que aparecem na cláusula `FROM`.
- ▶ Por exemplo: seleccionar todos os motores do tipo ABC de entre os produtos existentes em armazem:

```
mysql> SELECT * FROM Produtos WHERE nome LIKE '%ABC%' AND quantidade > 0;
```

codProduto	nome	preco	quantidade
1	Tower Hobbies .46 ABC BB	74.99	10
2	Tower Hobbies .61ABC BB	84.99	10
3	Tower Hobbies .75 ABC BB	89.99	5

- ▶ Os resultados de comparações podem ser combinados por intermédio dos conectivos lógicos `AND`, `OR`, e `NOT`.
- ▶ Os operadores relacionais habituais estão disponíveis.
- ▶ A comparação entre sequências de caracteres pode ser feita através de expressões regulares (simples, ou complexas).

2014/09/29 (v06)
181/311

Expressões Regulares

Uma expressão regular sobre um alfabeto finito V é definida de modo indutivo como se segue:

- ▶ ϵ (sequência vazia) é uma expressão regular;
- ▶ a é uma expressão regular, para todo o $a \in V$;
- ▶ se R é uma expressão regular sobre V , então também o é $(R)^*$ (iteração);
- ▶ se Q e R são expressões regulares sobre V , então também $(Q)(R)$ e $(Q)|(R)$ (concatenação e união respectivamente) o são.

2014/09/29 (v06)
183/311

Operações com Sequências de Caracteres

- ▶ O SQL inclui um mecanismo de concordância de padrões para comparações envolvendo cadeias de caracteres. Os padrões são descritos recorrendo a dois caracteres especiais:
 - ▶ percentagem (%): O carácter % concorda com qualquer sub-cadeia.
 - ▶ sublinhado (_): O carácter _ concorda com qualquer carácter.
- ▶ O SQL suporta uma variedade de operações com cadeias de caracteres, tais como:
 - ▶ concatenação (utilizando `||`);
 - ▶ conversão de maiúsculas para minúsculas (e vice-versa);
 - ▶ calcular o comprimento, extracção de subcadeias, etc.
- ▶ O MySQL suporta (versão 5) expressões regulares como mecanismo de comparação de padrões.

2014/09/29 (v06)
182/311

Expressões Regulares

O MySQL usa (versão 5) expressões regulares como mecanismo de comparação de padrões.

```
SELECT <seq_caracteres> REGEXP <expressão_regular>;
```

- ▶ `^` — associa com o início da seq. de caracteres.
- ▶ `$` — associa com o fim da seq. de caracteres.
- ▶ `.` — associa com um qualquer carácter, inclusive a mudança de linha.
- ▶ `a*` — associa com zero ou mais 'a'
- ▶ `a+` — associa com um ou mais 'a'
- ▶ `a?` — associa com zero ou um 'a'
- ▶ `de|abc` — associa com 'de' ou com 'abc'
- ▶ `[a-dX]`, `[^a-dX]` — associa (ou não) com os caracteres 'a' a 'd' e 'X'.

2014/09/29 (v06)
184/311

Expressões Regulares — Exemplos

- ▶ Para achar nomes que começam por 'b'

```
mysql> SELECT * FROM Pet WHERE name REGEXP '\\b';
+-----+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth | death |
+-----+-----+-----+-----+-----+-----+
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
| Bowser | Diane | dog | m | 1989-08-31 | 1995-07-29 |
+-----+-----+-----+-----+-----+-----+
```

- ▶ Para achar nomes contendo um 'w'

```
mysql> SELECT * FROM Pet WHERE name REGEXP 'w';
+-----+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth | death |
+-----+-----+-----+-----+-----+-----+
| Claws | Gwen | cat | m | 1994-03-17 | NULL |
| Bowser | Diane | dog | m | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen | bird | NULL | 1997-12-09 | NULL |
+-----+-----+-----+-----+-----+-----+
```

- ▶ Para achar nomes contendo 'Carlos Seixas'

```
SELECT * FROM Enderecos WHERE rua REGEXP 'Carlos.+Seixas';
```

2014/09/29 (v06)
185/311

Expressões Regulares — Exemplos

No MySQL, os padrões SQL são, por omissão, insensíveis ao tipo do carácter (maiúscula ou minúscula). Se se pretende que a associação seja feita tendo em conta o tipo do carácter, então é necessário usar a palavra chave «Binary».

- ▶ Para achar nomes começando por um 'b' (e não por um 'B')

```
mysql> SELECT * FROM Pet WHERE name REGEXP BINARY '^b';
+-----+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth | death |
+-----+-----+-----+-----+-----+-----+
| Claws | Gwen | cat | m | 1994-03-17 | NULL |
| Bowser | Diane | dog | m | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen | bird | NULL | 1997-12-09 | NULL |
+-----+-----+-----+-----+-----+-----+
```

2014/09/29 (v06)
186/311

Ordenando os tuplos

Listar em ordem alfabética os nomes de todos os fornecedores de motores do tipo «ABC»

```
SELECT Fornecedores.nome
FROM Fornecedores, Produtos
WHERE Produtos.nome LIKE '%ABC%'
ORDER BY Fornecedores.nome
```

- ▶ Pode-se especificar «DESC» para ordenação decendente ou «ASC» para ordenação ascendente, para cada atributo; por omissão, assume-se ordem ascendente.

```
ORDER BY Fornecedores.nome DESC
```

- ▶ Pode-se ter mais do que uma chave de ordenação, separando-as com vírgulas.

```
SELECT * FROM Produtos ORDER BY quantidade, preco
```

2014/09/29 (v06)
187/311

Funções de Agregação

Estas funções aplicam-se a multi-conjuntos de valores de uma coluna de uma relação, devolvendo um valor.

avg	valor médio
min	valor mínimo
max	valor máximo
sum	soma dos valores
count	número de valores

Por exemplo:

- ▶ Determinar o preço médio dos produtos:

```
SELECT AVG(preco) FROM Produtos
```

- ▶ Calcular o número de produtos:

```
SELECT COUNT(*) FROM Produtos
```

- ▶ Encontrar o preço máximo dos produtos:

```
SELECT MAX(preco) FROM Produtos
```

2014/09/29 (v06)
188/311

Funções de agregação - GROUP BY

Podemos agrupar os atributos que pretendemos agregar em diferentes grupos, através do comando `GROUP BY`.

Listar o número de encomendas por fornecedor:

```
SELECT nome, count(*)
FROM Encomendas, Fornecedores
WHERE Encomendas.nContribuinte=Fornecedores.nContribuinte
GROUP BY nome
```

- ▶ Os atributos na cláusula `SELECT` fora de funções de agregação, têm de aparecer na lista `GROUP BY`.
- ▶ Se aparecer mais do que um atributo em `GROUP BY`, então cada grupo é formado pelos tuplo com valores iguais em todos esses atributos.

2014/09/29 (v96)
189/311

Funções de Agregação - HAVING

Podemos limitar os elementos que vão ser agregados através da inclusão de um predicado.

Listar os nomes de todas os fornecedores cujo valor médio dos preços dos seus produtos é superior a 100€.

```
SELECT Fornecedores.nome,AVG(preco)
FROM FornecedorDe, Produtos, Fornecedores
WHERE FornecedorDe.nContribuinte=Fornecedores.nContribuinte
AND FornecedorDe.codProduto=Produtos.codProduto
GROUP BY Fornecedores.nome
HAVING AVG(preco)>100
```

- ▶ Os predicados no comando `HAVING` são aplicados depois da formação dos grupos, enquanto que os predicados no comando `WHERE` são aplicados antes da formação dos grupos.

2014/09/29 (v96)
190/311

Valores Nulos

Os tuplos podem conter valores nulos, denotado por `NULL`, nalguns dos seus atributos.

`NULL` significa um valor desconhecido ou que não existe.

O predicado `IS NULL` pode ser utilizado para testar a existência de valores nulos.

Por exemplo: mostrar todos os fornecedores que possuem FAX.

```
SELECT nome
FROM Fornecedores
WHERE NOT(fax IS NULL)
```

- ▶ O resultado da aplicação de uma operação aritmética ao valor `NULL` é `NULL`. Por exemplo: `5 + NULL` é igual a `NULL`.
- ▶ Qualquer comparação com `NULL` retorna `UNKNOWN`.
- ▶ As funções de agregação, com excepção de `COUNT(*)`, ignoram os nulos.

2014/09/29 (v96)
191/311

Valores Nulos e Lógica Trivalente

Qualquer comparação com `NULL` retorna `UNKNOWN`.

Lógica trivalente usando o valor lógico `UNKNOWN`.

OR	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

AND	TRUE	FALSE	UNKNOWN
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

NOT	UNKNOWN
UNKNOWN	UNKNOWN

- ▶ O predicado `IS UNKNOWN` pode ser usado para testar se o valor de um dado predicado `P` é, ou não, definido. `P IS UNKNOWN` é verdadeiro se o valor de `P` não está definido (é `UNKNOWN`).
- ▶ O resultado da condição do comando `WHERE` é tratado como falso quando o seu valor é `UNKNOWN`.

2014/09/29 (v96)
192/311

Valores Nulos e Agregados

- ▶ Todas as funções de agregação excepto `COUNT(*)` ignoram tuplos com valores nulos nos atributos agregados.
Por exemplo: calcule a média do prazo de entrega dos fornecedores.

```
SELECT AVG(prazo)
FROM Fornecedores
```

- ▶ os valores `NULL` serão ignorados no cálculo da média.
- ▶ o resultado é `NULL` se não existir nenhum montante não-nulo
- ▶ A função de agregação `COUNT` também ignora os valores nulos, por exemplo:

```
SELECT COUNT(prazo)
FROM Fornecedores
```

- ▶ não irá contabilizar os valores nulos.
- ▶ No entanto `COUNT(*)` irá contabilizar todos os tuplos, mesmo aqueles com todos os atributos a `NULL`.

2014/09/29 (v06)
193/311

Operações de Junção

- ▶ As operações de junção retornam uma relação como resultado da combinação de duas outras relações.
- ▶ Estas operações adicionais são utilizadas habitualmente em consultas na componente `FROM`.
- ▶ **Tipo de junção**: define como tratar os tuplos que não estão relacionados entre si (baseados na condição de junção).
- ▶ **Condição de junção**: define quais os tuplos que são combinados nas duas relações, assim como quais os atributos que aparecem no resultado da junção.

Tipos de Junção
inner join
left outer join
right outer join
full outer join

Condições de Junção
natural
on <predicado>
using (A_1, A_2, \dots, A_n)

2014/09/29 (v06)
194/311

Operação de Junção Natural

- ▶ Sejam r e s relações nos esquemas R e S respectivamente. O resultado é uma relação no esquema $R \cup S$ que é obtido considerando cada par de tuplos t_r de r e t_s de s .
- ▶ Se t_r e t_s têm o mesmo valor em cada um dos atributos em $R \cap S$, um tuplo t é adicionado ao resultado, em que:
 - ▶ t tem os mesmos valores que t_r em r .
 - ▶ t tem os mesmos valores que t_s em s
- ▶ Exemplo:

$R = (A, B, C, D)$

$S = (E, B, D)$

Esquema resultado: (A, B, C, D, E)

em que as colunas comuns B e D foram usadas para seleccionar os tuplos a incluir no resultado.

2014/09/29 (v06)
195/311

Junção Interna (ou interior)

- ▶ As junções internas devem ser usadas conjuntamente com uma expressão condicional através dos comandos `ON` e `USING`.
- ▶ A junção interna (`INNER JOIN`) e um produto cartesiano das relações envolvidas considerando somente os tuplos que verificam a expressão condicional especificada.
- ▶ Exemplos:

```
SELECT * FROM table1, table2 WHERE table1.id=table2.id
SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id
```

- ▶ O condicional `ON` é uma qualquer expressão condicional tais como as que são usadas na componente `WHERE`. Deve-se usar a componente `ON` para especificar a forma de junção das tabelas, enquanto a componente `WHERE` deve ser usada para restringir os tuplos que vão ser seleccionados.

2014/09/29 (v06)
196/311

Junção Externa (ou exterior)

- ▶ Uma extensão da operação de junção que evita a perda de informação.
- ▶ Calcula a junção e depois adiciona ao resultado os tuplos de uma relação que não estão relacionados com a outra relação na junção.
- ▶ Numa junção `LEFT OUTER JOIN`, se não existe na tabela da direita uma coluna que associe através da condição `ON`, ou `USING` então todos os valores correspondentes no resultado da consulta (referente à tabela da direita) serão preenchidos com o valor `NULL`.

2014/09/29 (v96)
197/311

Junções — Relações de Exemplo

- ▶ Relação `Emprestimo`

numero	balcaoNome	quantia
L-170	Central	3000
L-230	Pólo I	4000
L-260	Pólo II	1700

- ▶ Relação `ClienteEmp`

nome	numero
João	L-170
Paulo	L-230
Raquel	L-155

2014/09/29 (v96)
198/311

Junções — Exemplos I

```
SELECT *
  FROM Emprestimo
 INNER JOIN ClienteEmp
   ON Emprestimo.numero = ClienteEmp.numero
```

numero	balcaoNome	quantia	nome	numero
L-170	Central	3000	João	L-170
L-230	Pólo I	4000	Paulo	L-230

```
SELECT *
  FROM Emprestimo
 LEFT OUTER JOIN ClienteEmp
   ON Emprestimo.numero = ClienteEmp.numero
```

numero	balcaoNome	quantia	nome	numero
L-170	Central	3000	João	L-170
L-230	Pólo I	4000	Paulo	L-230
L-260	Pólo II	1700	NULL	NULL

2014/09/29 (v96)
199/311

Junções — Exemplos II

```
SELECT *
  FROM Emprestimo
 NATURAL INNER JOIN ClienteEmp
```

numero	balcaoNome	quantia	nome
L-170	Central	3000	João
L-230	Pólo I	4000	Paulo

- ▶ De forma equivalente podíamos escrever

```
SELECT *
  FROM Emprestimo
 INNER JOIN ClienteEmp
   USING (numero)
```

neste caso explicita-se o atributo sobre o qual recai a junção.

A junção natural (`NATURAL [LEFT] JOIN`) de duas tabelas é semanticamente equivalente a uma junção interna (`INNER JOIN` ou `LEFT JOIN`) com uma cláusula `USING` que considera todas as colunas comuns a ambas as tabelas.

2014/09/29 (v96)
200/311

Junções — Exemplos III

```
SELECT *
FROM Emprestimo
NATURAL RIGHT OUTER JOIN ClienteEmp
```

numero	balcaoNome	quantia	nome
L-170	Central	3000	João
L-230	Pólo I	4000	Paulo
L-155	NULL	NULL	Raquel

```
SELECT *
FROM Emprestimo
FULL OUTER JOIN ClienteEmp
USING (numero)
```

numero	balcaoNome	quantia	nome
L-170	Central	3000	João
L-230	Pólo I	4000	Paulo
L-260	Pólo II	1700	NULL
L-155	NULL	NULL	Raquel

2014/09/29 (v06)
201/311

Sub-consultas Embutidas

A linguagem SQL permite a inclusão de sub-consultas na cláusula FROM de uma consulta.

Por exemplo: achar a média dos balanços dos balcões cuja média dos balanços é maior do que 12000€.

```
SELECT balcaoNome, mediaBalanco
FROM (SELECT balcaoNome, avg(balanco)
FROM Conta
GROUP BY balcaoNome) AS MediaBalcao (balcaoNome, mediaBalanco)
WHERE mediaBalanco > 12000
```

Deixa de ser necessário a cláusula HAVING dado que se calcula a sub-consulta (vista temporária) e os atributos da mesma podem ser usados na consulta principal.

2014/09/29 (v06)
202/311

Relações Derivadas

A cláusula WITH permite a definição de uma sub-consulta (vista temporária) cujo alcance é somente o da consulta aonde a mesma ocorre.

Exemplo: achar todas as contas cujo balanço seja igual ao valor máximo achado.

```
WITH BalancoMax(valor)
AS SELECT max(balanco)
FROM Conta
SELECT numero
FROM Conta, BalancoMax
WHERE Conta.balanco = BalancoMax.valor
```

A cláusula WITH é o correspondente à instrução de atribuição na álgebra relacional. No entanto é de notar que a mesma não pode ser usada sem estar associada a uma consulta.

2014/09/29 (v06)
203/311

Relações Derivadas

Outro exemplo: Achar todos os balcões aonde o valor total dos depósitos é maior do que a média dos valores dos depósitos em todos os balcões.

```
WITH TotalBalcao(balcaoNome, valor)
AS SELECT balcaoNome, sum(balanco)
FROM Conta
GROUP BY balcaoNome
WITH MediaTotalBalcao(valor)
AS SELECT avg(valor)
FROM TotalBalcao
SELECT balcaoNome
FROM TotalBalcao, MediaTotalBalcao
WHERE TotalBalcao.valor > MediaTotalBalcao.valor
```

Notar que após a definição da relação derivada a mesma pode ser usada de imediato.

2014/09/29 (v06)
204/311

Vistas

- ▶ Em certas circunstâncias, não é desejável que todos os utilizadores possam aceder a todo o modelo lógico (i.e. a todas as relações armazenadas na base de dados)
- ▶ Considere o caso de um empregado que necessita de saber o número de empréstimo de um cliente, mas que não precisa de saber o montante desse empréstimo. Este empregado deveria ver apenas a relação

```
(SELECT nome, numero
  FROM ClienteEmp, Empréstimo
 WHERE ClienteEmp.numero = Empréstimo.numero)
```

- ▶ Uma **vista** providencia um mecanismo de sonegação de informação.
- ▶ Qualquer relação que não pertença ao modelo conceptual mas que se torne visível ao utilizador como uma «relação virtual» é designada por **vista**.

2014/09/29 (v06)
205/311

Exemplos de Vistas

- ▶ Considere-se a vista (com o nome todosOsClientes) contendo os nomes das agências e seus clientes.

```
CREATE VIEW TodosOsClientes
  AS (SELECT balcaoNome, nome
      FROM ClienteEmp, Conta
      WHERE ClienteEmp.numero = Conta.numero)
  UNION
  (SELECT balcaoNome, nome
      FROM ClienteEmp, Empréstimo
      WHERE ClienteEmp.numero = Empréstimo.numero)
```

- ▶ Pode-se assim encontrar todos os clientes da agência de Penacova através de:

```
SELECT nome
  FROM TodosOsClientes
 WHERE balcaoNome = 'Penacova'
```

2014/09/29 (v06)
207/311

Definição de Vistas

- ▶ Uma vista é definida por intermédio da instrução **CREATE VIEW**

```
CREATE VIEW NomeVista AS <consulta>
```

Em que <consulta> é uma consulta SQL qualquer. O nome da vista é NomeVista.

- ▶ Após a definição de uma vista, o seu nome pode ser utilizado para se referir à relação virtual gerada pela vista.
- ▶ Uma definição de uma vista não é o mesmo que criar uma nova relação a partir da avaliação da sua expressão. Em vez disso, a definição da vista permite guardar a expressão que depois é substituída nas consultas que utilizam a vista.

2014/09/29 (v06)
206/311

Vistas Definidas a Partir de Outras Vistas

- ▶ Uma vista pode ser utilizada na expressão de definição de outra vista.
- ▶ Uma vista v_1 **depende directamente** de uma vista v_2 se v_2 é utilizada na expressão que define v_1 .
- ▶ Uma vista v_1 **depende** de uma vista v_2 se v_1 depende directamente de v_2 ou se existe um caminho de dependências entre v_1 e v_2 .

2014/09/29 (v06)
208/311

Expansão de Vistas

- ▶ Forma de atribuir significado a vistas definidas em termos de outras vistas.
- ▶ Seja a vista v_i definida em termos de uma expressão e_1 que pode ela própria conter vistas.
- ▶ Para expandir as vistas numa expressão repete-se sucessivamente o seguinte passo:

Repetir

Encontrar uma vista v_i em e_1

Substituir a vista v_i pela expressão que a define

Até que não ocorram mais vistas em e_1

2014/09/29 (v06)
209/311

Modificação da Base de Dados - Inserção

A inserção de tuplos numa tabela é feita em SQL com a instrução INSERT.

```
INSERT INTO <tabela> [(<lista_de_atributos>)]  
VALUES (<lista_de_expressões>)
```

No caso de não se especificar a lista_de_atributos subentende-se que são todos os atributos.

Exemplos:

- ▶ Adicionar um novo tuplo a Produtos

```
INSERT INTO Produtos  
VALUES (8, 'F4U_Corsair', 403.57, 5)
```

ou equivalentemente

```
INSERT INTO Produtos (nome, preco, quantidade)  
VALUES ('F4U_Corsair', 403.57, 5)
```

Para todos os atributos da tabela não especificados são usados os valores por omissão (no caso do atributo `codProduto` é auto-incremental).

2014/09/29 (v06)
210/311

Modificação da Base de Dados - Inserção (cont.)

É também possível usar uma sintaxe alternativa.

```
INSERT INTO <tabela>  
SET <atributo1> = <expressão1>,  
    <atributo2> = <expressão2>,  
    ...  
    <atributoN> = <expressãoN>
```

Exemplo:

```
INSERT INTO Produtos  
SET nome='P-47D_Thunderbolt',  
    preco=403.57,  
    quantidade=3
```

2014/09/29 (v06)
211/311

Modificação da base de dados - Inserção (cont.)

Finalmente, é possível copiar valores entre tabelas através da instrução INSERT ... SELECT

Exemplo: dar como bônus a todos os clientes com empréstimos na agência Central de Coimbra da CGD, uma conta poupança de 2000€. O número do empréstimo servirá de número de conta poupança.

```
INSERT INTO Conta  
SELECT numero, 2000, balcaoNome  
FROM Emprestimo  
WHERE balcaoNome = 'Coimbra-central'
```

A instrução SELECT ... FROM ... WHERE é avaliada previamente à inserção de tuplos na relação (caso contrário consultas como INSERT INTO tabela1 SELECT * FROM tabela1 causariam problemas)

2014/09/29 (v06)
212/311

Modificação da base de dados - Actualização

A actualização de tuplos numa tabela é feita em SQL com a instrução UPDATE

```
UPDATE <tabela>
  SET <atributo1> = <expressão1>,
    ...
    <atributoN> = <expressãoN>
[ WHERE <condição> ]
```

Exemplos:

- ▶ Actualizar a quantidade em armazém de um produto após a recepção de uma encomenda:

```
UPDATE Produtos
  SET quantidade = quantidade+2
  WHERE codProduto=6;
```

- ▶ Actualizar os preços de acordo com uma taxa de inflação de 2%

```
UPDATE Produtos
  SET preco=preco*1.02;
```

2014/09/29 (v06)
213/311

Modificação da base de Dados - Remoção

A remoção de tuplos de uma tabela é feita em SQL com a instrução DELETE.

```
DELETE
  FROM <tabela>
[ WHERE <condição> ]
```

Por exemplo: retirar da tabela Produtos todos os produtos dos quais não haja nenhum exemplar em armazém

```
DELETE
  FROM Produtos
  WHERE quantidade=0
```

Nota: esta instrução vai falhar (na actual implementação da base de dados) à conta de uma restrição de integridade (FornecedorDe).

2014/09/29 (v06)
214/311

Mudanças Através de Vistas

É possível usar uma **Vista** para actualizar uma tabela, utilizando as instruções UPDATE, DELETE, ou INSERT.

Para que seja possível actualizar uma tabela com uma **Vista** é necessário que haja uma relação injectiva (um-para-um) entre as linhas da **Vista** e as linhas da tabela que ela referênciava.

Além das restrições eventualmente impostas pelas permissões de acesso às tabelas subjacentes existem outras restrições que podem impedir a actualização de valores através de **Vistas**.

2014/09/29 (v06)
215/311

Mudanças Através de Vistas (cont.)

Para poder actualizar as tabelas subjacentes a **Vista** não pode conter:

- ▶ funções de agregação;
- ▶ DISTINCT; GROUP BY; HAVING; UNION;
- ▶ uma sub-consulta na lista de selecção;
- ▶ alguns tipos de junções;
- ▶ uma **Vista** não actualizável na cláusula FROM;
- ▶ múltiplas referências a uma dada coluna na tabela base.

Quanto à inserção de valores verifica-se ainda que:

- ▶ não pode haver duplicação nos nomes das colunas;
- ▶ a **Vista** tem de conter todas as colunas de preenchimento obrigatório e em que não esteja definido um valor por omissão;
- ▶ as colunas da **Vista** não pode ser uma coluna derivada de outras colunas.

2014/09/29 (v06)
216/311

Operação de União em SQL

Relações R_1 e R_2 :

$$R_1 = \begin{array}{|c|c|} \hline \text{col1} & \text{col2} \\ \hline 1 & a \\ \hline 2 & b \\ \hline 3 & c \\ \hline 4 & d \\ \hline \end{array}$$

$$R_2 = \begin{array}{|c|c|} \hline \text{col1} & \text{col2} \\ \hline 1 & a \\ \hline 3 & c \\ \hline 5 & e \\ \hline \end{array}$$

```
SELECT * from R1
UNION
SELECT * from R2
```

$$R_1 \cup R_2 = \begin{array}{|c|c|} \hline \text{col1} & \text{col2} \\ \hline 1 & a \\ \hline 2 & b \\ \hline 3 & c \\ \hline 4 & d \\ \hline 5 & e \\ \hline \end{array}$$

2014/09/29 (v06)
217/311

Operação de Diferença em SQL

O MySQL não suporta directamente a operação de diferença.

$$R_1 - R_2 = \begin{array}{|c|c|} \hline \text{col1} & \text{col2} \\ \hline 2 & b \\ \hline 4 & d \\ \hline \end{array}$$

- ▶

```
SELECT * FROM R1
WHERE (col1,col2)
NOT IN (SELECT *
FROM R2);
```
- ▶

```
SELECT * FROM R1
WHERE NOT EXISTS (SELECT * FROM R2
WHERE R1.col1=R2.col1
AND R1.col2=R2.col2);
```
- ▶ O modo mais eficiente:

```
SELECT DISTINCT *
FROM R1
LEFT OUTER JOIN R2 USING (col1,col2)
WHERE R2.col1 IS NULL
```

2014/09/29 (v06)
218/311

Operação de Intersecção em SQL

O MySQL não suporta directamente a operação de intersecção.

$$R_1 \cap R_2 = \begin{array}{|c|c|} \hline \text{col1} & \text{col2} \\ \hline 1 & a \\ \hline 3 & c \\ \hline \end{array}$$

- ▶

```
SELECT * FROM R1
WHERE (col1,col2)
IN (SELECT *
FROM R2);
```
- ▶

```
SELECT * FROM R1
WHERE EXISTS (SELECT * FROM R2
WHERE R1.col1=R2.col1
AND R1.col2=R2.col2);
```
- ▶ O modo mais eficiente:

```
SELECT DISTINCT *
FROM R1
INNER JOIN R2 USING (col1,col2)
```

2014/09/29 (v06)
219/311

Operação de Divisão em SQL

- ▶ Propriedade
 - ▶ Seja $q = r \div s$
 - ▶ Então q é a maior relação satisfazendo $q \times s \subseteq r$.
- ▶ Definição em termos de operações básicas da álgebra rel.
 Sejam $r(R)$ e $s(S)$ relações, com $S \subset R$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

Porquê?

- ▶ $\Pi_{R-S}(r) \times s$ dá os elementos de r com todos os valores de S .
- ▶ $\Pi_{R-S,S}(r)$ construi uma versão de r com os atributos da expressão anterior.
- ▶ $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)$ dá os tuplos t em $\Pi_{R-S}(r)$ tal que para algum tuplo $u \in s$, $tu \notin r$.

2014/09/29 (v06)
220/311

Operação de Divisão em SQL

Adequada para consultas que incluam a frase "para todo".

$$\begin{aligned} r \div s &= \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)) \\ &= \Pi_{(A,B,C)}(r) - \Pi_{(A,B,C)}((\Pi_{(A,B,C)}(r) \times s) - \Pi_{(A,B,C,D,E)}(r)) \end{aligned}$$

$$R = (A, B, C, D, E) \quad S = (D, E) \quad R - S = (A, B, C)$$

- ▶ $\Pi_{(A,B,C)}(r) \times s$ dá os elementos de r com todos os valores de (D, E) .
- ▶ $\Pi_{(A,B,C,D,E)}(r)$ constrói uma versão de r com os atributos da expressão anterior. Neste caso: $\Pi_{(A,B,C,D,E)}(r) = r$.
- ▶ $\Pi_{(A,B,C)}((\Pi_{(A,B,C)}(r) \times s) - \Pi_{(A,B,C,D,E)}(r))$ dá os tuplos t em $\Pi_{(A,B,C)}(r)$ tal que para algum tuplo $u \in s$, $tu \notin r$.

2014/09/29 (v06)
221 / 311

MySQL — Exemplos de Consultas

Uma aproximação às consultas do tipo \div (do manual do MySQL)

- ▶ Que tipos de lojas estão presentes em uma, ou mais, cidades?

```
SELECT DISTINCT tiposLojas
FROM Lojas
WHERE EXISTS
  (SELECT *
   FROM CidadesLojas
   WHERE CidadesLojas.tipoLojas = Lojas.tiposLojas)
AS Aux;
```

- ▶ Que tipos de lojas não estão presentes em nenhuma cidade?

```
SELECT DISTINCT tipoLojas
FROM Lojas
WHERE NOT EXISTS
  (SELECT *
   FROM CidadesLojas
   WHERE CidadesLojas.tipoLojas = Lojas.tipoLojas)
AS Aux;
```

2014/09/29 (v06)
222 / 311

MySQL — Exemplos de Consultas

- ▶ Que tipos de lojas estão presentes em todas as cidades?

```
SELECT DISTINCT tipoLojas FROM Lojas
WHERE NOT EXISTS
  (SELECT * FROM Cidades
   WHERE NOT EXISTS
     (SELECT * FROM CidadesLojas
      WHERE CidadesLojas.cidade = Cidades.cidade
      AND CidadesLojas.tipoLojas = Lojas.tipoLojas)
   AS Aux1)
AS Aux2;
```

Este último exame é uma consulta «NOT EXISTS» duplamente embutida. Isto é, contém uma cláusula «NOT EXISTS» dentro de uma outra cláusula do mesmo tipo. Formalmente, responde à questão «não existe uma cidade com uma loja que não está em Lojas?». É mais fácil de ver que uma consulta «NOT EXISTS» duplamente embutida responde à questão «é x verdade para todo o y?»

2014/09/29 (v06)
223 / 311

SQL Embutido

- ▶ SQL fornece uma linguagem declarativa para manipulação de bases de dados.
- ▶ Facilita a manipulação e permite optimizações muito difíceis se fossem programadas em linguagens imperativas.
- ▶ Mas há razões para usar SQL juntamente com linguagens de programação gerais (imperativas):
 - ▶ o SQL não tem a expressividade de uma máquina de Turing, há perguntas impossíveis de codificar em SQL (por exemplo, fechos transitivos); usando SQL juntamente com linguagens gerais é possível suprir esta deficiência;
 - ▶ nem tudo nas aplicações de bases de dados é declarativo (por exemplo, acções de afixar resultados, interfaces, etc). Essa parte pode ser programado em linguagens genéricas.
- ▶ O standard SQL define uma série de “encaixes”, para várias linguagens de programação (e.g. Pascal, PHP, C, C++, Cobol, etc). À linguagem na qual se incluem comandos SQL chama-se linguagem hospedeira. Às estruturas SQL permitidas na linguagem hospedeira chama-se SQL embutido.

2013/10/08 (v06)
224 / 311

SQL Embutido

- ▶ Permite acesso a bases e dados SQL, via outra linguagens de programação.
- ▶ Toda a parte de acesso e manipulação da base de dados é feito através de código embutido. Todo o processamento associado é feito pelo sistema de bases de dados. A linguagem hospedeira recebe os resultados e manipula-os.
- ▶ O código tem que ser pré-processado. A parte SQL é transformada em código da linguagem hospedeira, mais chamadas a “run-time” do servidor.
- ▶ A forma particular como é feita a ligação entre a linguagem hospedeira e a linguagem SQL varia de linguagem para linguagem.

2013/10/08 (v06)
225 / 311

MySQL & PHP

- ▶ `mysql_connect` — Abre ou reutiliza uma conexão com um servidor MySQL.
`resource mysql_connect([string server[, string username[, string password[, bool new_link[, int client_flags]]]])`
Retorna um identificador de ligação MySQL em caso de sucesso, ou “FALSE” em caso de insucesso.
- ▶ `mysql_select_db` — Seleciona uma base de dados MySQL.
`bool mysql_select_db (string database_name[, resource link_identifier])`
Devolve “TRUE” em caso de sucesso ou “FALSE” em caso de insucesso.

```
<?php
$ligacao = mysql_connect('localhost', 'mysql.user', 'mysql.password');
if (!$ligacao) {
    die('Não foi possível fazer a ligação: ' .mysql_error());
}
```

```
echo 'Ligação bem sucedida';
mysql_close($ligacao);
?>
```

2013/10/08 (v06)
227 / 311

MySQL & PHP

Este exemplo simples mostra como, na linguagem PHP, fazer a ligação, executar uma consulta, mostrar as linhas do resultado e desligar do banco de dados MySQL.

Exemplo 1. Exemplo de visão geral da extensão MySQL

```
<?php
// Ligação e escolha da base de dados
$ligacao = mysql_connect('mysql.host', 'mysql.user', 'mysql.password')
    or die('Não foi possível ligar: ' .mysql_error());

echo 'Ligação bem sucedida';

mysql_select_db('my.database') or die('Não foi possível selecionar
a base de dados');
?>
```

A ligação entre o PHP e o MySQL é feita através de funções PHP próprias que estão definidas no módulo (extensão) `php-mysql`:

2013/10/08 (v06)
226 / 311

MySQL & PHP

```
// Executando a consulta SQL
$consulta = 'SELECT *
            FROM my_table';
$resultado = mysql_query($consulta) or die('A consulta falhou!:' .mysql_error());

// Exibindo os resultados em HTML
echo "<table>";
while ($line = mysql_fetch_array($resultado, MYSQL_ASSOC)) {
    echo "<tr>";
    foreach ($line as $col_value) {
        echo "<td> $col_value</td>";
    }
    echo "</tr>";
}
echo "</table>";

// Libertar o conjunto de resultados
mysql_free_result($resultado);

// Fechar a ligação
mysql_close($ligacao);
?>
```

2013/10/08 (v06)
228 / 311