

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:

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

2012/11/07 (v68)
177 / 299

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
```

2012/11/07 (v68)
179 / 299

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 deverá utilizar, por exemplo, `balcaoNome` em vez de `balcao-nome` num sistema existente.
- ▶ NOTA: As maiúsculas e minúsculas não são distinguidas em identificadores da linguagem SQL.

2012/11/07 (v68)
178 / 299

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*1.21 FROM Produtos;
+-----+-----+
| nome          | preco*1.21           |
+-----+-----+
| 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.

2012/11/07 (v68)
180 / 299

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

2012/11/07 (v68)
181 / 299

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.

2012/11/07 (v68)
183 / 299

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).

2012/11/07 (v68)
182 / 299

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) são.

2012/11/07 (v68)
184 / 299

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'.

2012/11/07 (v68)
185 / 299

Expressões Regulares — Exemplos

- ▶ Para achar nomes que começam por 'b'

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
```
- ▶ Para achar nomes contendo um 'w'

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
```
- ▶ Para achar nomes contendo 'Carlos Seixas'

```
mysql> SELECT * FROM endereços WHERE rua REGEXP 'Carlos.+Seixas';
```

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';
```

2012/11/07 (v68)
186 / 299

Ordenando os tuplos

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

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

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

```
ORDER BY Fornecedoros.nome DESC
```

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

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

2012/11/07 (v68)
187 / 299

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(preço) FROM Produtos
```
- ▶ Calcular o número de produtos:

```
SELECT COUNT(*) FROM Produtos
```
- ▶ Encontrar o preço máximo dos produtos:

```
SELECT MAX(preço) FROM Produtos
```

2012/11/07 (v68)
188 / 299

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.

2012/11/07 (v68)
189 / 299

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.

2012/11/07 (v68)
190 / 299

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.

2012/11/07 (v68)
191 / 299

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.

2012/11/07 (v68)
192 / 299

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`.

2012/11/07 (v68)
193 / 299

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).

2012/11/07 (v68)
194 / 299

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
```

2012/11/07 (v68)
195 / 299

Modificação da base de dados - Inserção (cont.)

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

Exemplo: dar como bónus 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)

2012/11/07 (v68)
196 / 299

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;
```

2012/11/07 (v68)
197 / 299

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).

2012/11/07 (v68)
198 / 299

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)

2012/11/07 (v68)
199 / 299

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

2012/11/07 (v68)
200 / 299

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.
- ▶ Numa junção `LEFT 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`.

2012/11/07 (v68)
201 / 299

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.

2012/11/07 (v68)
203 / 299

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

2012/11/07 (v68)
202 / 299

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.

2012/11/07 (v68)
204 / 299

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.
- ▶ Utiliza valores nulos :
 - ▶ NULL significa que o valor é desconhecido ou que não existe.
 - ▶ Simplificadamente, todas as comparações com NULL são falsas por definição.

2012/11/07 (v68)
205 / 299

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

2012/11/07 (v68)
206 / 299

Operação de União em SQL

Relações R_1 e R_2 :

$$R_1 =$$

col1	col2
1	a
2	b
3	c
4	d

$$R_2 =$$

col1	col2
1	a
3	c
5	e

```

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

$$R_1 \cup R_2 =$$

col1	col2
1	a
2	b
3	c
4	d
5	e

2012/11/07 (v68)
207 / 299

Operação de Diferença em SQL

O MySQL não suporta directamente a operação de diferença.

$$R_1 - R_2 =$$

col1	col2
2	b
4	d

- ▶

```

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
    
```

2012/11/07 (v68)
208 / 299

Operação de Intersecção em SQL

O MySQL não suporta directamente a operação de intersecção.

$$R_1 \cap R_2 =$$

col1	col2
1	a
3	c

- ▶

```
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)
```

2012/11/07 (v68)
209 / 299

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 $u \in s$, $tu \notin r$.

2012/11/07 (v68)
210 / 299

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) .
- ▶ $\Pi_{(A,B,C,D,E)}(r)$ construi 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 $u \in s$, $tu \notin r$.

2012/11/07 (v68)
211 / 299

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 tipos_lojas FROM Lojas
WHERE EXISTS
(SELECT * FROM cidades_lojas
WHERE cidades_lojas.tipo_loja = lojas.tipos_lojas)
AS Aux;
```

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

```
SELECT DISTINCT tipo_loja FROM lojas
WHERE NOT EXISTS
(SELECT * FROM cidades_lojas
WHERE cidades_lojas.tipo_lojas = lojas.tipo_lojas)
AS Aux;
```

2012/11/07 (v68)
212 / 299

MySQL — Exemplos de Consultas

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

```
SELECT DISTINCT store_type FROM stores s1
WHERE NOT EXISTS
  (SELECT * FROM cities
   WHERE NOT EXISTS
     (SELECT * FROM cities_stores
      WHERE cities_stores.city = cities.city
       AND cities_stores.store_type = stores.store_type)
  AS Aux1)
AS Aux2;
```

Este último exame é uma consulta “NOT EXISTS” duplamente aninhada. 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 aninhada responde à questão “é x verdade para todo o y?”

2012/11/07 (v68)
213 / 299

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.

2012/09/17 (v62)
215 / 299

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.

2012/09/17 (v62)
214 / 299

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 seleccionar
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:

2012/09/17 (v62)
216 / 299

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);  
?>
```

2012/09/17 (v62)
217 / 299

MySQL & PHP

- ▶ `mysql_query` — Realiza uma consulta MySQL

```
resource mysql_query(string query[,resource link_identifier])
```

`mysql_query()` envia uma consulta para a base de dados activa no servidor da ligação presente `link_identifier`.

Nota: A string da query **não** deve terminar com ponto e virgula(;).

Para os comandos `SELECT`, `SHOW`, `EXPLAIN` ou `DESCRIBE` o valor de retorno é identificador de recurso ou “FALSE” se a consulta não foi executada correctamente. Para outros tipos de comandos SQL, o valor de retorno é “TRUE” em caso de sucesso e “FALSE” em erro.

O identificador de recurso pode depois ser usado em funções que lidam com os resultados das consultas (tabelas).

Com `mysql_num_rows()` e com `mysql_affected_rows()` podem-se obter quantas linhas foram devolvidas, ou quantas linhas foram afectadas, respectivamente.

`mysql_free_result()` liberta (explicitamente) os recursos.

2012/09/17 (v62)
219 / 299

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);  
?>
```

2012/09/17 (v62)
218 / 299

MySQL & PHP

- ▶ `mysql_fetch_array` — Obtém uma linha como uma matriz associativa, uma matriz numérica, ou ambas.

```
array mysql_fetch_array (resource resultado [, int result.type] )
```

Retorna uma matriz que corresponde a linha obtida **e move o ponteiro interno dos dados adiante**.

Parâmetros: resultado (o resultado obtido pela consulta)

O tipo de vector (“array”) que é obtido é uma constante e pode ter os seguintes valores: `MYSQL_ASSOC`, `MYSQL_NUM`, e o valor por omissão de `MYSQL_BOTH`.

Valores de retorno: uma vector que corresponde à linha obtida, ou `FALSE` se não houver mais linhas. O tipo do vector devolvido depende de como `result.type` esta definido.

Usando `MYSQL_BOTH` (por omissão), você terá um vector com ambos os índices, numérico e associativo. Usando `MYSQL_ASSOC`, você tem apenas os índices associativos, usando `MYSQL_NUM`, você tem apenas os índices numéricos.

2012/09/17 (v62)
220 / 299