Bases de Dados

Universidade do Porto

Faculdade de Engenharia

FEUP

- INTRODUÇÃO
- MODELOS CONCEPTUAIS
  - Diagrama de Classes UML
  - Modelo Entidade-Associação (E-A)
- MODELO RELACIONAL
- LINGUAGEM DE DEFINIÇÃO DE DADOS
- INTERROGAÇÃO DE DADOS
  - Álgebra relacional
  - **LINGUAGEM DE MANIPULAÇÃO DE DADOS (LMD) SQL**

Observação: baseado em slides desenvolvidos pelo Prof. Gabriel David (FEUP)

Índice

- *Persistent Stored Modules (PSM)*
- SQL embebido

PSM

SQL na prática

- Até agora viu-se como se usa SQL através de uma interface própria para a execução de consultas à base de dados.
- A realidade costuma ser diferente: programas convencionais interagem com a(s) base(s) de dados através de SQL.

PSM

Opções

1. Código escrito em linguagens especializadas é guardado na base de dados (ex:, PL/SQL do SGDB Oracle, SQL PL do DB2, PL/pgPSM do PostgreSQL, ...).
2. Instruções SQL são embebidas numa *linguagem hospedeira* (ex: C).
3. Ferramentas de ligação são utilizadas de forma a permitir a uma linguagem conventional aceder a uma base de dados (ex: CLI, JDBC, PHP/DB).

PSM

Procedimentos guardados

- PSM, ou "*persistent stored modules*," permitem-nos guardar procedimentos como elementos da estrutura da base de dados.
- PSM = mistura de instruções convencionais (if, while, etc.) e SQL.
- Permitem-nos fazer coisas que não se conseguem fazer só com SQL ou fazê-las de forma mais eficiente.

PSM

Estrutura básica de PSMs

CREATE PROCEDURE <nome> (
<lista de parâmetros>)
<declarações locais opcionais>
<corpo>;

- Função alternativa:

CREATE FUNCTION <nome> (
<lista de parâmetros>) RETURNS <tipo>

PSM

Passagem de parâmetros em PSMs

- Ao contrário dos pares nome-tipo de linguagens como C, PSM usa tuplos modo-nome-tipo, onde o *modo* pode ser:
  - IN = o procedimento usa valores, não os altera.
  - OUT = o procedimento altera, não os usa.
  - INOUT = ambos.

PSM

Exemplo: procedimento guardado

- Vamos escrever um procedimento com dois argumentos *c* e *p*, que adiciona um registo a Vende(bar, cerveja, preco) com bar = 'Pipa Velha', cerveja = *c*, e preco = *p*.
  - Utilizado pelo dono do bar Pipa Velha para adicionar este menu mais facilmente.

PSM

O procedimento

```
CREATE PROCEDURE MenuPipaVelha (
    IN  c     CHAR(20),
    IN  p     REAL
)
INSERT INTO Vende
VALUES('Pipa Velha', c, p);
```

Ambos os parâmetros são só de leitura, não são modificados.

O corpo do procedimento: apenas um INSERT

PSM

Invocação de procedimentos

- Usar a instrução CALL, com o nome do procedimento desejado e respectivos argumentos.
- Exemplo:

```
CALL MenuPipaVelha('Guinness', 3.00);
```

- As funções podem ser usadas em expressões SQL desde que o valor devolvido seja do tipo apropriado.

PSM

Tipos de instruções PSM (1)

- RETURN <expressão> define o valor devolvido pela função.
  - Ao contrário de C, etc., RETURN *não* termina a execução da função.
- DECLARE <nome> <tipo> usado para declarar variáveis locais.
- BEGIN . . . END quando há mais do que uma instrução.
  - Separar as instruções por “;”.

PSM

Tipos de instruções PSM (2)

- Instruções de atribuição:

  SET <variável> = <expressão>;

  - Exemplo: `SET c = 'Abadia';`
- Etiquetas: dar nome às instruções colocando um nome e ":" como prefixo.

PSM

Instruções IF

- Forma mais simples:
  IF <condição> THEN <instrução(ões)>
  END IF;
- Acrescentar ELSE <instrução(ões)> caso se queira,
  IF . . . THEN . . . ELSE . . . END IF;
- Acrecentar casos adicionais fazendo ELSEIF <instrução(ões)>: IF ... THEN ... ELSEIF ... THEN ... ELSEIF ... THEN ... ELSE ... END IF;

PSM

Exemplo: IF (1)

- Vamos classificar os bares em função do seu número de clientes, com base em Frequenta(cliente,bar) e com as seguintes regras:
  - <100 clientes: 'impopular'.
  - 100-199 clientes: 'média'.
  - >= 200 clientes: 'popular'.
- A funcão Classifica(b) classifica o bar b.

PSM

Exemplo: IF (2)

```
CREATE FUNCTION Classifica (IN b CHAR(20) )
        RETURNS CHAR(10)
   DECLARE custo INTEGER;
   BEGIN
        SET custo = (SELECT COUNT(*) FROM Frequenta
                     WHERE bar = b);
      IF custo < 100 THEN RETURN 'impopular'
         ELSEIF custo < 200 THEN RETURN 'média'
         ELSE RETURN 'popular'
      END IF;
END;
```

Número de clientes do bar b

Instrução IF

A devolução do valor da função acontece aqui e não quando a instrução RETURN aparece

PSM

Ciclos

- Estrutura básica:

  <nome do ciclo>: LOOP <instruções>
  END LOOP;

- Sai-se de um ciclo fazendo:

  LEAVE <nome do ciclo>

PSM

Exemplo: sair de um ciclo

```
ciclo1: LOOP
        . . .
        LEAVE ciclo1;
        . . .
END LOOP;
```

← Se esta instrução for executada . . .

← O controlo passa para aqui

PSM

Outros tipos de ciclos

- WHILE <condição> DO
  <instruções>
  END WHILE;
- REPEAT <instruções>
  UNTIL <condição> END REPEAT;

PSM

Consultas

- Consultas genéricas do tipo SELECT-FROM-WHERE *não* são permitidas em PSM.
- Há três formas de obter os valores de uma consulta:
  1. Consultas que devolvem um só valor podem ser o lado direito de uma instrução de atribuição.
  2. SELECT . . . INTO em consultas que devolvem um só registo.
  3. Cursores.

PSM

Exemplo: instrução de atribuição/consulta

- Utilizando a variável local *p* e Vende(bar, cerveja, preco), podemos obter o preco da cerveja Abadia no bar Pipa Velha fazendo:

```
SET p = (SELECT preco FROM Vende
    WHERE bar = 'Pipa Velha' AND
        cerveja = 'Abadia');
```

PSM

SELECT ... INTO

- Outra forma de obter um valor de uma consulta que devolve um só tuplo consiste em colocar a cláusula INTO <variável> depois da cláusula SELECT.
- Exemplo:

```
SELECT preco INTO p FROM Vende
WHERE bar = 'Pipa Velha' AND
      cerveja = 'Abadia';
```

PSM

Cursores

- Um *cursor* é basicamente um índice de uma estrutura constituída pelos tuplos-resultado de uma dada consulta.
- Declara-se um cursor *c* fazendo:

DECLARE c CURSOR FOR <consulta>;

PSM

Abrir e fechar cursores

- Para se usar o cursor *c*, é necessário fazer:

  OPEN c;

  - A consulta de *c* é avaliada, e *c* (o cursor) é colocado no primeiro tuplo do resultado.
- Para terminar a utilização de *c* fazer:

  CLOSE c;

PSM

Obtenção de tuplos através de cursores

- Para obter o tuplo seguinte do cursor c, fazer:

  FETCH FROM c INTO x1, x2,…,x*n* ;

- Os *x* 's são uma lista de variáveis, uma para cada componente do tuplo referenciado por *c*.
- c passa automaticamente para o tuplo seguinte.

PSM

Finalização de ciclos com cursores (1)

- A forma mais comum de utilizar um cursor é através da criação de um ciclo, usando a instrução FETCH a cada iteração e fazendo alguma operação com o tuplo referenciado.
- Uma questão importante é a de saber como se sai do ciclo quando o cursor chegou ao fim.

PSM

Finalização de ciclos com cursores (2)

- Cada operação SQL devolve um *estado*, que é uma *string* com 5 dígitos.
  - Por exemplo, 00000 = "Everything OK," e 02000 = "Failed to find a tuple."
- Em PSM, pode-se obter o valor do estado através da variável SQLSTATE.

PSM

Finalização de ciclos com cursores (3)

- Podemos declarar uma *condição*, que é verdade se e só se SQLSTATE tiver um dado valor.
- Exemplo: Podemos declarar a condição `NaoExiste` para representar 02000 através de:

```
DECLARE NaoExiste CONDITION FOR
  SQLSTATE '02000';
```

PSM

Finalização de ciclos com cursores (4)

- A estrutura de um ciclo com cursor é:

```
cicloCursor: LOOP
  …
  FETCH c INTO … ;
  IF NaoExiste THEN LEAVE cicloCursor;
  END IF;
  …
END LOOP;
```

PSM

Exemplo: cursor

- Vamos escrever um procedimento que examine Vende(bar, cerveja, preco), e que aumente em 1€ o preço de todas as cervejas do bar Pipa Velha cujo preço seja inferior a 3€.
  - É claro que se pode fazer o mesmo usando um simples UPDATE, mas fica aqui como exemplo da utilização de cursores.

PSM

Exemplo: cursor

```
CREATE PROCEDURE AumentoPV( )
    DECLARE aCerveja CHAR(20);
    DECLARE oPreco NUMBER;
    DECLARE NaoExiste CONDITION FOR
        SQLSTATE '02000';
    DECLARE c CURSOR FOR
        (SELECT cerveja, preco FROM Vende
         WHERE bar = 'Pipa velha');
```

Usam-se para guardar os valores de cerveja e preco ao percorrer os registos com o cursor c

Devolve o menu do bar Pipa Velha

PSM

Exemplo: o corpo do procedimento

```
BEGIN
   OPEN c;
   cicloMenu: LOOP
         FETCH c INTO aCerveja, oPreco;
         IF NaoExiste THEN LEAVE cicloMenu END IF;
         IF oPreco < 3.00 THEN
            UPDATE Vende SET preco = oPreco + 1.00
            WHERE bar = 'Pipa Velha' AND cerveja = aCerveja;
         END IF;
   END LOOP;
   CLOSE c;
END;
```

Verifica se o último FETCH falhou na obtenção de um tuplo

Se a Pipa Velha cobrar menos de 3€ pela cerveja, o preço aumenta 1€ (no bar Pipa Velha, claro).

SQL embebido

- Ideia chave: Um pre-processador transforma instruções SQL em chamadas de procedimentos compatíveis com o código da linguagem hospedeira.
- Todas as instruções SQL embebidas começam com EXEC SQL, para que o pre-processador as possa encontrar mais facilmente.

SQL embebido

Variáveis partilhadas

- Para ligar SQL e programas em linguagens hospedeiras, as duas partes têm de partilhar algumas variáveis.
- Declarações de variáveis partilhadas são delimitadas por:

```
EXEC SQL BEGIN DECLARE SECTION;
<declarações na linguagem hospedeira>
EXEC SQL END DECLARE SECTION;
```

É sempre necessário

SQL embebido

Utilização de variáveis partilhadas

- Em SQL as variáveis partilhadas têm de ser precedidas por ":".
  - Podem ser usadas como constantes providenciadas pelo programa em linguagem hospedeira.
  - Os seus valores podem ser obtidos através de instruções SQL e e serem posteriormente passados para o programa em linguagem hospedeira.
- Na linguagem hospedeira, as variáveis partilhadas comportam-se como qualquer outra variável.

SQL embebido

Exemplo: consulta de preços

- Vamos usar C com SQL embebido para exemplificar quais as partes importantes de uma função que obtem uma cerveja e um bar, e consulta o preço dessa cerveja nesse bar.
- Assume-se qua a base de dados tem a tabela Vende(bar, cerveja, preco), como de costume.

SQL embebido

Exemplo: C e SQL

```
EXEC SQL BEGIN DECLARE SECTION;
  char oBar[21], aCerveja[21];
  float oPreco;
EXEC SQL END DECLARE SECTION;
  /* obtem valores para oBar e aCerveja */
  EXEC SQL SELECT preco INTO :oPreco
  FROM Vende
  WHERE bar = :oBar AND cerveja = :aCerveja;
  /* faz qualquer coisa com oPreco */
```

Notar que são precisos arrays de 21-char para 20 chars + marca de fim

SELECT-INTO como em PSM

SQL embebido

Consultas embebidas

- SQL embebido tem as mesmas limitações que PSM no que diz respeito a consultas:
  - SELECT-INTO para uma consulta que garantidamente dê um único tuplo como resultado.
  - Caso contrário, tem de se usar cursores.
    - Pequenas diferenças sintáticas, mas as mesmas ideias chave.

SQL embebido

Consultas embebidas

- Declarar um cursor *c* com:

EXEC SQL DECLARE *c* CURSOR FOR <consulta>;

- Abrir e fechar o cursor c com:

EXEC SQL OPEN CURSOR c;

EXEC SQL CLOSE CURSOR c;

- Obter valores a partir de *c* com:

EXEC SQL FETCH c INTO <variavel(is)>;

  - Macro NOT FOUND é TRUE se e só se o FETCH falha na procura de um tuplo.

SQL embebido

Exemplo: imprimir o menu de Pipa Velha

- Vamos usar C + SQL para imprimir o menu de Pipa Velha – a list de pares cerveja-preco existentes em Vende(bar, cerveja, preco) com bar = Pipa Velha.
- Um cursor vai percorrer os tuplos da tabela Vende com bar = 'Pipa Velha'.

SQL embebido

Exemplo: declarações

```
EXEC SQL BEGIN DECLARE SECTION;
   char aCerveja[21]; float oPreco;
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE c CURSOR FOR
   SELECT cerveja, preco FROM Vende
   WHERE bar = 'Pipa Velha';
```

O cursor é declarado fora da secção de declaração

SQL embebido

Exemplo: parte executável

```
EXEC SQL OPEN CURSOR c;
while(1) {
   EXEC SQL FETCH c
                INTO :aCerveja, :oPreco;
   if (NOT FOUND) break;
   /* formatar e imprimir aCerveja e oPreco */
}
EXEC SQL CLOSE CURSOR c;
```

A forma como se sai de ciclos em C

SQL embebido

A necessidade de SQL dinâmico

- A maior parte das aplicações usam instruções pré-definidas para interagir com a base de dados.
  - O SGBD compila instruções EXEC SQL ... para chamadas de procedimentos específicos e produz um programa na linguagem hospedeira que utiliza uma biblioteca.
- E quanto ao sqlplus, que não sabe do que precisa até ser executado?
  - As instruções só são conhecidas no momento em que são executadas.

SQL embebido

SQL dinâmico

- Preparação de uma consulta:

EXEC SQL PREPARE <nome da consulta>
FROM <texto da consulta>;

- Execução da consulta:

EXEC SQL EXECUTE <nome da consulta>;

- “Prepare” = optimizar consulta.
- Prepara uma vez, executa muitas vezes.

SQL embebido

Exemplo: uma interface genérica

```
EXEC SQL BEGIN DECLARE SECTION;
   char consulta[MAX_LENGTH];
EXEC SQL END DECLARE SECTION;
while(1) {
   /* Disponibiliza o prompt para introdução da consulta */
   /* lê a consulta escrita pelo utilizador para um array */
   EXEC SQL PREPARE c FROM :consulta;
   EXEC SQL EXECUTE c;
}
```

c é uma variável SQL que representa a forma optimizada de uma qualquer instrução escrita pelo utilizador e que se encontra em :consulta

SQL embebido

Execução imediata

- Se só se vai executar a consulta uma vez, podemos combinar os passos PREPARE e EXECUTE num só.
- Usar:

EXEC SQL EXECUTE IMMEDIATE <texto>;

SQL embebido

Exemplo: outra vez a interface genérica

```
EXEC SQL BEGIN DECLARE SECTION;
  char consulta[MAX_LENGTH];
EXEC SQL END DECLARE SECTION;
while(1) {
  /* Disponibiliza o prompt */
  /* Lê a consulta para um array*/
  EXEC SQL EXECUTE IMMEDIATE :consulta;
}
```