

10. *Aplicações usando CGI, ASP e JSP*

Este módulo apresenta exemplos de aplicações usando CGI, JSP e ASP. Não pretende mostrar como construir aplicações em ASP e JSP pois tais tecnologias exigem conhecimentos adicionais que fogem ao escopo deste curso (Java, OLE/ActiveX e VBScript). São apresentados programas simples que poderão ser comparados com programas em Perl desenvolvidos durante o curso e que poderão ser demonstrados em sala de aula.

10.1. *Aplicações de acesso a bancos de dados*

Para os exemplos deste módulo, usamos o arquivo Access `anuncios.mdb` distribuído no disquete. Não é preciso ter Access instalado para usá-lo, mas é preciso ter um driver para o MS-Access disponível no Windows. Você também pode construir o arquivo em outro banco de dados. Ele consiste de apenas uma tabela (por simplicidade) com quatro colunas e inicialmente 6 registros. O formato das colunas é o seguinte:

Título	numero	data	texto	autor
Tipo	INT PRIMARY KEY	CHAR(24)	CHAR(8192)	CHAR(50)

É possível desenvolver programas CGI que realizem a conexão direta a um banco de dados específico, utilizando seus drivers proprietários. Como focamos em tecnologias abertas, mostraremos algumas aplicações que usam drivers abertos, baseados em ODBC e JDBC.

ODBC

Para criar e administrar bancos de dados relacionais precisamos ter um ambiente próprio, geralmente fornecido pelo fabricante. Para usar esses bancos de dados dentro de aplicações, precisamos de uma maneira de encapsular o SQL dentro de uma linguagem de programação, já que embora SQL seja eficiente na administração de bancos de dados, ela não possui recursos de uma linguagem de programação de propósito geral. Usando SQL podemos ter

acesso a bancos de dados de uma forma padrão dentro de programas escritos em C ou C++ através da interface ODBC.

ODBC – Open Database Connectivity é uma interface de baixo nível baseada na linguagem C que oferece uma interface consistente para a comunicação com um banco de dados usando SQL. Surgiu inicialmente como um padrão para computadores desktop, desenvolvido pela Microsoft, mas em pouco tempo tornou-se um padrão de fato da indústria. Todos os principais fabricantes de bancos de dados dispõem de drivers ODBC.

ODBC possui diversas limitações. As principais referem-se à dependência de plataforma da linguagem C (recentemente, também em Perl), dificultando o porte de aplicações ODBC para outras plataformas. Diferentemente das aplicações desktop, onde praticamente domina a plataforma Windows, aplicações de rede e bancos de dados freqüentemente residem em máquinas bastante diferentes. A independência de plataforma nesses casos é altamente desejável. Java oferece as vantagens de ODBC juntamente com a independência de plataforma com sua interface JDBC.

Muitos bancos de dados já possuem drivers JDBC, porém é possível ainda encontrar bancos de dados que não os possuem, mas têm drivers ODBC. Também, devido a ubiquidade da plataforma Windows, que contém um conjunto de drivers ODBC nativos, é interessante poder interagir com esses drivers em várias ocasiões, por exemplo, ao montar um banco de dados SQL baseado em arquivos de texto. Como a plataforma Java contém um driver JDBC para ODBC, podemos usar JDBC em praticamente qualquer banco de dados.

JDBC

Java Database Connectivity - JDBC, é uma interface baseada em Java para acesso a bancos de dados através de SQL. É uma versão Java de ODBC - uma alternativa que acrescenta ao ODBC a portabilidade entre plataformas. Oferece uma interface uniforme para bancos de dados de fabricantes diferentes, permitindo que sejam manipulados de uma forma consistente. O suporte a JDBC é proporcionado por uma API Java padrão (pacote java.sql) e faz parte da distribuição Java. Usando JDBC, pode-se obter acesso direto a bancos de dados através de applets e outras aplicações Java.

JDBC é uma interface de nível de código. Consiste de um conjunto de classes e interfaces que permitem embutir código SQL como argumentos na invocação de seus métodos. Por oferecer uma interface uniforme, independente de fabricante de banco de dados, é possível construir uma aplicação Java para acesso a qualquer banco de dados SQL. A aplicação poderá ser usada com qualquer banco de dados que possua um driver JDBC: Sybase, Oracle, Informix, ou qualquer outro que ainda não inventado, desde que implemente um driver JDBC.

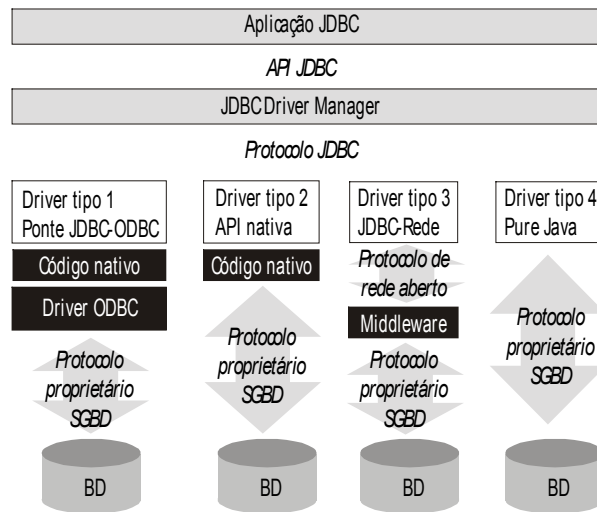
Para que se possa usar JDBC na comunicação com um banco de dados, é preciso que exista um driver para o banco de dados que implemente os métodos JDBC. Para que uma aplicação se comunique com um banco de dados, ela precisa carregar o driver (pode ser em tempo de execução) e obter uma conexão ao mesmo. Depois de obtida a conexão, pode-se enviar requisições de pesquisa e atualização e analisar os dados retornados usando métodos Java e

passando instruções SQL como argumentos. Não é preciso conhecer detalhes do banco de dados em questão. Em uma segunda execução do programa, o programa pode carregar outro driver e utilizar os mesmos métodos para ter acesso a um banco de dados diferente.

Muitos bancos de dados não têm driver JDBC, mas têm driver ODBC. Por causa disso, um driver JDBC para bancos de dados ODBC é fornecido pela Sun e incluído na distribuição Java. Com essa ponte JDBC-ODBC é possível usar drivers ODBC através de drivers JDBC. Esse driver é somente um dos quatro tipos diferentes de drivers JDBC previstos pela especificação.

A figura abaixo mostra um diagrama em camadas da arquitetura JDBC ilustrando os diferentes tipos de drivers. Existem quatro tipos de drivers JDBC:

- Tipo 1 - drivers que usam uma ponte para ter acesso a um banco de dados. Este tipo de solução geralmente requer a instalação de software do lado do cliente. Um exemplo de driver do tipo 1 é a ponte JDBC-ODBC distribuída pela Sun na distribuição Java.
- Tipo 2 - drivers que usam uma API nativa. Esses drivers contêm métodos Java implementados em C ou C++. São Java na superfície e C/C++ no interior. Esta solução também requer software do lado do cliente. A tendência é que esses drivers evoluam para drivers do tipo 3
- Tipo 3 - drivers que oferecem uma API de rede ao cliente para que ele possa ter acesso a uma aplicação middleware no servidor que traduz as requisições do cliente em uma API específica ao driver desejado. Esta solução não requer software do lado do cliente.
- Tipo 4 - drivers que se comunicam diretamente com o banco de dados usando soquetes de rede. É uma solução puro Java. Não requer código do lado do cliente. Este tipo de driver geralmente é distribuído pelo próprio fabricante do banco de dados.



Arquitetura JDBC

A ponte JDBC-ODBC distribuída juntamente com o JDK é um driver do tipo 1. Nos exemplos apresentados neste trabalho, utilizamos esse driver apenas para acesso local através do ODBC nativo do Windows. É o menos eficiente de todos pois não permite otimizações e é dependente das limitações do driver com o qual faz ponte.

Uma aplicação JDBC pode carregar ao mesmo tempo diversos drivers. Para determinar qual driver será usado em uma conexão, uma URL é passada como argumento do método usado para obter uma conexão. Esta URL tem a sintaxe seguinte:

```
jdbc:subprotocolo:dsn
```

O subprotocolo é o nome do tipo de protocolo de banco de dados que está sendo usado para interpretar o SQL. É um nome dependente do fabricante. A aplicação usa o subprotocolo para identificar o driver a ser instanciado. O dsn é o nome que o subprotocolo utilizará para localizar um determinado servidor ou base de dados. Pode ser o nome de uma fonte de dados do sistema local (Data Source Name) ou uma fonte de dados remota. Veja alguns exemplos:

```
jdbc:odbc:anuncios
jdbc:oracle:contas
jdbc:mysql://alnitak.orion.org/clientes
```

Para conhecer mais sobre JDBC é preciso conhecer Java, o que está fora do escopo deste curso. Mais adiante será apresentado um exemplo de acesso a bancos de dados usando JSP que utiliza um driver JDBC para realizar a conexão.

10.2. Acesso ODBC usando Perl

CGI para acesso via ODBC

O módulo usado nos nossos exemplos (no disquete) foi obtido de <http://www.roth.net> e é compatível com o *ActivePerl* (www.activestate.com). Neste site, pode-se baixar todo o módulo ODBC para instalação ou apenas os arquivos já compilados e colocá-los nos lugares adequados. Veja o capítulo 5 (Perl) para mais detalhes sobre este módulo e como instalá-lo.

O exemplo abaixo mostra um acesso simples a um banco de dados Access. Para que funcione é preciso que o módulo ODBC esteja instalado (seção anterior) e que haja uma fonte de dados ODBC (Data Source) no sistema com o nome “mdbdados1” que aponte para a base `anuncios.mdb`, distribuída no disquete. O programa apenas lista o conteúdo da base.

```
#!c:\perl\bin\perl.exe
use Win32::ODBC;
print "Content-type: text/html\n\n";

&buscaTudo;
exit(0);

sub buscaTudo
{
    $sql = "SELECT * FROM anuncios;";
    $dsn = "mdbdados1";
    $bd = new Win32::ODBC($dsn);
    $bd->Sql($sql);
    print "<table border=1>";
    while ($bd->FetchRow())
    {
```

```

        @registros = $bd->Data("numero", "data", "texto", "autor");
        print "<tr valign=top>";
        print "<td>$registros[0]</td>";
        print "<td>$registros[1]</td>";
        print "<td><pre>$registros[2]</pre></td>";
        print "<td>$registros[3]</td>";
        print "</tr>";
    }
    print "</table>";
    $bd->Close();
}

```

Supondo que o programa acima esteja armazenado em um arquivo dados.pl, situado no diretório CGI-BIN do Apache que roda em localhost:8080, a URL `http://localhost:8080/cgi-bin/dados.pl` digitada em um browser, em um link ou no atributo ACTION de um formulário devolverá uma página com os dados formatados em uma tabela. Veja outros exemplos, com acesso completo, atualização, inserção, remoção e busca nesta base, no disquete que acompanha esta apostila.

10.3. *Acesso usando Servlets*

Para utilizar servlets ou JSP é preciso que o servidor os suporte. Isto pode ser uma característica nativa do servidor ou pode ser implementado através de módulos externos (plug-ins) como o JRun e o Jakarta (produtos disponíveis gratuitamente na Internet). Os exemplos a seguir foram executados em servidores Personal Web Server da Microsoft, rodando em Windows98 e NT, usando o módulo Allaire JRun.

Servlets são aplicações Java. Não são aplicações standalone (como programas CGI escritos em Java). Também não executam como applets. Para executar um servlet é preciso primeiro instalá-lo em um servidor Web. Depois, é necessário enviar uma requisição do cliente ao servidor, que o faça iniciar o servlet. Um servlet HTTP é um componente Java que pode ser usado como extensão de um servidor Web, assim como um applet pode ser usado como extensão de um browser. Com um servlet, o servidor HTTP pode oferecer serviços adicionais e personalizados como suporte a novos protocolos de comunicação, geração automática de páginas, acesso a banco de dados, controle remoto de aplicações e dispositivos, etc.

Dependendo de como o servlet é instalado, ele pode ser iniciado uma única vez e permanecer ativo até que o servidor seja inicializado, ou pode ser iniciado quando for requisitado por um cliente e destruído quando o cliente terminar o serviço. O primeiro tipo é chamado de servlet permanente; o segundo é um servlet temporário.

A API para desenvolvimento de servlets distribuída pelo Java Servlet Development Kit (JSDK) consiste de dois pacotes:

- javax.servlet - contém classes e interfaces para o desenvolvimento e uso de servlets genéricos

- javax.servlet.http - classes e interfaces para o desenvolvimento de servlets HTTP.

Módulo JRun

O módulo JRun é o mais popular plug-in para servidores Web que os habilita a suportar servlets. Distribuído gratuitamente pela LiveSoftware (www.livesoftware.com) e recentemente comprado pela Allaire, pode ser instalado nos principais servidores das plataformas Windows, Macintosh e Unix.

Cada servidor tem a sua interface característica. A interface do JRun é bastante diferente daquela do servletrunner ou do Java Web Server (Sun). A instalação dos servlets é realizada através de uma tela gráfica onde se pode definir duas propriedades e outras informações de inicialização.

Entre os servidores disponíveis no laboratório, o JRun pode ser instalado em todos (O'Reilly WebSite for Windows, Personal Web Server, Internet Information Server, Netscape Fastrack Server e Apache for Linux), exceto no Apache for Windows. Após a instalação, um diretório especial é mapeado especialmente para a execução dos servlets.

Exemplo: Hello World

O exemplo abaixo ilustra um servlet simples (a título de exemplo, pois Java está fora do escopo deste curso). Para rodá-lo é preciso compilar o código Java abaixo usando as bibliotecas (pacotes) javax.servlet, instalá-lo no servidor e executá-lo através de uma requisição do browser. Se o servlet (arquivo .class compilado) estiver instalado na máquina local, usando JRun e PWS na configuração default, a URL para executá-lo deve ser `http://localhost/servlet/SimpleServlet`.

```

/*
 * Copyright (c) 1996-1997 Sun Microsystems, Inc. All Rights Reserved.
 * Hello World para Servlets
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleServlet extends HttpServlet
{
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException
    {
        PrintWriter out;
        String title = "Simple Servlet Output";
        out = response.getWriter();
        out.println("<HTML><HEAD><TITLE>");
    }
}

```

```

        out.println(title);
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>" + title + "</H1>");
        out.println("<P>This is output from SimpleServlet.");
        out.println("</BODY></HTML>");
        out.close();
    }
}

```

Acesso a banco de dados usando Servlets

Para entender como funcionam os servlets é preciso entender Java. O exemplo abaixo está incluído aqui para que possa ser comparado às outras tecnologias utilizadas. A listagem contém o código-fonte. É preciso compilá-lo, gerar um arquivo-objeto (com extensão .class) e instalá-lo no servidor. Para executá-lo, pode-se usar uma URL localizada no atributo ACTION de um formulário, link ou digitada diretamente no browser. No PWS use a URL: <http://nomedamaquina/servlet/Dados> para executá-lo.

```

import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Dados
    extends HttpServlet {

    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws ServletException{

        PrintWriter out = null;
        try {
            out = response.getWriter();
            response.setContentType("text/html");
            out.println("<html><body>");
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con =
                DriverManager.getConnection("jdbc:odbc:mdbdados1", "", "");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM anuncios");
            out.println("<table border=1>");
            out.println("<tr><td>Número</td>    <td> Autor </td><td>" +
                "Texto </td><td> Data </td></tr>");
            while (rs.next()) {
                String aut = rs.getString("autor");
                String txt = rs.getString("texto");
                String dat = rs.getString("data");
                int num = rs.getInt("numero");
            }
        }
    }
}

```

```

        out.println("<tr><td>" + num + "</td><td>" + aut + "</td><td>" +
                    txt + "</td><td>" + dat + "</td></tr>");
    }
    out.println("</table>");

} catch (Exception e) {
    out.println("<h1>Exceção: " + e + "</h1>");
}
out.println("</body></html>");
out.close();
}
}

```

10.4. *Acesso usando Active Server Pages*

Active Server Pages ou ASP é uma solução da Microsoft compatível com o Internet Information Server (IIS) e, através de plug-in, com os principais servidores do mercado. Consiste de páginas HTML que podem conter *scripts* e interpretados pelo servidor e fazer o mesmo (ou mais) que CGI ou *servlets*.

As páginas têm a extensão .ASP para que o servidor possa identificá-las como tal. São páginas HTML como outras quaisquer mas possuem código de programas para o servidor embutido.

O código pode ser escrito em várias linguagens de roteiro. As mais populares são VBScript e JavaScript (JScript). Trechos do programa podem ser incluídos entre os descritores `<script>` e `</script>` usando o atributo `RUNAT="server"`:

```
<script language=VBScript runat=server> ... </script>
```

Esses blocos `<SCRIPT>` são usados principalmente para incluir trechos maiores do programa. Jamais chegam ao browser. São totalmente consumidos pelo servidor ao gerar a página HTML que será enviada.

O código ASP pode ser misturado com o HTML da página através dos descritores `<% e %>`. Qualquer coisa escrita entre `<% ... %>` é ignorado pelos servidores que não suportam o formato e por browsers, caso por algum motivo, o servidor falhe em interpretar o ASP.

ASP pode ser usado para gerar páginas dinâmicas, cujo conteúdo é definido por fatores como preferências do usuário, localidade, etc. Pode usar todos os recursos de programação do JavaScript ou VBScript inclusive para tarefas proibidas no browser como acesso a disco e à rede. Há vários objetos próprios do ASP (utilizáveis tanto em JavaScript como em VBScript) com métodos específicos para abrir conexões de bancos de dados, enviar requisições, fazer conexões de rede, gerar novas páginas, formatar páginas, salvar em disco, manter sessões com cookies de forma transparente, etc. O poder do ASP é maior quando utiliza objetos do sistema Windows, através de sua interface DCOM. Vários objetos leves ADO (*ActiveX Data Objects*)

para servidor podem ser manipulados pelo ASP. A programação em ASP e ADO está fora do escopo deste curso. Para maiores informações sobre ASP, consulte a documentação do Internet Information Server ou Personal Web Server da Microsoft que contém tutoriais sobre o tema

Uma página ASP é um programa que gera uma nova página. Pode ser usada como destino (ACTION) de um formulário. Usar ASP em vez de ISAPI, CGI ou servlets pode ser uma boa idéia quando há mais texto HTML para ser copiado sem alterações que modificações a serem feitas na página. Com CGI, servlets ou ISAPI seria necessário embutir o HTML na linguagem de programação e mandar imprimir cada linha. Com ASP se faz o contrário: embute-se o programa dentro do HTML. O servidor interpreta a página, gera uma nova e envia para o browser como text/html. O browser não tem a menor idéia que a página foi gerada por ASP, CGI ou se veio diretamente do servidor.

Veja um exemplo simples de como ASP pode ser usado para criar uma página dinâmica:

```
<html><body>
<h1>Bem-vindo ao servidor da ATKM em Marte!
<h2>Evite conectar-se entre 19 e 23 horas locais. A data e hora de
hoje são:
<%=Now %> </h2>
</body></html>
```

O exemplo acima informa a hora local do servidor guardada na variável Now. O exemplo acima usa uma expressão ASP, que é expressa entre `<%=` e `%>`. Uma expressão ASP começa com `<%=` e termina com `%>`. O resultado de uma expressão é calculado no momento em que a página é lida pelo servidor.

Além das expressões, ASP também define diretivas (entre `<%@` e `%>`) que permitem definir propriedades do código, como linguagem utilizada (se não for VBScript), etc. Uma diretiva começa com `<%@` e termina com `%>`. Ela permite definir o tipo de dados de saída, importar arquivos, especificar o uso de outra linguagem em vez de Java, etc. Algumas variáveis podem receber valor nas diretivas, da forma:

```
<%@nomeVar = "valor" %>
```

A manipulação de dados em ASP é realizada através de objetos do servidor que representam entidades de entrada e saída, como o servidor, a requisição, a resposta, etc. O exemplo a seguir utiliza ainda objetos ADO para obter acesso ao driver ODBC e enviar uma requisição a um banco de dados (o mesmo usado com CGI).

Acesso ao banco de dados usando ASP

A página abaixo deve ser instalada em um diretório do Internet Information Server que esteja habilitado a rodar scripts. Deve também estar na máquina onde há uma fonte de dados

ODBC chamada mdbdados vinculada ao arquivo anuncios.mdb (veja seções anteriores). O código ASP está mostrado em negrito.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
  <title>Acesso a Dados via ASP</title>
  <%@ language="javascript" %>
  <% dsn = "mdbdados1"; %>
</head>

<body>
<h1>ASP: Informação armazenada em <%=dsn %> </h1>
<table border=1>
<tr><td>Número</td><td> Autor </td><td> Texto </td><td> Data </td></tr>
<%  con = Server.CreateObject("ADODB.Connection");
    con.Open ("DSN=" + dsn);
    rs = Server.CreateObject("ADODB.Recordset");
    rs.Open("SELECT * from anuncios", con);
    while(!rs.EOF) {
      num = rs("numero");
      txt = rs("texto");
      aut = rs("autor");
      dat = rs("data");
      rs.MoveNext();
    %>
      <tr>
      <td> <%=num %> </td>
      <td><pre> <%=txt %> </pre></td>
      <td> <%=aut %> </td>
      <td> <%=dat %> </td>
      </tr>

<%  }  %>

</table>

</body>
</html>
```

Supondo que o programa acima esteja armazenado em um arquivo dados.asp, situado na raiz do servidor em localhost, a URL <http://localhost/dados.asp> digitada em um browser, em um link ou no atributo ACTION de um formulário devolverá uma página com os dados formatados em uma tabela, idêntica àquela usando CGI.

10.5. Acesso usando Java Server Pages

A Sun possui uma tecnologia equivalente ao ASP que se chama Java Server Pages ou JSP. A sintaxe para inclusão na página é a mesma. A única diferença é que o código deve ser escrito em Java. Na execução, o código JSP é compilado e transformado em um servlet que permanece no cache do servidor e atende a outras solicitações. Em aplicações que exigem um maior desempenho do servidor, esta característica permite que páginas JSP sejam mais eficientes que páginas ASP, que são interpretadas a cada solicitação.

O exemplo abaixo ilustra uma aplicação Hello World simples escrita com JSP (código em negrito). O programa aceita um parâmetro que pode ser passado via QUERY_STRING ou entrada padrão:

```
<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>
<%
if (request.getParameter("name") == null) {
    out.println("Hello World");
}
else {
    out.println("Hello, " + request.getParameter("name"));
}
%>
</H1>
</BODY></HTML>
```

Um *scriptlet* – nome usado para se referir aos programas escritos usando JSP – possui quatro variáveis pré-definidas para facilitar a entrada e saída. O programador pode definir outras, pois tem toda a linguagem Java à disposição. Os tipos são classes Java usadas na construção de servlets. As variáveis são:

- request** – a requisição do cliente, do tipo `HttpServletRequest`
- response** – a resposta do servidor, do tipo `HttpServletResponse`
- out** – a saída padrão, um objeto `PrintWriter`
- in** – a entrada padrão, um objeto `BufferedReader`.

Além do código incluído nos *scriptlets*, JSP permite, como ASP, o uso de expressões e diretivas. Uma expressão JSP começa com `<%=` e termina com `%>`. O resultado de uma expressão é calculado no momento em que a página é lida pelo servidor, e o resultado é transformado em uma `String`. Uma diretiva começa com `<%@` e termina com `%>`. Ela permite definir o tipo de dados de saída, importar um determinado pacote, estender outra classe e até especificar o uso de outra linguagem em vez de Java. Algumas variáveis podem receber valor nas diretivas, da forma:

```
<%@nomeVar = "valor" %>
```

Quando for preciso declarar métodos, variáveis locais e praticamente embutir um *servlet* em uma página, pode-se usar declarações através do bloco `<script>` com o argumento `RUNAT=SERVER`, como ocorre em ASP. Esse bloco jamais chegará ao cliente mas será totalmente consumido pelo servidor antes que a página seja gerada. Veja um exemplo (a explicação do código Java está fora do escopo deste curso).

```
<HTML><HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>Hello, <%= getName(request) %></H1>
</BODY></HTML>

<SCRIPT RUNAT="server">
private static final String DEFAULT_NAME = "World";

private String getName(HttpServletRequest req) {
    String name = req.getParameter("name");
    if (name == null)
        return DEFAULT_NAME;
    else
        return name;
}
</SCRIPT>
```

Acesso a banco de dados usando JSP

O programa abaixo realiza a conexão a um banco de dados através do driver JDBC para ODBC (que está vinculado ao arquivo `anuncios.mdb`). O código JSP, em negrito, utiliza a linguagem Java. Compare com o código ASP e o programa CGI vistos anteriormente.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>Acesso a dados via JSP</title>
    <%@ import = "java.sql.*, java.io.*, java.util.*" %>
    <% String dsn = "mdbdados1"; %>
</head>

<body>
<h1>JSP: Informação armazenada em <%=dsn %></h1>

<table border=1>
<tr><td>Número</td><td> Autor </td><td> Texto </td><td> Data </td></tr>
<%
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```

Connection con = DriverManager.getConnection("jdbc:odbc:"+dsn);
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM anuncios");
while (rs.next()) {
    String aut = rs.getString("autor");
    String txt = rs.getString("texto");
    String dat = rs.getString("data");
    int num = rs.getInt("numero");
%>
        <tr>
        <td> <%=num %> </td>
        <td><pre> <%=txt %> </pre></td>
        <td> <%=aut %> </td>
        <td> <%=dat %> </td>
        </tr>
<%     }
    } catch (Exception e) { %>
        <h1>Exception: <%=e %> </h1>
<% } %>

</table>

</body>
</html>

```

Supondo que o programa acima esteja armazenado em um arquivo dados.jsp, situado na raiz do servidor em localhost, a URL `http://localhost/dados.jsp` digitada em um browser, em um link ou no atributo ACTION de um formulário devolverá uma página com os dados formatados em uma tabela, idêntica àquela usando CGI e ASP.

11. Apêndice: SQL

Um banco de dados relacional pode ser definido de maneira simples como um conjunto de tabelas (com linhas e colunas) onde as linhas de uma tabela podem ser relacionadas a linhas de outra tabela. Este tipo de organização permite a criação de modelos de dados compostos de várias tabelas. Uma contendo informação, outras contendo referências que definem relações entre as tabelas de informação. O acesso às informações armazenadas em bancos de dados relacionais é em geral bem mais eficiente que o acesso a dados organizados seqüencialmente ou em estruturas de árvore.

11.1. *Structured Query Language (SQL)*

Para utilizar um banco de dados relacional, é preciso ter um conjunto de instruções para recuperar, atualizar e armazenar dados. A maior parte dos bancos de dados relacionais suportam uma linguagem padrão chamada SQL – Structured Query Language. O conjunto de instruções SQL foi padronizado em 1992 de forma que as mesmas instruções podem ser usadas por bancos de dados diferentes. Mas vários fabricantes possuem extensões e certas operações mais específicas possuem sintaxes diferentes em produtos de diferentes fabricantes.

Poucos sistemas implementam totalmente o SQL92. Existem vários níveis que foram definidos durante a padronização do SQL em 1992. O conjunto mínimo de instruções é chamado de entry-level e é suportado por JDBC.

Nas seções seguintes, apresentaremos os seis principais comandos da linguagem SQL. Este não é um guia completo. Apresentamos apenas o suficiente para permitir a compreensão dos exemplos que mostraremos em JDBC. Para um tratamento mais detalhado de SQL, consulte a bibliografia no final deste capítulo.

CREATE, DROP

Antes de ilustrar a recuperação e atualização de dados em uma tabela, precisamos ter uma tabela. Para criar uma nova tabela em um banco de dados, usamos a instrução CREATE TABLE. A sintaxe básica desta instrução é:

```
CREATE TABLE nome_da_tabela
    (nome_coluna tipo_de_dados [modificadores],
    [nome_coluna tipo_de_dados [modificadores], ... ])
```

Os modificadores são opcionais e geralmente dependentes de fabricante de banco de dados. A maior parte das implementações suporta: NOT NULL, PRIMARY KEY e UNIQUE como modificadores:

```
CREATE TABLE anuncios (numero INT PRIMARY KEY,
                        data DATE,
                        texto CHAR(8192),
                        autor CHAR(50) );
```

A sintaxe exata do CREATE é dependente de banco de dados. Todos suportam a sintaxe principal (CREATE TABLE). As incompatibilidades surgem no suporte a tipos de dados e modificadores. A instrução acima funciona com o driver ODBC do Microsoft Access, via ponte ODBC-JDBC. Não funciona, no entanto com o driver JDBC do banco de dados mSQL, já que o mSQL não suporta o tipo DATE. Também não funciona com o driver ODBC para arquivos de texto, da Microsoft que não suporta o modificador PRIMARY KEY nem campos com mais de 255 caracteres.

Para remover uma tabela do banco de dados, pode-se usar a instrução DROP. A sintaxe é simples:

```
DROP TABLE nome_da_tabela
```

INSERT, UPDATE, DELETE

Depois que uma tabela é criada, dados podem ser inseridos usando a instrução INSERT. É preciso respeitar os tipos de dados definidos para cada coluna de acordo com a estrutura definida previamente para cada tabela. A sintaxe básica do INSERT é:

```
INSERT INTO nome_da_tabela (nome_da_coluna, ..., nome_da_coluna)
      VALUES (valor, ..., valor)
```

No lugar de valor, pode ser passado toda uma expressão SQL que resulte em um valor. Um exemplo do uso de INSERT, na tabela criada na seção anterior seria:

```
INSERT INTO anuncios
      VALUES (156, '13/10/1998', 'Novo anuncio!', 'Fulano');
```

O apóstrofo (') é usado para representar strings.

UPDATE permite que dados previamente inseridos sejam modificados. Sua sintaxe básica é:

```
UPDATE nome_da_tabela
      SET nome_da_coluna = valor,
          ...,
          nome_da_coluna = valor
      WHERE expressao_condicional
```

No lugar de valor, pode ser passado toda uma expressão SQL que resulte em um valor ou a palavra reservada NULL. Eis um exemplo do uso de UPDATE:

```
UPDATE anuncios
      SET texto = 'Em branco!',
          autor = ''
      WHERE codigo > 150
```

Para remover registros (linhas da tabela), usa-se DELETE:

```
DELETE FROM nome_da_tabela
      WHERE expressao_condicional
```

Por exemplo, a instrução:

```
DELETE FROM anuncios
      WHERE texto LIKE '%Para o Lixo%'
```

apaga todos os registros cujo texto contém 'Para o Lixo'.

SELECT

O comando SQL mais freqüentemente utilizado é SELECT. Com ele é possível selecionar linhas (registros) de um banco de dados de acordo com uma determinada condição. A sintaxe básica de SELECT é:

```
SELECT nome_da_coluna, ..., nome_da_coluna
      FROM nome_da_tabela
      WHERE expressão_condicional
```

A lista de colunas a serem selecionadas pode ser substituída por “*” se todas as colunas serão selecionadas. A sintaxe mostrada acima é básica. Para selecionar todos os números e textos de registros escritos pelo autor Mefisto, pode-se fazer:

```
SELECT codigo, texto
      FROM anuncios
      WHERE autor = 'Mefisto'
```