Fundamentos de Base de Dados

SQL – Parte II

Pedro Nogueira Ramos

(Pedro.Ramos@iscte.pt)

ISTA / DCTI

UPDATE

Um comando UPDATE para alteração de linhas obedece à seguinte estrutura:

UPDATE *tabela a alterar*

SET *coluna a alterar = expressão*

WHERE *expressão lógica que indica quais as linhas que pretendemos alterar*

O seguinte comando transforma os códigos postais 1200 em 1500

```
UPDATE Cliente SET CodPostal = 1500
WHERE CodPostal = 1200;
```

DELETE

Um comando DELETE para anulação de linhas obedece à seguinte estrutura:

DELETE FROM *tabela a anular*

WHERE *expressão lógica que indica quais as linhas que pretendemos alterar*

O seguinte comando apaga os códigos postais 1200

```
DELETE FROM Cliente
WHERE CodPostal = 1200;
```

INSERT – Uma Linha

Através do comando INSERT podem-se inserir uma linha ou várias linhas em simultâneo. Para inserir uma linha um comando INSERT obedece à seguinte estrutura:

INSERT INTO *tabela a inserir* (*colunas onde vão ser inseridos os valores*)

VALUES (*valores a inserir*)

```
INSERT INTO Produto (cod_produto, tipo) VALUES (123456, 'MP');
```

INSERT – várias linhas

Para inserir um conjunto de linhas, um comando INSERT obedece à seguinte estrutura:

INSERT INTO *tabela a inserir* (*colunas onde vão ser inseridos os valores*)

SELECT *valores a inserir*

FROM ...

```
INSERT INTO Produto (cod_produto, tipo)
SELECT cod_materia, 'MP' FROM Materia_Prima;
```

Views

As Views não são mais do que comandos SELECT *armazenados*. São por vezes denominadas tabelas temporárias. Note-se que o resultado de uma execução de uma view (os registos que ela *devolve)* depende dos registos armazenados no momento nas tabelas de suporte à view. As views podem ser utilizadas dentro de comandos SELECT.

```
CREATE VIEW Clientes_Lisboa (BI, Nome)
AS Select BI, Nome FROM Cliente KEY JOIN
Localidade
 Where Localidade = 'Lisboa';


Select Nome From Clientes_Lisboa;
```

Views

As views não podem conter a cláusula ORDER BY e apenas permitem a inserção, remoção e alteração de registos caso não contenham as cláusulas GROUP BY e UNION.

A cláusula CHECK OPTION rejeita alterações e inserções na view que não obedeçam ao critério da cláusula SELECT que a define.

```
CREATE VIEW Clientes_Lisboa (BI, Nome)
AS Select BI, Nome FROM Cliente KEY JOIN
Localidade
  Where Localidade = 'Lisboa'
WITH CHECK OPTION;
```

Views

BI	Nome	Localidade
1	Ana	Lisboa
4	João	Porto
23	Luísa	Évora
40	Pedro	NULL
7	André	Lisboa

```
CREATE VIEW Clientes_Lisboa (BI, Nome)
AS Select BI, Nome FROM Cliente KEY JOIN
Localidade
  Where Localidade = 'Lisboa'
WITH CHECK OPTION;

Select Nome From Clientes_Lisboa;
```

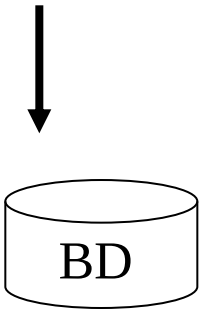
BI	Nome
1	Ana
7	André

Nome
Ana
André

Prepare Statement

O comando Prepare Statement permite optimizar acessos à base de dados que são efectuados múltiplas vezes em uma transacção.

Java Programa
sql_var String
sql_i int
while i<100 {
sqlvar= " update cliente set .... Where id = ***i_var****"*
ExecuteSql(sqlvar);
i++;
}



100 vezes (!!!)

1 Parsing (interpretar)

2 Planear (usar índice?, busca sequencial?)

3 Gerar código (c);

4 Executar;

Com Prepare Statement apenas fazemos uma vez as tarefas 1,2 e 3.

Prepare Statement

O comando apenas se justifica quando o SQL é utilizado dentro de outra linguagem de programação (Java, C, Visual Basic, etc.) ou em Stored Procedures (ver mais adiante).

Exemplo (simplificado)

```
PREPARE calc_factura("Update Factura Set Valor = ?  Where Num_Factura = ? ;");
calc_factura.setDouble(1, totalFactura);
calc_factura.setInt(2, lastFactura);
calc_factura.executeSQL()
```

Declaração de parâmetros

Execução

Passagem de parâmetros

Stored Procedure e Funções

Os Stored Procedures (SP) e Funções são procedimentos/funções SQL compilados e armazenados junto da base de dados. Trata-se da forma mais eficiente de executar comandos SQL. Para além dos comandos SQL é possível utilizar as habituais primitivas de controlo (If, While), os habituais operadores lógicos e variáveis. Trata-se no entanto de uma linguagem simples que não substitui as linguagens procedimentais (C, Java, etc.).

É possível passar parâmetros para um SP/Função assim como uma Função retornar valores (nomeadamente o resultado de comandos SELECT). Os SP7Funções podem ser chamados dentro de um comando SELECT ou a partir de uma linguagem procedimental (através da primitiva CALL).

Stored Procedures - Exemplos

```
create procedure dba.newPubdelay()
begin
  declare lastISBN integer;
  select max(ISBN) into lastISBN from Pub;
  insert into pub(isbn) values(lastISBN+1);
  commit work
end
```

```
create procedure dba.getPubAno(in Par_Ano integer)
result(ISBN integer,Titulo long varchar)
begin
   select ISBN,Titulo from Pub where DAta = Par_Ano
end
```

→ O comando `CALL getPubAno(2000);` retorna as publicações de 2000

Funções - Exemplos

```
CREATE function DBA.totalReserva(in numRes integer)
returns numeric
begin
  declare total numeric;
  select sum(valor) into total from
       Reserva_quarto as rq,Quarto as q where
    rq.numero = q.numero and rq.sigla = q.sigla and
    rq.numeroReserva = numRes;
  return(total)
end
```

O comando `SELECT "totalReserva"(10)` retorna o valor da reserva número 10

Triggers (I)

Os *triggers* são procedimentos armazenados junto da base de dados que são associados a eventos que ocorrem nas tabelas. Através dos *triggers* o motor de base de dados reage automaticamente quando esses eventos (alterações dos dados nas tabelas) ocorrem. A forma como reagem é definida pelo procedimento associado ao *trigger* (um *trigger* apenas pode estar associado a um evento de uma tabela). Os eventos aos quais podem se associar os *triggers*:

Evento	Before	After
Insert	Antes de inserido o registo....	Depois de o registo ser inserido
Delete	Antes ... anulado	Depois ...anulado
Update	Antes de pelo menos um campo ser alterado	Depois
Update Of	Antes de um campo específico ser alterado	Depois...

Os eventos *Insert*, *Update* e *Delete* ocorrem ao nível do registo completo, enquanto que o evento *Update Of* ocorre ao nível de um campo da tabela.

Triggers (II)

Os *triggers* podem ainda ser ***row-level*** ou ***statement-level***. Os triggers *row-level* ocorrem após cada linha da tabela ser alterada (anulada ou inserida). Os triggers *statement-level* ocorrem após uma operação sobre a tabela ser concluída (por exemplo, após um conjunto de registos ser inserido).

Triggers – Exemplo row level

Produto

Cod_produto	Nome	tipo	....
1	Peça A	PA	
2	Matéria XZ	MP	
3	Peça F	PA	
4	Matéria GF	MP	

```
create trigger "dba".MateriaPrima
after insert  on Produto
referencing new as new_produto
for each row
when(new_produto.Tipo='MP')
Begin

insert into
  MateriaPrima(cod_produto)
  values(new_produto.cod_produto)
end
```

Matéria Prima

Cod_produto	-----
2	
4	