O pacote JDBC fornece uma biblioteca padrão para acessar bancos de dados relacionais em Java. Existe também uma API, que permite o acesso a uma ampla variedade de bancos de dados que utilizam SQL com exatamente a mesma sintaxe Java.

Sete etapas para manipulação de banco de dados em Java

1 **Carregue o driver JDBC**. Para carregar um driver, você deve especificar o nome do driver do banco de dados que será utilizado no método `Class.forName`. Ao fazer isso, você cria automaticamente uma instância do driver que é registrado no gerenciador de drivers do JDBC.

2 **Defina a URL de conexão**. Em JDBC, uma URL de conexão especifica o servidor host, a porta e nome do banco de dados com que se quer estabelecer uma conexão.

3 **Estabeleça a conexão**. Com o nome de usuário, a URL e uma senha, a conexão de rede para o banco de dados pode ser estabelecida. Uma vez que a conexão é estabelecida, as consultas de banco de dados podem ser realizadas até que a conexão seja fechada.

4 **Crie um objeto do tipo `Statement`**. Criando um objeto do tipo `Statement` é possível enviar consultas e comandos para o banco de dados.

5 **Execute uma consulta ou atualização no banco de dados**. Dado um objeto do tipo `Statement`, você pode enviar instruções SQL para o banco de dados usando `execute` ou os métodos `executeQuery`, `executeUpdate` ou `executeBatch`.

6 **Processe os resultados**. Quando uma consulta de banco de dados é executada, um `ResultSet` é retornado. O `ResultSet` representa um conjunto de linhas e colunas que podem ser processados por chamadas a métodos `getXXX`.

7 **Feche a conexão**. Quando as consultas e processamento de resultados se encerrarem, a conexão deve ser fechada, liberando os recursos do banco de dados.

Carregando o driver JDBC

O driver é um software que "sabe" como falar com o banco de dados. Para definir o driver, basta carregar a classe apropriada em função do tipo do banco de dados que será utilizado. Para isso, deve-se utilizar o método `Class.forName`. Este método recebe uma string que representa um nome de uma classe totalmente qualificado (ie, que inclui nomes de pacotes) e carrega a classe correspondente. Esta chamada pode lançar a exceção **ClassNotFoundException**, portanto, deve estar dentro de um bloco `try...catch...`, conforme o exemplo abaixo:

```
try   {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      } catch(ClassNotFoundException cnfe) {
            System.err.println("Error loading driver: " + cnfe);
      }
```

Neste exemplo foi carregado o driver do banco de dados Oracle. Repare que a carga está dentro do bloco `try... catch...`

Veja na tabela abaixo, a lista dos principais drivers para acesso aos bancos de dados mais comuns:

Access	`Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
MySQL	`Class.forName("com.mysql.jdbc.Driver");`
Sybase	`Class.forName("com.sybase.jdbc2.jdbc.SybDriver");`
PostgreSQL	`Class.forName("org.postgresql.Driver");`
DB2/Neon	`Class.forName("com.neon.jdbc.Driver");`

A maioria dos fornecedores distribuem seus drivers JDBC dentro de arquivos JAR. Assim, durante o desenvolvimento deve-se lembrar de incluir o caminho para o arquivo JAR do driver na definição do CLASSPATH.

Definindo a URL de conexão

Uma vez carregado o driver JDBC, deve-se especificar o local do servidor de banco de dados. As URLs referentes a bancos de dados usam `jdbc: protocolo` e incorporam o servidor host, a porta e o nome do banco de dados dentro da URL. O formato exato é definido na documentação que vem com o driver.Veja alguns exemplos:

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String oracleURL = "jdbc:oracle:thin:@" + host +
":" + port + ":" + dbName;
String sybaseURL = "jdbc:sybase:Tds:" + host +
":" + port + ":" + "?SERVICENAME=" + dbName;
```

```
String msAccessURL = "jdbc:odbc:" + dbName;
```

Estabelecendo a conexão

Para fazer a conexão com o banco de dados deve-se informar a URL, o nome de usuário do banco de dados, e a senha de acesso ao banco de dados para o método `getConnection` da classe `DriverManager`. Note que `getConnection` pode gerar uma `SQLException`, então é necessário codificá-lo dentro de um bloco try... catch...

```
String username = "jay_debesee";
String password = "secret";
Connection connect = DriverManager.getConnection(oracleURL,username,password);
```

A classe `Connection` tem outros métodos bastante úteis, como por exemplo, os métodos:

- `close()` – encerra a conexão
- `isClosed()` – verifica se a conexão foi encerrada ou se aconteceu um "time out" no banco de dados

Criando um objeto do tipo `Statement`

Um objeto do tipo `Statement` é usado para enviar consultas e comandos para o banco de dados. Ele é criado a partir da classe `Connection`, através do método `createStatement()`.

```
Statement statement = connection.createStatement();
```

Executando uma consulta ou atualização no banco de dados

Uma vez que já tenha sido criada uma instância de `Statement`, é possível utilizá-la para enviar comandos SQL utilizando-se o método `executeQuery()`, que retorna um objeto do tipo `ResultSet`.

```
String query = "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet = statement.executeQuery(query);
```

Alguns métodos da classe `Statement` são:

- **`executeQuery`** ➔ executa uma consulta SQL e retorna os dados em um `ResultSet`. O `ResultSet` pode estar vazio, porém nunca `null`.
- **`executeUpdate`** ➔ utilizado para a execução dos comandos SQL UPDATE, INSERT ou DELETE. O método retorna o número se linhas afetadas pelo comando, que poderá ser igual a zero.
- **`setQueryTimeout`** ➔ especifica a quantidade de tempo que um driver irá esperar pelo resultado entes de lançar a exceção `SQLException`.

Processando os resultados

A forma mais simples de manipular os resultados é através da utilização do método `next` do objeto `ResultSet`. Desta forma é possível movimentar-se através da tabela uma linha de cada vez.

Dentro de uma linha, `ResultSet` fornece vários métodos `getXXX` que levam um nome de coluna ou índice de coluna como um argumento e retornam o resultado de uma variedade de diferentes tipos de Java.

Por exemplo, usar `getInt` se o valor deve ser um inteiro, `GetString` para uma String, e assim por diante para a maioria dos outros tipos de dados . Para apenas exibir os resultados, pode-se usar `getString` para a maioria das colunas.

No entanto, se for utilizada a versão do `getXXX` que manipula um índice para a coluna (ao invés de um nome de coluna), é importante observar que as colunas são indexados a partir de 1, e não em 0 como em matrizes, vetores, etc. na linguagem Java. Veja um exemplo:

```
while(resultSet.next()) {
      System.out.println(resultSet.getString(1) + " " +
            resultSet.getString(2) + " " +
            resultSet.getString("firstname") + " "
            resultSet.getString("lastname"));
}
```

Fechando a conexão

Para fechar a conexão executa-se:

```
      connection.close ();
```

Fechando a conexão também são fechados os objetos `Statement` e `ResultSet` correspondentes. Se existir interesse em realizar operações adicionais no banco de dados, recomenda-se adiar seu fechamento, já que a sobrecarga de abrir uma conexão é geralmente muito grande.