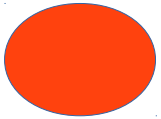
Desenvolvimento de aplicações com banco de dados

Programas de aplicações de banco de dados

Prof. André Britto



Aplicações de BD

- A maioria dos acessos ao banco de dados é feito pelo usuário final.
- Não possui conhecimento técnico, logo precisa de uma aplicação para acessar o banco de dados.
  - Interfaces de acesso simples.

Aplicações com BD

- As aplicações de banco de dados seguem três abordagens:
  - Comandos incorporados em linguagens de programação.
  - Uso de bibliotecas de funções de acesso ao banco de dados, APIs.
  - Linguagem de programação de banco de dados.

Aplicações de BD

- Uso de bibliotecas de funções de acesso ao banco de dados (APIs).
  - Biblioteca de funções (API) disponibilizam funções para o acesso e manipulação do banco de dados.
  - As consultas são inseridas como parâmetros dessas funções.
  - A biblioteca SQL/CLI é disponibilizada para a linguagem C e a JDBC para a linguagem Java.

Aplicações de BD

- Comandos incorporados em linguagens de programação.
  - Comandos de acesso ao banco de dados são incorporados dentro de uma linguagem de programação.
    - Host languages.
  - Esses comandos passam por um pré-compilador que processa os comandos do banco de dados.
  - Essa técnica é chamada de SQL incorporado para linguagem C e SQLJ para a linguagem Java.

Aplicações de BD

- Linguagem de programação de banco de dados:
  - Utilizadas para construir stored procedures.
  - Um exemplo é a linguagem PL/SQL (PL/pgSQL no postgres).

Aplicações de BD

- Linguagem de programação de banco de dados:
  - Uma linguagem de programação de banco de dados é desenvolvida especialmente para ser compatível com o modelo de dados e linguagem de consulta.
  - Além dos comandos da linguagem de consulta, a linguagem de programação de banco de dados possui comandos de laço e condição para ter um maior poder de expressão.

Incompatibilidade de Impedâncias

- Problemas que ocorrem entre o modelo do banco de dados e o modelo da linguagem de programação.
- Diferença dos tipos das linguagens de programação e os tipos de atributos do modelo de dados.
- É feito um mapeamento (binding) entre os tipos semelhantes dos dois modelos.

Incompatibilidade de Impedâncias

- Outro problema é referente à estrutura de dados que irá armazenar o resultado da consulta.
- A linguagem deve prover um mecanismo para armazenar os dados do retorno da consulta e um meio para iterar por esse retorno.

- Mapeameto dos tipos SQL em Java

SQL type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
BINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time

Uso de APIs

- Diversas linguagens disponibilizam bibliotecas para acessar e fazer consultas em banco de dados ou arquivos.
  - Java - JDBC.
- As consultas são construídas de forma dinâmica.
- São passadas como parâmetros de um função que acessa o banco de dados.
- Retorno na consultas segue um padrão.

JDBC

- Torna fácil enviar comandos SQL para os SGBDs, mas vai além do SQL pois permite interagir com outros tipos de fontes de dados, como arquivos.
- Interoperabilidade: o mesmo programa Java acessa Oracle, Postgres, MySQL,etc.
- Serve de base para outras APIs, como SQLJ.

JDBC

- Características:
  - fácil mapeamento para o modelo relacional;
  - independência de banco de dados;
  - independente de plataforma;

JDBC

- Procedimento básico:
  - Estabelecer conexão.
  - Criar statments.
  - Executar consultas.
  - Iterar sobre as consultas.
  - Fechar conexão.

JDBC

- Exemplo:

```
Connection con = DriverManager.getConnection(“jdbc:meuDriver:meuDB”, “meuLogin”,
“minhaPassword”);

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(
   “Select mat_professor from universidade.professor”);
While (rs.next()) {
int  mat = rs.getInt(“matricula”);
String nome =  rs.getString(“nome”);
System.out.println( mat + “,” + nome );
}
rs.close();
…
```

JDBC

- Além de efetuar consultas também podemos:
  - Chamar stored procedures.
  - Ativar triggers.
  - Executar transações.

Comandos incorporados

- Comandos **incorporados** em linguagens de programação.
- São definidas palavras-chave para identificar o trecho do código que se refere ao banco de dados.
- O código é compilado por um **pré-compilador**.

SQLJ

- SQLJ é uma tecnologia que permite a um programa Java acessar um banco de dados utilizando comandos SQL embutidos.
  - Arquivo fonte “.sqlj”
  - Comandos possíveis no SQLJ
    - Create, Alter, Drop
    - Select, Insert, Update, Open, Fetch
    - Controle de Transações: Commit, Rollback

SQLJ

- SQLJ não pode ser compilado diretamente pelo compilador Java
- Solução: usar um tradutor
  - SQLJ Translator

SQLJ

- O tradutor verifica erros de sintaxe e semântica de SQL em tempo de compilação:
  - Nomes incorretos;
  - Checagem de tipos;
  - Verifica se o comando SQL está de acordo com o Banco de Dados;
  - Entre outros.

SQLJ

- Palavra chave:
  - #sql{<comando sql>}
- Antes de executar os comandos deve ser criada a conexão com o banco de dados.
  - Contexto – DefaultContext
  - sqlj.runtime.ref.ConnectionContextImpl
  - +--sqlj.runtime.ref.DefaultContex

```
import java.sql.SQLException;
import oracle.sqlj.runtime.Oracle;

// iterator for the select
#sql iterator MyCheckedIter (String ITEM_NAME);

public class TesteSQLJ {
    //Main method
    public static void main (String args[]){
        try{
            Oracle.connect(TesteSQLJ.class, "connect.properties");
            TesteSQLJ ti = new TesteSQLJ();
            ti.runExample();
        } catch (SQLException e){
            System.err.println("Error running the example: " + e);
        } finally {
            try {
                Oracle.close();
            } catch (SQLException e) { }
        }
    } //End of method main

    //Method that runs the example
    void runExample() throws SQLException {
      //Issue SQL command to clear the SALES table
      #sql { DROP TABLE SALES};

      #sql { CREATE TABLE SALES (ITEM_NUMBER NUMBER, ITEM_NAME CHAR(30), SALES_DATE DATE,
                    COST NUMBER, SALES_REP_NUMBER NUMBER, SALES_REP_NAME CHAR(20))};
      #sql { INSERT INTO SALES(ITEM_NAME) VALUES ('Hello, SQLJ Checker!')};

      MyCheckedIter iter;
      #sql iter = { SELECT ITEM_NAME FROM SALES };

      while (iter.next()) {
          System.out.println(iter.ITEM_NAME());
      }
    }
}
```

Stored procedures

- Outra forma de desenvolver aplicações de banco de dados é através de stored procedures.
  - Ficam persistentes no banco de dados.
  - Podem ser executadas através de programas externos.

Stored procedures

- O SGBD pode executar esses procedimentos caso ocorra algum efeito colateral (triggers).
- Podem ser escritas em linguagens de programação específicas par ao banco de dados ou de uso geral.

Stored procedures

- Linguagens de banco de dados
  - São desenvolvidas exclusivamente para o modelo de dados do SGBD.
  - Não sofrem da **incompatibilidade de impedâncias**.
  - Exemplo:
    - PL/SQL (Oracle), SQL/PSM

Stored procedures

- SQL/PSM
  - Disponibilizada pelo padrão da linguagem SQL.
  - Utilizada para a criação e escrita de stored procedures.
  - Combina:
    - Comandos da linguagem SQL.
    - Estruturas de controle para aumentar a expressividade dos programas.
    - Condições, laços, etc.

Stored procedures

- PL/pgSQL
  - Linguagem procedural para o PostgresSQL.
  - Usada para criar funções e triggers.
  - Junta estruturas de programação com comandos SQL.
  - Pode ser utilizada para efetuar computações complexas.

Stored procedures

- PL/pgSQL
  - Trabalha e retorna:
    - Tipos escalares;
    - Arrays;
    - Registros que representam linhas retornadas por uma consulta;
    - Tipos polimórficos;
    - Sets (Tabelas)
    - Void.

PL/pgSQL

- Sintaxe

```
[ <<label>> ]
[ DECLARE
    declarations ]
BEGIN
    statements
END [ label ];
```

```
CREATE FUNCTION somefunc() RETURNS integer AS $$
<< outerblock >>
DECLARE
  quantity integer := 30;
BEGIN
   RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 30
   quantity := 50;
   DECLARE
      quantity integer := 80;
   BEGIN
      RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 80
      RAISE NOTICE 'Outer quantity here is %', outerblock.quantity;  -- Prints 50
   END;
   RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 50
   RETURN quantity;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION universidade.aplicar_aumento(cod VARCHAR, porcentagem REAL) RETURNS real AS $$
<<local>>
DECLARE
    departamento_row universidade.departamento%ROWTYPE;
    orcamento REAL;
    aumento REAL;

BEGIN
    SELECT * INTO departamento_row FROM universidade.departamento AS d WHERE d.cod_depto = cod ;
    orcamento := departamento_row.orcamento;
    aumento := orcamento * porcentagem;

IF(aumento < 1000) THEN
        orcamento = orcamento + 1000;
   ELSE
        orcamento = orcamento + aumento;
    END IF;
    UPDATE universidade.departamento SET orcamento = local.orcamento WHERE cod_depto = cod;
    RETURN aumento;
END;
$$ LANGUAGE plpgsql;
```

Comparação entre as três abordagens

- Comandos incorporados
  - Vantagem
    - Possui a vantagem de ser compilado.
    - Checa se há erro nas consultas em tempo de compilação.
    - As consultas são mais simples, pois se conhece os seus retornos quando está se escrevendo o código.
  - Desvantagem
    - A consulta deve ser conhecida em tempo de compilação.
    - Não se pode construir consultas dinâmicas.
  - Essa abordagem é recomenda caso sejam necessárias consultas que se repetem muito na aplicação.

Comparação entre as três abordagens

- Uso de APIs
  - Vantagem
    - Abordagem mais flexível.
    - Consultas são construídas de forma dinâmica.
  - Desvantagem
    - Mais propensa a erros.
    - Checagem é feita somente em tempo de execução.
    - Não se sabe previamente o retorno das consultas.
- É recomendável quando é necessário construir consultas em tempo de execução

Comparação entre as três abordagens

- Linguagens de banco de dados
  - Vantagem
    - Não sofre da incompatibilidade de impedâncias.
    - Ficam armazenadas diretamente no banco de dados.
    - Podem ser executadas por diferentes aplicações.
    - São executadas de forma mais rápida.
  - Desvantagem
    - É preciso conhecer a linguagem específica para cada modelo de dados.
    - Não possui tanta expressividade e recursos quanto as linguagens de programação de uso geral.

Leitura recomendada

- ELMASRI, R; NAVATHE, S.B. **Sistemas de Banco de Dados**, Addison Wesley, 6ª Edição.
  - Capítulos 28 e 29.
- Silberschatz, A; Korth H.F.; Sudarshan S. **Sistemas de Banco de Dados**, Editora Campus, 6ª Edição.
  - Capítulo 5.
- Material de aula do professor Cláudio de Souza Baptista – UFCG.