JDBC指南

极客学院出版

前言

JDBC API 是一个 Java API，它可以访问任何类型的表格数据，特别是可以访问存储在关系数据库里的数据。JDBC 可以用 Java 语言在各种平台上实现，比如 Windows 系统， Mac OS 系统，和各种版本的 UNIX 系统。

适用人群

本教程是给那些想详细了解 JDBC 框架以及想实际使用这些框架的 Jave 程序员编写的。

学习前提

在学习本教程之前，你需要掌握 Java 编程语言。当你准备使用关系数据库管理系统时，需要对 SQL 和数据库的概念有了一定的了解。

你将学会

- JDBC 框架
- JDBC 与数据库操作

更新日期	更新内容
2015-06-10	JDBC 指南

目录

前言 1

第 1 章 JDBC 指南 4

简介 5

SQL语法 8

环境 12

示例代码 17

驱动类型 20

连接数据库 24

Statement 对象 29

结果集 35

数据类型 40

事务 44

异常 47

批处理 50

存储过程 53

流数据 58

第 2 章 JDBC 示例 62

创建数据库实例 63

选择数据库实例 66

删除数据库实例 68

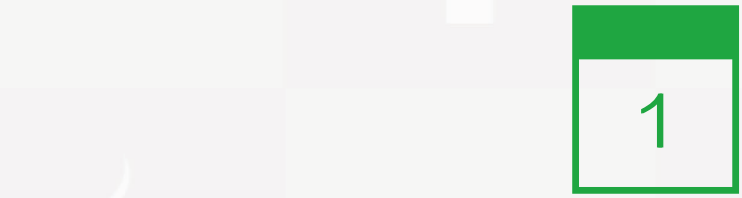
创建表实例 71

删除表实例 74

插入记录实例 77

查询记录实例 80

更新记录实例 83
删除记录实例 86
WHERE 子句实例 89
LIKE 子句实例 93
排序实例 97



1 JDBC 指南

简介

什么是 JDBC?

JDBC 指 Java 数据库连接，是一种标准Java应用编程接口（ JAVA API ），用来连接 Java 编程语言和广泛的数据库。

JDBC API 库包含下面提到的每个任务，都是与数据库相关的常用用法。

- 制作到数据库的连接。
- 创建 SQL 或 MySQL 语句。
- 执行 SQL 或 MySQL 查询数据库。
- 查看和修改所产生的记录。

从根本上来说，JDBC 是一种规范，它提供了一套完整的接口，允许便携式访问到底层数据库，因此可以用 Java 编写不同类型的可执行文件，例如：

- Java 应用程序
- Java Applets
- Java Servlets
- Java ServerPages (JSPs)
- Enterprise JavaBeans (EJBs)

所有这些不同的可执行文件就可以使用 JDBC 驱动程序来访问数据库，这样可以方便的访问数据。

JDBC 具有 ODBC 一样的性能，允许 Java 程序包含与数据库无关的代码。

先决条件

为了更好的理解本教程，需要对以下两个主题内容很好的理解：

- 核心 JAVA 编程
- SQL 或 MySQL 数据库

JDBC 架构

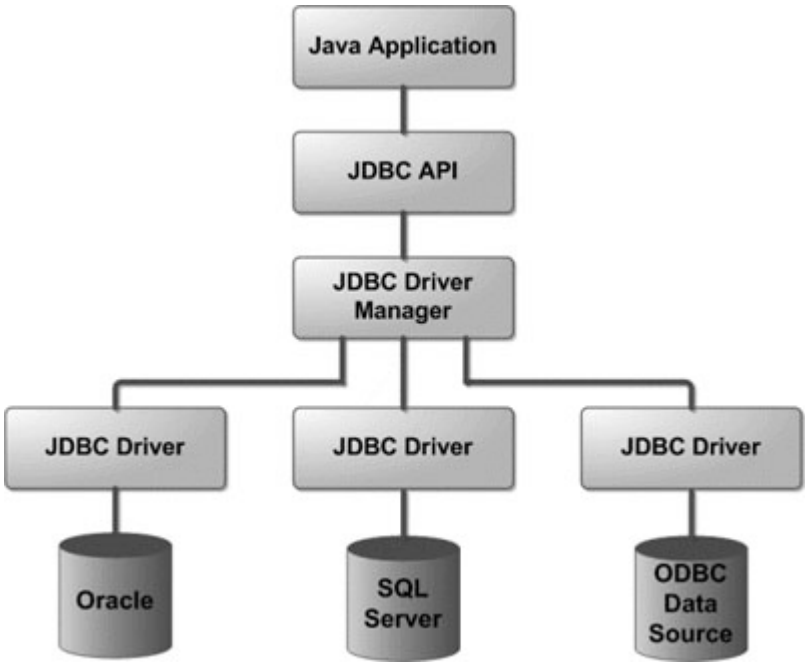
JDBC 的 API 支持两层和三层处理模式进行数据库访问，但一般的 JDBC 架构由两层处理模式组成：

- JDBC API: 提供了应用程序对 JDBC 管理器的连接。
- JDBC Driver API: 提供了 JDBC 管理器对驱动程序连接。

JDBC API 使用驱动程序管理器和数据库特定的驱动程序来提供异构（heterogeneous）数据库的透明连接。

JDBC 驱动程序管理器可确保正确的驱动程序来访问每个数据源。该驱动程序管理器能够支持连接到多个异构数据库的多个并发的驱动程序。

以下是结构图，其中显示了驱动程序管理器相对于在 JDBC 驱动程序和 Java 应用程序所处的位置。



常见的 JDBC 组件

JDBC 的 API 提供了以下接口和类：

DriverManager： 这个类管理一系列数据库驱动程序。匹配连接使用通信子协议从 JAVA 应用程序中请求合适的数据库驱动程序。识别 JDBC 下某个子协议的第一驱动程序将被用于建立数据库连接。

Driver : 这个接口处理与数据库服务器的通信。你将很少直接与驱动程序互动。相反，你使用 DriverManager 中的对象，它管理此类型的对象。它也抽象与驱动程序对象工作相关的详细信息。

Connection : 此接口具有接触数据库的所有方法。该连接对象表示通信上下文，即，所有与数据库的通信仅通过这个连接对象进行。

Statement : 使用创建于这个接口的对象将 SQL 语句提交到数据库。除了执行存储过程以外，一些派生的接口也接受参数。

ResultSet : 在你使用语句对象执行 SQL 查询后，这些对象保存从数据获得的数据。它作为一个迭代器，让您可以通过它的数据来移动。

SQLException : 这个类处理发生在数据库应用程序的任何错误。

JDBC 4.0 软件包

JDBC 4.0 主要包含 java.sql 包和 javax.sql 包，在编写这本教程的时候这是 JDBC 最新的版本。它提供的主要类与数据源进行交互。

在这些包中的新功能包括改变在以下几个方面:

- 自动数据库驱动程序加载
- 异常处理的改进
- 增强的 BLOB/CLOB 功能
- 增强的连接和语句界面
- 国家字符集支持
- SQL ROWID 访问
- SQL 2003 XML 数据类型支持
- 注释

SQL语法

结构化查询语言（SQL）是一种标准化的语言，它可以让你对数据库进行操作，如创建项目，读取内容，更新内容和删除项目。

SQL 支持你可能会使用到的任何数据库，它可以让你编写独立于底层数据库的数据库代码。

本章介绍了 SQL，这是一个了解 JDBC 概念的先决条件。在经历了这一章后，你将能够创建，读取，更新和删除（通常被称为 CRUD 操作）一个数据库中的数据。

为了详细理解 SQL，你可以阅读我们的 MySQL 教程。

创建数据库

CREATE DATABASE 语句用于创建一个新的数据库。语法是 −

```
SQL> CREATE DATABASE DATABASE_NAME;
```

示例

下面的 SQL 语句创建一个名为 EMP 的数据库 −

```
SQL> CREATE DATABASE EMP;
```

删除数据库

使用 DROP DATABASE 语句用于删除现有的数据库。语法是 −

```
SQL> DROP DATABASE DATABASE_NAME;
```

注意：要创建或删除数据库，你必须有数据库服务器的管理员权限。请注意，删除数据库会把存储在该数据库中的数据一并删除。

创建表

CREATE TABLE 语句用于创建新表。语法是 −

```
SQL> CREATE TABLE TABLE_NAME
 (
  COLUMN_NAME column_data_type，
  COLUMN_NAME column_data_type，
  COLUMN_NAME column_data_type
  ...
 ) ;
```

示例

下面的 SQL 语句创建一个含有四列名为 Employees 的表 −

```
SQL> CREATE TABLE Employees
(
  id INT NOT NULL,
  age INT NOT NULL,
  first VARCHAR(255),
  last VARCHAR(255),
  PRIMARY KEY ( id )
);
```

删除表

DROP TABLE 语句用于删除现有的表。语法是 −

```
SQL> DROP TABLE table_name;
```

示例

下面的 SQL 语句删除名为 Employees 的表 −

```
SQL> DROP TABLE Employees;
```

INSERT 数据

INSERT 的语法如下所示，其中 column1，column2 等数据出现在相应的列中 −

```
SQL> INSERT INTO table_name的VALUES ( column1，column2，... ) ;
```

示例

下面的 SQL INSERT 语句将在前面创建的 Employees 数据库中插入新的一行数据 −

```
SQL> INSERT INTO Employees VALUES（100，18，'Zara'，'Ali'）;
```

SELECT 数据

SELECT 语句用于从数据库中检索数据。SELECT 的语法 −

```
SQL> SELECT column_name, column_name, ...
    FROM table_name
    WHERE conditions;
```

WHERE 子句可以使用 =，!=，<，>，<=，>=，BETWEEN 和 LIKE 这些比较操作符。

示例

下面的 SQL 语句从 Employees 表中选出 ID 列是100的年龄、第一列、最后一列这些信息

```
SQL> SELECT first, last, age FROM Employees WHERE id = 100;
```

下面的 SQL 语句从 Employees 表中选出第一列包含 Zara 字符的年龄、第一列、最后一列这些信息

```
SQL> SELECT first, last, age FROM Employees WHERE first LIKE '%Zara%';
```

UPDATE 数据

UPDATE 语句用于更新数据。UPDATE 的语法 −

```
SQL> UPDATE table_name
    SET column_name = value, column_name = value, ...
    WHERE conditions;
```

WHERE 子句可以使用=，!=，<，>，<=，>=，BETWEEN 和 LIKE 这些比较操作符。

示例

下面的 SQL UPDATE 语句改变了 ID 是100的员工的age列的数据 −

```
SQL> UPDATE Employees SET age=20 WHERE id=100;
```

DELETE 数据

DELETE 语句用于从表中删除数据。DELETE 的语法−

```
SQL> DELETE FROM table_name WHERE conditions;
```

WHERE 子句可以使用=，!=，<，>，<=，>=，BETWEEN 和 LIKE 这些比较操作符。

示例

下面的 SQL DELETE 语句将 Employees 表中 ID 是100的记录删除−

```
SQL> DELETE FROM Employees WHERE id=100;
```

环境

在开始使用 JDBC 之前，你必须如下面所示设置你的 JDBC 环境。我们假设你正在使用的是 Windows 平台。

安装 Java

从 Java 官方网站 (http://www.oracle.com/technetwork/java/index.html) 上安装 J2SE Development Kit 6.0 (JDK 5.0)。

请按如下所述设置环境变量−

- JAVA_HOME : 该环境变量应该指向你安装的JDK目录，例如：C:\Program Files\Java\jdk1.6.0。
- CLASSPATH : 该环境变量应该有适当的路径设置，例如：C:\Program Files\Java\jdk1.6.0_20\jre\lib。
- PATH： 这个环境变量应该指向 JRE bin，例如：C:\Program Files\Java\jre1.6.0_20\bin。

可能你已经设置好这些变量了，这儿告诉你如何设置是为了确保正确。

- 进入控制面板，双击系统。如果你是 Windows XP 的用户，在打开性能和维护之前，你会看到系统的图标。
- 进入高级选项卡，然后单击环境变量。
- 现在检查所有上述变量的设置是否正确。

当你安装 J2SE Development Kit 6.0 (JDK 6.0) 的时候，会自动获得了 JDBC包 **java.sql** 和 **javax.sql** 。

安装数据库

你最重要的事情当然是安装一个实际运行的数据库，你可以在该数据库里查询和修改表。

大部分的数据库都适合你用。你可以有多种选择，最常见的是−

- **MySQL 数据库：** MySQL 是一个开源数据库。你可以从 MySQL 官方网站 (http://dev.mysql.com/downloads/mysql) 上下载。我们建议你下载完整的 Windows 版本。

此外，下载并安装 MySQL Administrator (http://dev.mysql.com/downloads/gui-tools/) 以及 MySQL Query Browser (http://dev.mysql.com/downloads/gui-tools/) 。这些都是基于 GUI 的工具，可以让你的开发变得更加容易。

最后，下载并解压缩 MySQL Connector/J (http://dev.mysql.com/downloads/connector/j/3.1.html)（MySQL JDBC驱动程序）到合适的目录。对于本教程的目的，我们假设你已经安装了驱动程序在 C:\Program Files\MySQL\mysql-connector-java-5.1.8 目录下。

因此，设置 CLASSPATH 变量为 C:\Program Files\MySQL\mysql-connector-java-5.1.8\mysql-connector-java-5.1.8-bin.jar。你的驱动程序版本可能会根据你的安装而有所不同。

- **PostgreSQL 数据库：** PostgreSQL 是一个开源数据库。你可以从 PostgreSQL 的官方网站 (http://www.postgresql.org/download/) 下载。

  Postgres 安装包包含了一个名为 pgAdmin III 基于 GUI 的管理工具。JDBC 驱动程序也是安装包的一部分。

- **Oracle 数据库：** Oracle 数据库是 Oracle 公司销售的商用数据库。我们假设你已经安装了必须的安装介质。

  Oracle 安装包包括一个名为 Enterprise Manager 基于 GUI 的管理工具。JDBC 驱动程序也是安装包的一部分。

安装数据库驱动程序

最新的 JDK 包含一个 JDBC-ODBC 桥驱动程序，程序员通过使用 JDBC API 可以操作大多数开放式数据库（ODBC）驱动程序。

现在，大部分的数据库厂商都提供与数据库安装相适应的 JDBC 驱动程序。所以，这部分你不必担心。

设置数据库认证

在本教程中，我们将使用 MySQL 数据库。当你安装上述任一数据库时，管理员 ID 设置为 root，并给出规定设置选择的密码。

使用 root 账号和密码，你可以创建另一个账号和密码，或者你可以在 JDBC 的程序中直接使用 root 账户和密码。

各种数据库的操作，比如数据库的创建和删除，都需要管理员的账户和密码。

对于 JDBC 教程的其余部分，在 MySQL 数据库中我们将使用 username 作为账户 password 作为密码。

如果你没有足够的权限创建新用户，那么你可以要求你的数据库管理员（DBA）创建一个用户账户和密码给你。

创建数据库

要创建 EMP 数据库，请使用以下步骤−

步骤 1

通过以下步骤打开一个**命令提示符**并切换到安装目录下−

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

注意： mysqld.exe 的路径会根据你系统上安装的路径而可能会有所不同。你也可以查看如何启动和停止你的数据库服务器的文档。

步骤 2

通过以下步骤来启动数据库服务器，如果该数据库服务器没有运行。

```
C:\Program Files\MySQL\bin>mysqld
C:\Program Files\MySQL\bin>
```

步骤 3

通过以下步骤来创建 EMP 数据库−

```
C:\Program Files\MySQL\bin> mysqladmin create EMP −u root −p
Enter password: ********
C:\Program Files\MySQL\bin>
```

创建表

通过以下步骤在 EMP 数据库中创建 Employees 表−

步骤 1

通过以下步骤打开一个**命令提示符**并切换到安装目录下−

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

步骤 2

通过以下步骤登录到数据库。

```
C:\Program Files\MySQL\bin>mysql −u root −p
Enter password: ********
mysql>
```

步骤 3

通过以下命令来创建 Employee 表−

```
mysql> use EMP;
mysql> create table Employees
   −> (
   −> id int not null,
   −> age int not null,
   −> first varchar (255),
   −> last varchar (255)
   −> );
Query OK, 0 rows affected (0.08 sec)
mysql>
```

创建数据记录

最后，通过以下步骤你可以在 Employee 表中创建几条记录−

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)

mysql>
```

通过研究学习 MySQL 教程，可以完整的了解 MySQL 数据库。

现在，你就可以尝试使用 JDBC。下一章将为你提供了一个简单的 JDBC 编程示例。

示例代码

本章提供了如何创建一个简单 JDBC 应用程序的示例。这个示例演示如何打开一个数据库连接，执行 SQL 查询，并显示结果。

所有在这个模版示例中提到的步骤，将在本教程的后续章节中进行详细描述。

创建 JDBC 应用程序

构建一个 JDBC 应用程序包括以下六个步骤−

- **导入数据包：**需要你导入含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动器：**需要你初始化一个驱动器，以便于你打开一个与数据库的通信通道。
- **打开连接：**需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库的物理连接。
- **执行查询：**需要使用类型声明的对象建立并提交一个 SQL 语句到数据库。
- **提取结果数据：**要求使用适当的 *ResultSet.getXXX()* 方法从结果集中检索数据。
- **清理环境：**依靠 JVM 的垃圾收集来关闭所有需要明确关闭的数据库资源。

示例代码

当你在未来需要创建自己的 JDBC 应用程序时，本示例可以作为一个**模板**。

在前面的章节里，基于对环境和数据库安装的示例代码已经写过。

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class FirstExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/EMP";
```

```
// Database credentials
static final String USER = "username";
static final String PASS = "password";

public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
  //STEP 2: Register JDBC driver
  Class.forName("com.mysql.jdbc.Driver");

  //STEP 3: Open a connection
  System.out.println("Connecting to database...");
  conn = DriverManager.getConnection(DB_URL,USER,PASS);

  //STEP 4: Execute a query
  System.out.println("Creating statement...");
  stmt = conn.createStatement();
  String sql;
  sql = "SELECT id, first, last, age FROM Employees";
  ResultSet rs = stmt.executeQuery(sql);

  //STEP 5: Extract data from result set
  while(rs.next()){
    //Retrieve by column name
    int id  = rs.getInt("id");
    int age = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
  }
  //STEP 6: Clean-up environment
  rs.close();
  stmt.close();
  conn.close();
}catch(SQLException se){
  //Handle errors for JDBC
  se.printStackTrace();
}catch(Exception e){
```

```
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end FirstExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java FirstExample
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
C:\>
```

驱动类型

什么是 JDBC 驱动程序?

JDBC 驱动实现了 JDBC API 中定义的接口，该接口用于与数据库服务器进行交互。

例如，使用 JDBC 驱动程序可以让你打开数据库连接，并通过发送 SQL 或数据库命令，然后通过 Java 接收结果。

java.sql 包中附带的 JDK，包含了定义各种类与他们的行为和实际实现，这些类都在第三方驱动程序中完成。第三方供应商在他们的数据库驱动程序中都实现了 *java.sql.Driver* 接口。

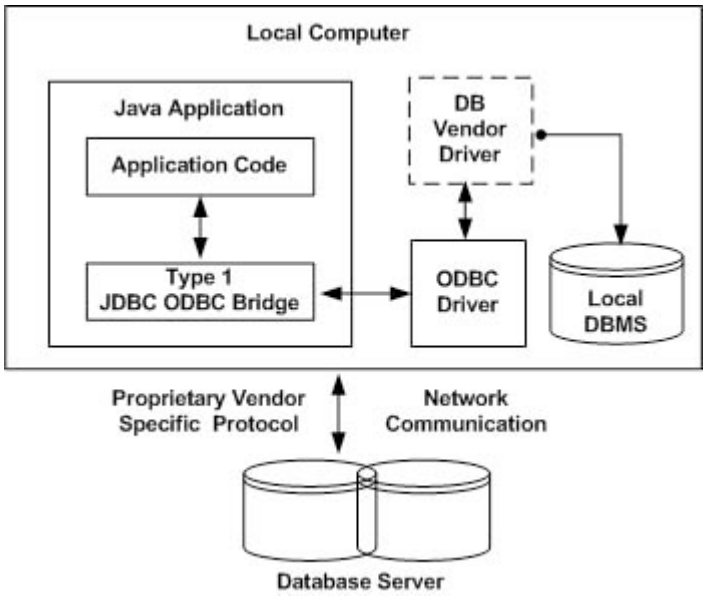
JDBC 驱动程序类型

JDBC 驱动程序的实现，因为各种各样的操作系统和 Java 运行在硬件平台的不同而不同。Sun 公司将实现类型分为四类：类型1，2，3，4，其解释如下-

类型1：JDBC-ODBC 桥驱动程序：

在类型1驱动程序中，一个 JDBC 桥接器是用来访问安装在每个客户机上的 ODBC 驱动程序。为了使用 ODBC，需要在目标数据库上配置系统数据源名称（DSN）。

当 Java 刚出来时，这是一个很有用的驱动程序，因为大多数的数据库只支持 ODBC 访问，但现在此类型的驱动程序仅适用于实验用途或在没有其他选择的情况。

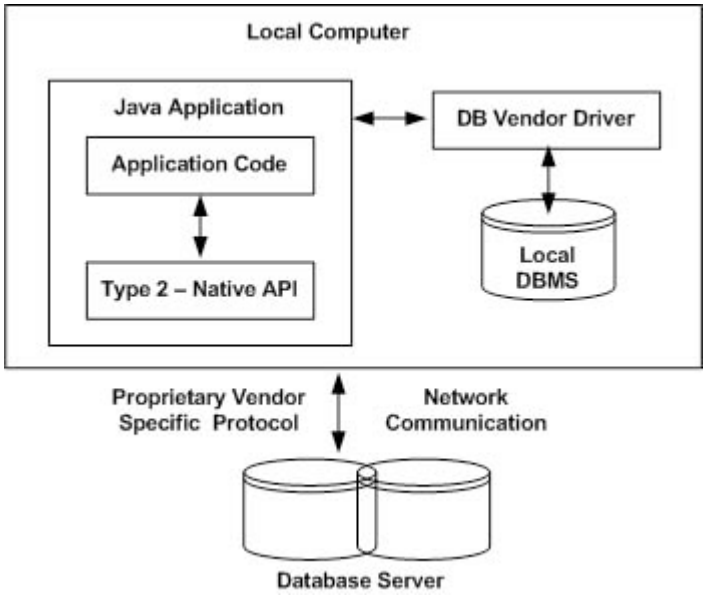


自带 JDK 1.2 中的 JDBC-ODBC 桥是这类驱动程序的一个很好的例子。

类型2：JDBC-Native API

在类型2驱动程序中，JDBC API 调用转换成原生的 C/C++ API 调用，这对于数据库来说具有唯一性。这些驱动程序通常由数据库供应商提供，并和 JDBC-ODBC 桥驱动程序同样的方式使用。该供应商的驱动程序必须安装在每台客户机上。

如果我们改变了当前数据库，我们必须改变原生 API ，因为它是具体到某一个数据库，并且他们大多已经失效了。即使这样用类型2驱动程序也能提高一些速度，因为他消除了 ODBC 的开销。

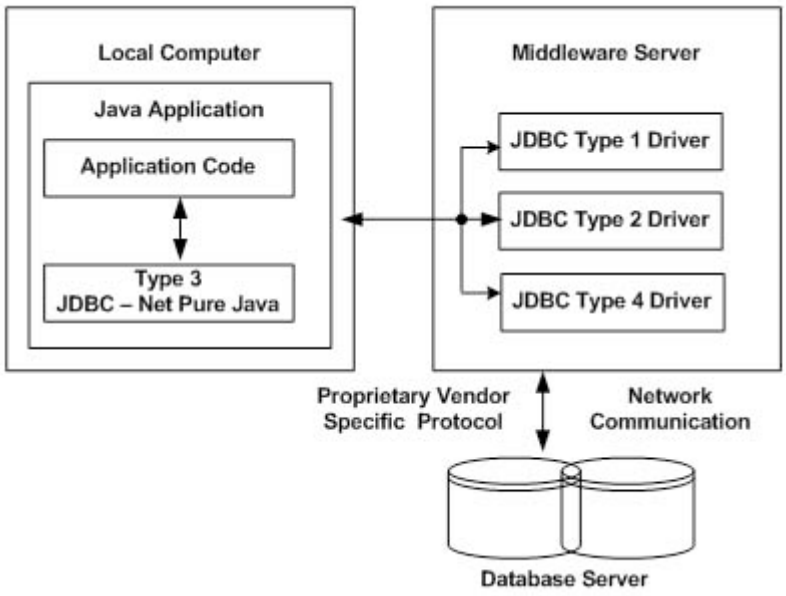


Oracle 调用接口（ OCI ）驱动程序是一个类型2驱动程序的示例。

类型3：JDBC-Net 纯 Java

在类型3驱动程序中，一般用三层方法来访问数据库。JDBC 客户端使用标准的网络套接字与中间件应用服务器进行通信。套接字的相关信息被中间件应用服务器转换为数据库管理系统所要求的的调用格式，并转发到数据库服务器。

这种驱动程序是非常灵活的，因为它不需要在客户端上安装代码，而且单个驱动程序能提供访问多个数据库。



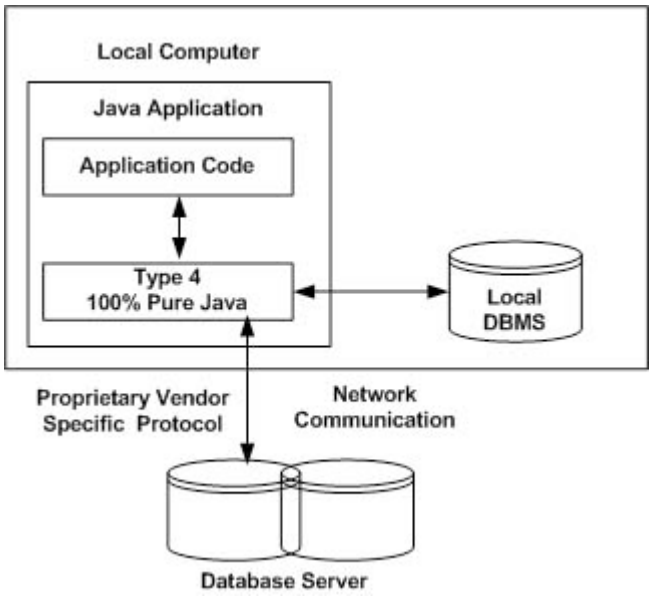
你可以将应用服务器作为一个 JDBC “代理”，这意味着它会调用客户端应用程序。因此，你需要一些有关服务器配置方面的知识，这样可以高效地使用此驱动程序类型。

你的应用服务器可能使用类型1，2，或4驱动程序与数据库进行通信，了解它们的细微之处将会很有帮助。

类型4：100%纯 Java

在类型4驱动程序中，一个纯粹的基于 Java 的驱动程序通过 socket 连接与供应商的数据库进行通信。这是可用于数据库的最高性能的驱动程序，并且通常由供应商自身提供。

这种驱动器是非常灵活的，你不需要在客户端或服务端上安装特殊的软件。此外，这些驱动程序是可以动态下载的。



MySQL Connector/J 的驱动程序是一个类型4驱动程序。因为它们的网络协议的专有属性，数据库供应商通常提供类型4的驱动程序。

该使用哪种驱动程序?

如果你正在访问一个数据库，如 Oracle，Sybase 或 IBM，首选的驱动程序是类型4。

如果你的 Java 应用程序同时访问多个数据库类型，类型3是首选的驱动程序。

类型2驱动程序是在你的数据库没有提供类型3或类型4驱动程序时使用的。

类型1驱动程序不被认为是部署级的驱动程序，它存在的目的通常仅用于开发和测试。

连接数据库

在你安装相应的驱动程序后，就可以用 JDBC 建立一个数据库连接。

编写建立一个 JDBC 连接的程序是相当简单的。下面是简单的四个步骤−

- **导入 JDBC 包**：在你的 Java 代码中，用 import 语句添加你所需的类。
- **注册 JDBC 驱动程序**：这一步会导致 JVM 加载所需的驱动程序到内存中执行，因此它可以实现你的 JDBC 请求。
- **数据库 URL 制定**：这是用来创建格式正确的地址指向你想要连接的数据库。
- **创建连接对象**：最后，代码调用 *DriverManager* 对象的 *getConnection()* 方法来建立实际的数据库连接。

导入 JDBC 包

import 语句告诉 Java 编译器在哪里可以找到你在代码中引用的类，这些引用放置在你的源代码起始位置。

使用标准的 JDBC 包，它允许你选择，插入，更新和删除 SQL 表中的数据，添加以下引用到您的源代码中−

```
import java.sql.* ;  // for standard JDBC programs
import java.math.* ; // for BigDecimal and BigInteger
```

注册 JDBC 驱动程序

在使用驱动程序之前，你必须在你的程序里面注册它。通过加载 Oracle 驱动程序的类文件到内存中来注册驱动程序，因此它可以采用 JDBC 接口来实现。

你需要在你的程序里做一次注册即可。你可以通过以下两种方式来注册一个驱动程序。

方法1 − Class.forName()

注册一个驱动程序中最常用的方法是使用 Java 的Class.forName() 方法来动态加载驱动程序的类文件到内存中，它会自动将其注册。这种方法更优越一些，因为它允许你对驱动程序的注册信息进行配置，便于移植。

下面是使用 Class.forName() 来注册 Oracle 驱动程序的示例：

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

你可以使用 getInstance() 方法来解决不兼容的 JVM，但你必须编写如下所示的两个额外的异常−

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
catch(IllegalAccessException ex) {
   System.out.println("Error: access problem while loading!");
   System.exit(2);
catch(InstantiationException ex) {
   System.out.println("Error: unable to instantiate driver!");
   System.exit(3);
}
```

方法2 − DriverManager.registerDriver()

你注册一个驱动程序的第二种方法是使用静态 staticDriverManager.registerDriver() 方法。

如果你使用的是不兼容 JVM 的非 JDK，比如微软提供的，你必须使用 *registerDriver()* 方法。

下面是使用 registerDriver() 来注册 Oracle 驱动程序的示例：

```
try {
   Driver myDriver = new oracle.jdbc.driver.OracleDriver();
   DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

数据库 URL 制定

当你加载了驱动程序之后，你可以通过 DriverManager.getConnection() 方法建立一个连接。为方便参考，以下列出了三个加载 DriverManager.getConnection() 方法：

- getConnection(String url)
- getConnection(String url, Properties prop)
- getConnection(String url, String user, String password)

在这里，每个格式需要一个数据库 URL ，数据库 URL 是指向数据库的地址。

在建立一个数据连接的时候，大多数会在配置一个数据库 URL 时遇到问题。

下表列出了常用的 JDBC 驱动程序名和数据库URL。

RDBMS	JDBC 驱动程序名称	URL 格式
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:hostname: port Number/databaseName

URL 格式所有加粗的部分都是静态的，你需要将剩余部分按照你的数据库实际情况进行设置。

创建连接对象

我们已经列出了三种用 DriverManager.getConnection() 方法来创建一个连接对象。

使用数据库 URL 的用户名和密码

getConnection() 最常用的方式是需要你提供一个数据库 URL，用户名和密码：

假设你使用的是 Oracle 的简化驱动程序，你可以从 URL 获得数据库的主机名：端口：数据库名称的信息。

如果你有一台名为 amrood 的主机，它的 TCP / IP 地址 192.0.0.1，你的 Oracle 监听器被配置为监听端口 1521，数据库名称是 EMP，然后完整的数据库 URL 是–

```
jdbc:oracle:thin:@amrood:1521:EMP
```

现在，你必须调用适当的用户名和密码以及 getConnection() 方法来获得一个 Connection 对象，如下所示：

```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
String USER = "username";
String PASS = "password"
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

只使用数据库 URL

第二种 DriverManager.getConnection() 方法调用的方式只需要数据库 URL 参数−

```
DriverManager.getConnection(String url);
```

然而，在这种情况下，数据库的 URL ，包括用户名和密码，将表现为以下的格式−

```
jdbc:oracle:driver:username/password@database
```

所以上述连接对象可以如下所示创建连接−

```
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
Connection conn = DriverManager.getConnection(URL);
```

使用数据库 URL 和 Properties 对象

第三种 DriverManager.getConnection() 方法调用需要数据库 URL 和 Properties 对象−

```
DriverManager.getConnection(String url, Properties info);
```

Properties 对象保存了一组关键数值。它通过调用 getConnection() 方法，将驱动程序属性传递给驱动程序。

使用下面的代码可以建立与上述示例相同的连接−

```
import java.util.*;

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
Properties info = new Properties( );
info.put( "user", "username" );
info.put( "password", "password" );

Connection conn = DriverManager.getConnection(URL, info);
```

关闭 JDBC 连接

在 JDBC 程序的末尾，它必须明确关闭所有的连接到数据库的连接，以结束每个数据库会话。但是，如果忘了，Java 垃圾收集器也会关闭连接，它会完全清除过期的对象。

依托垃圾收集器，特别是在数据库编程，是非常差的编程习惯。你应该养成用 close()方法关闭连接对象的习惯。

为了确保连接被关闭，你可以在代码中的 'finally' 程序块中执行。 无论异常是否发生，*finally* 程序是肯定会被执行的。

要关闭上面打开的连接，你应该调用 close()方法，如下所示−

```
conn.close();
```

对你的数据库管理员来说，明确的关闭连接到 DBMS 的连接，是相当开心的事。

Statement 对象

一旦我们获得了数据库的连接，我们就可以和数据库进行交互。JDBC 的 Statement，CallableStatement 和 PreparedStatement 接口定义的方法和属性，可以让你发送 SQL 命令或 PL/SQL 命令到数据库，并从你的数据库接收数据。

在数据库中，它们还定义了帮助 Java 和 SQL 数据类型之间转换数据差异的方法。

下表提供了每个接口的用途概要，根据实际目的决定使用哪个接口。

接口	推荐使用
Statement	可以正常访问数据库，适用于运行静态 SQL 语句。 Statement 接口不接受参数。
PreparedStatement	计划多次使用 SQL 语句， PreparedStatement 接口运行时接受输入的参数。
CallableStatement	适用于当你要访问数据库存储过程的时候， CallableStatement 接口运行时也接受输入的参数。

Statement 对象

创建 Statement 对象

在你准备使用 Statement 对象执行 SQL 语句之前，你需要使用 Connection 对象的 createStatement() 方法创建一个，如下面的示例所示−

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

当你创建了一个 Statement 对象之后，你可以用它的三个执行方法的任一方法来执行 SQL 语句。

- **boolean execute(String SQL) :** 如果 ResultSet 对象可以被检索，则返回的布尔值为 true ，否则返回 false 。当你需要使用真正的动态 SQL 时，可以使用这个方法来执行 SQL DDL 语句。

- int executeUpdate(String SQL) : 返回执行 SQL 语句影响的行的数目。使用该方法来执行 SQL 语句，是希望得到一些受影响的行的数目，例如，INSERT，UPDATE 或 DELETE 语句。
- ResultSet executeQuery(String SQL) : 返回一个 ResultSet 对象。当你希望得到一个结果集时使用该方法，就像你使用一个 SELECT 语句。

关闭 Statement 对象

正如你关闭一个 Connection 对象来节约数据库资源，出于同样的原因你也应该关闭 Statement 对象。

简单的调用 close() 方法就可以完成这项工作。如果你关闭了 Connection 对象，那么它也会关闭 Statement 对象。然而，你应该始终明确关闭 Statement 对象，以确保真正的清除。

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   stmt.close();
}
```

为了更好地理解，我们建议你研究学习 Statement 示例教程 (http://www.tutorialspoint.com/jdbc/statement-object-example.htm) 。

PreparedStatement 对象

PreparedStatement 接口扩展了 Statement 接口，它让你用一个常用的 Statement 对象增加几个高级功能。

这个 statement 对象可以提供灵活多变的动态参数。

创建 PreparedStatement 对象

```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
```

```
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .
}
```

JDBC 中所有的参数都被用 ? 符号表示，这是已知的参数标记。在执行 SQL 语句之前，你必须赋予每一个参数确切的数值。

setXXX() 方法将值绑定到参数，其中 XXX 表示你希望绑定到输入参数的 Java 数据类型。如果你忘了赋予值，你将收到一个 SQLException。

每个参数标记映射它的序号位置。第一标记表示位置 1 ，下一个位置为 2 等等。这种方法不同于 Java 数组索引，它是从 0 开始的。

所有的 Statement对象 的方法都与数据库交互，(a) execute()，(b) executeQuery()，及 (c) executeUpdate() 也能被 PreparedStatement 对象引用。然而，这些方法被 SQL 语句修改后是可以输入参数的。

关闭 PreparedStatement 对象

正如你关闭一个 Statement 对象，出于同样的原因，你也应该关闭 PreparedStatement 对象。

简单的调用 close() 方法可以完成这项工作。如果你关闭了 Connection 对象，那么它也会关闭 PreparedStatement 对象。然而，你应该始终明确关闭 PreparedStatement 对象，以确保真正的清除。

```
PreparedStatement pstmt = null;
try {
  String SQL = "Update Employees SET age = ? WHERE id = ?";
  pstmt = conn.prepareStatement(SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  pstmt.close();
}
```

CallableStatement 对象

正如一个 Connection 对象可以创建 Statement 对象和 PreparedStatement 对象，它也可以创建被用来执行调用数据库存储过程的 CallableStatement 对象。

创建 CallableStatement 对象

假如你需要执行以下的 Oracle 存储过程−

```
CREATE OR REPLACE PROCEDURE getEmpName
  (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END;
```

注意：上面的存储过程已经写入到 Oracle 数据库中，但我们正在使用 MySQL 数据库，那么我们可以在 MySQL 的 EMP 数据库中创建相同的存储过程。

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
  (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END $$

DELIMITER ;
```

三种类型的参数有：IN，OUT 和 INOUT。PreparedStatement 对象只使用 IN 参数。CallableStatement 对象可以使用所有的三个参数。

这里是每个参数的定义−

参数	描述
IN	在 SQL 语句创建的时候该参数是未知的。你可以用 setXXX() 方法将值绑定到IN参数中。

OUT	该参数由 SQL 语句的返回值提供。你可以用 getXXX() 方法获取 OUT 参数的值。
INOUT	该参数同时提供输入输出的值。你可以用 setXXX() 方法将值绑定参数，并且用 getXXX() 方法获取值。

下面的代码片段展示了基于存储过程如何使用 Connection.prepareCall() 方法来实例化 CallableStatement 对象。

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

SQL 的 String 变量使用参数占位符表示存储过程。

使用 CallableStatement 对象就像使用 PreparedStatement 对象。你必须在执行该语句之前将值绑定到所有的参数，否则你将收到一个 SQL 异常。

如果你有 IN 参数，只要使用适用于 PreparedStatement 对象相同的规则和技巧；使用 setXXX() 方法绑定对应的 Java 数据类型。

当你使用 OUT 和 INOUT 参数时，你就必须使用额外的 CallableStatement 方法 – registerOutParameter()。 registerOutParameter() 方法绑定 JDBC 数据类型，该数据是存储过程返回的值。

一旦你调用存储过程，你可以用适当的 getXXX() 方法来获取 OUT 参数的值。这个方法将检索到的 SQL 类型映射成 Java 数据类型。

关闭 CallableStatement 对象

正如你关闭其它的 Statement 对象，出于同样的原因，你也应该关闭 PreparedStatement 对象。

简单的调用 close() 方法可以完成这项工作。如果你关闭了 Connection 对象，那么它也会关闭 CallableStatement 对象。然而，你应该始终明确关闭 CallableStatement 对象，以确保真正的清除。

```
CallableStatement cstmt = null;
try {
```

```
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   cstmt.close();
}
```

结果集

SQL 语句从数据库查询中获取数据，并将数据返回到结果集中。SELECT 语句是一种标准的方法，它从一个数据库中选择行记录，并显示在一个结果集中。 *java.sql.ResultSet* 接口表示一个数据库查询的结果集。

一个 ResultSet 对象控制一个光标指向当前行的结果集。术语“结果集”是指包含在 ResultSet 对象中的行和列的数据。

ResultSet 接口的方法可细分为三类−

- **导航方法**：用于移动光标。
- **获取方法**：用于查看当前行被光标所指向的列中的数据。
- **更新方法**：用于更新当前行的列中的数据。这些更新也会更新数据库中的数据。

光标的移动基于 ResultSet 的属性。用相应的语句生成 ResultSet 对象时，同时生成 ResultSet 的属性。

JDBC 提供了连接方法通过下列创建语句来生成你所需的 ResultSet 对象：

- createStatement(int RSType, int RSConcurrency);
- prepareStatement(String SQL, int RSType, int RSConcurrency);
- prepareCall(String sql, int RSType, int RSConcurrency);

第一个参数表示 ResultSet 对象的类型，第二个参数是两个 ResultSet 常量之一，该常量用于判断该结果集是只读的还是可修改的。

ResultSet 的类型

可能的 RSType 如下所示。如果你不指定 ResultSet 类型，将自动获得的值是 TYPE_FORWARD_ONLY。

类型	描述
ResultSet.TYPE_FORWARD_ONLY	光标只能在结果集中向前移动。
ResultSet.TYPE_SCROLL_INSENSITIVE	光标可以向前和向后移动。当结果集创建后，其他人对数据库的操作不会影响结果集的数据。
ResultSet.TYPE_SCROLL_SENSITIVE.	光标可以向前和向后移动。当结果集创建后，其他人对数据库的操作会影响结果集的数据。

ResultSet 的并发性

RSConcurrency 的值如下所示，如果你不指定并发类型，将自动获得的值是 CONCUR_READ_ONLY。

并发性	描述
ResultSet.CONCUR_READ_ONLY	创建一个只读结果集，这是默认的值。
ResultSet.CONCUR_UPDATABLE	创建一个可修改的结果集。

到目前为止我们的示例可以如下所示，可以写成初始化一个 Statement 对象来创建一个只能前进，而且只读的 ResultSet 对象−

```
try {
   Statement stmt = conn.createStatement(
                        ResultSet.TYPE_FORWARD_ONLY,
                        ResultSet.CONCUR_READ_ONLY);
}
catch(Exception ex) {
   ....
}
finally {
   ....
}
```

导航结果集

在 ResultSet 接口中包括如下几种方法涉及移动光标−

S.N.	方法 & 描述
1	public void beforeFirst() throws SQLException 将光标移动到第一行之前。
2	public void afterLast() throws SQLException 将光标移动到最后一行之后。
3	public boolean first() throws SQLException 将光标移动到第一行。
4	public void last() throws SQLException

	将光标移动到最后一行。
5	public boolean absolute(int row) throws SQLException 将光标移动到指定的第 row 行。
6	public boolean relative(int row) throws SQLException 将光标移动到当前指向的位置往前或往后第 row 行的位置。
7	public boolean previous() throws SQLException 将光标移动到上一行，如果超过结果集的范围则返回 false。
8	public boolean next() throws SQLException 将光标移动到下一行，如果是结果集的最后一行则返回 false。
9	public int getRow() throws SQLException 返回当前光标指向的行数的值。
10	public void moveToInsertRow() throws SQLException 将光标移动到结果集中指定的行，可以在数据库中插入新的一行。当前光标位置将被记住。
11	public void moveToCurrentRow() throws SQLException 如果光标处于插入行，则将光标返回到当前行，其他情况下，这个方法不执行任何操作。

为了更好地理解，让我们研究学习**导航示例代码** (http://www.tutorialspoint.com/jdbc/navigate-result-sets.htm) 。

查看结果集

ResultSet接口中含有几十种从当前行获取数据的方法。

每个可能的数据类型都有一个 get 方法，并且每个 get 方法有两个版本-

- 一个需要列名。

- 一个需要列的索引。

例如，如果你想查看的列包含一个 int 类型，你需要在 ResultSet 中调用 getInt()方法−

S.N.	方法 & 描述
1	public int getInt(String columnName) throws SQLException 返回当前行中名为 columnName 的列的 int 值。
2	public int getInt(int columnIndex) throws SQLException 返回当前行中指定列的索引的 int 值。列索引从 1 开始，意味着行中的第一列是 1 ，第二列是 2 ，以此类推。

同样的，在 ResultSet 接口中还有获取八个 Java 原始类型的 get 方法，以及常见的类型，比如 java.lang.String，java.lang.Object 和 java.net.URL。

也有用于获取 SQL 数据类型 java.sql.Date， java.sql.Time， java.sql.Timestamp， java.sql.Clob，java.sql.Blob 中的方法。查看文档可以了解使用这些 SQL 数据类型的更多的信息。

更新的结果集

ResultSet 接口包含了一系列的更新方法，该方法用于更新结果集中的数据。

用 get 方法可以有两个更新方法来更新任一数据类型−

- 一个需要列名。
- 一个需要列的索引。

例如，要更新一个结果集的当前行的 String 列，你可以使用任一如下所示的 updateString()方法−

S.N.	方法 & 描述
1	public void updateString(int columnIndex, String s) throws SQLException 将指定列的字符串的值改为 s。
2	public void updateString(String columnName, String s) throws SQLException 类似于前面的方法，不同之处在于指定的列是用名字来指定的，而不是它的索引。

八个原始数据类型都有其更新方法，比如 String，Object，URL，和在 java.sql 包中的 SQL 数据类型。

更新结果集中的行将改变当前行的列中的 ResultSet 对象，而不是基础数据库中的数据。要更新数据库中一行的数据，你需要调用以下的任一方法−

S.N.	方法 & 描述
1	public void updateRow() 通过更新数据库中相对应的行来更新当前行。
2	public void deleteRow() 从数据库中删除当前行。
3	public void refreshRow() 在结果集中刷新数据，以反映数据库中最新的数据变化。
4	public void cancelRowUpdates() 取消对当前行的任何修改。
5	public void insertRow() 在数据库中插入一行。本方法只有在光标指向插入行的时候才能被调用。

数据类型

JDBC 驱动程序在将 Java 数据类型发送到数据库之前，会将其转换为相应的 JDBC 类型。对于大多数数据类型都采用了默认的映射关系。例如，一个 Java int 数据类型转换为 SQL INTEGER。通过默认的映射关系来提供驱动程序之间的一致性。

当你调用 PreparedStatement 中的 setXXX()方法或 CallableStatement 对象或 ResultSet.updateXXX()方法时， Java 数据类型会转换为默认的 JDBC 数据类型，如下表概述。

SQL	JDBC/Java	setXXX	updateXXX
VARCHAR	java.lang.String	setString	updateString
CHAR	java.lang.String	setString	updateString
LONGVARCHAR	java.lang.String	setString	updateString
BIT	boolean	setBoolean	updateBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	updateBigDecimal
TINYINT	byte	setByte	updateByte
SMALLINT	short	setShort	updateShort
INTEGER	int	setInt	updateInt
BIGINT	long	setLong	updateLong
REAL	float	setFloat	updateFloat
FLOAT	float	setFloat	updateFloat
DOUBLE	double	setDouble	updateDouble
VARBINARY	byte[ ]	setBytes	updateBytes
BINARY	byte[ ]	setBytes	updateBytes
DATE	java.sql.Date	setDate	updateDate
TIME	java.sql.Time	setTime	updateTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	updateTimestamp
CLOB	java.sql.Clob	setClob	updateClob
BLOB	java.sql.Blob	setBlob	updateBlob
ARRAY	java.sql.Array	setARRAY	updateARRAY
REF	java.sql.Ref	SetRef	updateRef
STRUCT	java.sql.Struct	SetStruct	updateStruct

JDBC 3.0 增强了对 BLOB，CLOB，ARRAY 和 REF 数据类型的支持。 ResultSet 对象现在有 UPDATEBLOB()，updateCLOB()， updateArray()，和 updateRef()方法，通过这些方法你可以直接操作服务器上的相应数据。

你能用 setXXX()方法和 updateXXX()方法将 Java 类型转换为特定的 JDBC 数据类型。你能用 setObject()方法和 updateObject()方法将绝大部分的 Java 类型映射到 JDBC 数据类型。

ResultSet 对象为任一数据类型提供相应的 getXXX()方法，该方法可以获取任一数据类型的列值。上述任一方法的使用需要列名或它的顺序位置。

SQL	JDBC/Java	setXXX	getXXX
VARCHAR	java.lang.String	setString	getString
CHAR	java.lang.String	setString	getString
LONGVARCHAR	java.lang.String	setString	getString
BIT	boolean	setBoolean	getBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	getBigDecimal
TINYINT	byte	setByte	getByte
SMALLINT	short	setShort	getShort
INTEGER	int	setInt	getInt
BIGINT	long	setLong	getLong
REAL	float	setFloat	getFloat
FLOAT	float	setFloat	getFloat
DOUBLE	double	setDouble	getDouble
VARBINARY	byte[ ]	setBytes	getBytes
BINARY	byte[ ]	setBytes	getBytes
DATE	java.sql.Date	setDate	getDate
TIME	java.sql.Time	setTime	getTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	getTimestamp
CLOB	java.sql.Clob	setClob	getClob
BLOB	java.sql.Blob	setBlob	getBlob
ARRAY	java.sql.Array	setARRAY	getARRAY
REF	java.sql.Ref	SetRef	getRef
STRUCT	java.sql.Struct	SetStruct	getStruct

日期和时间数据类型

java.sql.Date 类映射 SQL DATE 类型，java.sql.Time 类和 java.sql.Timestamp 类也分别映射 SQL TIME 数据类型和 SQL TIMESTAMP 数据类型。

以下示例显示了日期和时间类如何转换成标准的 Java 日期和时间值，并匹配成 SQL 数据类型所要求的格式。

```
import java.sql.Date;
import java.sql.Time;
import java.sql.Timestamp;
import java.util.*;

public class SqlDateTime {
  public static void main(String[] args) {
    //Get standard date and time
    java.util.Date javaDate = new java.util.Date();
    long javaTime = javaDate.getTime();
    System.out.println("The Java Date is:" +
        javaDate.toString());

    //Get and display SQL DATE
    java.sql.Date sqlDate = new java.sql.Date(javaTime);
    System.out.println("The SQL DATE is: " +
        sqlDate.toString());

    //Get and display SQL TIME
    java.sql.Time sqlTime = new java.sql.Time(javaTime);
    System.out.println("The SQL TIME is: " +
        sqlTime.toString());
    //Get and display SQL TIMESTAMP
    java.sql.Timestamp sqlTimestamp =
    new java.sql.Timestamp(javaTime);
    System.out.println("The SQL TIMESTAMP is: " +
        sqlTimestamp.toString());
   }//end main
}//end SqlDateTime
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java SqlDateTime
The Java Date is:Tue Aug 18 13:46:02 GMT+04:00 2009
The SQL DATE is: 2009-08-18
The SQL TIME is: 13:46:02
The SQL TIMESTAMP is: 2009-08-18 13:46:02.828
C:\>
```

处理 NULL 值

SQL 使用 NULL 值和 Java 使用 null 是不同的概念。那么，你可以使用三种策略来处理 Java 中的 SQL NULL 值−

- 避免使用返回原始数据类型的 getXXX()方法。
- 使用包装类的基本数据类型，并使用 ResultSet 对象的 wasNull()方法来测试收到 getXXX()方法返回的值是否为 null，如果是 null，该包装类变量则被设置为 null。
- 使用原始数据类型和 ResultSet 对象的 wasNull()方法来测试通过 getXXX()方法返回的值，如果是 null，则原始变量应设置为可接受的值来代表 NULL。

下面是一个处理 NULL 值的示例−

```
Statement stmt = conn.createStatement( );
String sql = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);

int id = rs.getInt(1);
if( rs.wasNull( ) ) {
   id = 0;
}
```

事务

如果你的 JDBC 连接是处于自动提交模式下，该模式为默认模式，那么每句 SQL 语句都是在其完成时提交到数据库。

对简单的应用程序来说这种模式相当好，但有三个原因你可能想关闭自动提交模式，并管理你自己的事务−

- 为了提高性能
- 为了保持业务流程的完整性
- 使用分布式事务

你可以通过事务在任意时间来控制以及更改应用到数据库。它把单个 SQL 语句或一组 SQL 语句作为一个逻辑单元，如果其中任一语句失败，则整个事务失败。

若要启用手动事务模式来代替 JDBC 驱动程序默认使用的自动提交模式的话，使用 Connection 对象的的 setAutoCommit()方法。如果传递一个布尔值 false 到 setAutoCommit()方法，你就关闭自动提交模式。你也可以传递一个布尔值 true 将其再次打开。

例如，如果有一个名为 conn 的 Connection 对象，以下的代码将关闭自动提交模式−

```
conn.setAutoCommit(false);
```

提交和回滚

当你完成了你的修改，并且要提交你的修改，可以在 connection 对象里调用 commit ( ) 方法，如下所示−

```
conn.commit( );
```

另外，用名为 conn 的连接回滚数据到数据库，使用如下所示的代码−

```
conn.rollback( );
```

下面的例子说明了如何使用提交和回滚对象−

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   String SQL = "INSERT INTO Employees  " +
```

```
            "VALUES (106, 20, 'Rita', 'Tez')";
  stmt.executeUpdate(SQL);
  //Submit a malformed SQL statement that breaks
  String SQL = "INSERTED IN Employees  " +
            "VALUES (107, 22, 'Sita', 'Singh')";
  stmt.executeUpdate(SQL);
  // If there is no error.
  conn.commit();
}catch(SQLException se){
  // If there is any error.
  conn.rollback();
}
```

在这种情况下，之前的 INSERT 语句不会成功，一切都将被回滚到最初状态。

使用还原点

新的 JDBC 3.0 还原点接口提供了额外的事务控制。大部分现代的数据库管理系统的环境都支持设定还原点，例如 Oracle 的 PL/SQL。

当你在事务中设置一个还原点来定义一个逻辑回滚点。如果在一个还原点之后发生错误，那么可以使用 rollback 方法来撤消所有的修改或在该还原点之后所做的修改。

Connection 对象有两个新的方法来管理还原点−

- **setSavepoint(String savepointName):** 定义了一个新的还原点。它也返回一个 Savepoint 对象。
- **releaseSavepoint(Savepoint savepointName):** 删除一个还原点。请注意，它需要一个作为参数的 Savepoint 对象。这个对象通常是由 setSavepoint() 方法生成的一个还原点。

有一个 **rollback (String savepointName)** 方法，该方法可以回滚到指定的还原点。

下面的例子说明了如何使用 Savepoint 对象−

```
try{
  //Assume a valid connection object conn
  conn.setAutoCommit(false);
  Statement stmt = conn.createStatement();

  //set a Savepoint
  Savepoint savepoint1 = conn.setSavepoint("Savepoint1");
  String SQL = "INSERT INTO Employees " +
            "VALUES (106, 20, 'Rita', 'Tez')";
  stmt.executeUpdate(SQL);
```

```
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees " +
                "VALUES (107, 22, 'Sita', 'Tez')";
   stmt.executeUpdate(SQL);
   // If there is no error, commit the changes.
   conn.commit();

}catch(SQLException se){
   // If there is any error.
   conn.rollback(savepoint1);
}
```

在这种情况下，之前的 INSERT 语句不会成功，一切都将被回滚到最初状态。

异常

异常处理可以允许你处理一个异常情况，例如可控方式的程序定义错误。

当异常情况发生时，将抛出一个异常。抛出这个词意味着当前执行的程序停止，控制器被重定向到最近的适用的 catch 子句。如果没有适用的 catch 子句存在，那么程序执行被终止。

JDBC 的异常处理是非常类似于 Java 的异常处理，但对于 JDBC，最常见的异常是 java.sql.SQLException。

SQLException 方法

SQLException 异常在驱动程序和数据库中都可能出现。当出现这个异常时，SQLException 类型的对象将被传递到 catch 子句。

传递的 SQLException 对象具有以下的方法，以下的方法可用于检索该异常的额外信息−

方法	描述
getErrorCode( )	获取与异常关联的错误号。
getMessage( )	获取 JDBC 驱动程序的错误信息，该错误是由驱动程序处理的，或者在数据库错误中获取 Oracl 错误号和错误信息。
getSQLState( )	获取 XOPEN SQLstate 字符串。对于 JDBC 驱动程序错误，使用该方法不能返回有用的信息。对于数据库错误，返回第五位的 XOPEN SQLstate 代码。该方法可以返回 null。
getNextException( )	获取异常链的下一个 Exception 对象。
printStackTrace( )	打印当前异常或者抛出，其回溯到标准的流错误。
printStackTrace(PrintStream s)	打印该抛出，其回溯到你指定的打印流。
printStackTrace(PrintWriter w)	打印该抛出，其回溯到你指定的打印写入。

通过利用可从 Exception 对象提供的信息，你可以捕获异常并继续运行程序。这是一个 try 块的一般格式−

```
try {
   // Your risky code goes between these curly braces!!!
}
catch(Exception ex) {
   // Your exception handling code goes between these
   // curly braces, similar to the exception clause
   // in a PL/SQL block.
}
finally {
   // Your must-always-be-executed code goes between these
```

```
  // curly braces. Like closing database connection.
}
```

示例

研究学习下面的示例代码来了解 try .... catch ... finally 块的使用。

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/EMP";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";

  public static void main(String[] args) {
  Connection conn = null;
  try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting to database...");
    conn = DriverManager.getConnection(DB_URL,USER,PASS);

    //STEP 4: Execute a query
    System.out.println("Creating statement...");
    Statement stmt = conn.createStatement();
    String sql;
    sql = "SELECT id, first, last, age FROM Employees";
    ResultSet rs = stmt.executeQuery(sql);

    //STEP 5: Extract data from result set
    while(rs.next()){
      //Retrieve by column name
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");

      //Display values
```

```
      System.out.print("ID: " + id);
      System.out.print(", Age: " + age);
      System.out.print(", First: " + first);
      System.out.println(", Last: " + last);
    }
    //STEP 6: Clean-up environment
    rs.close();
    stmt.close();
    conn.close();
  }catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
  }catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
  }finally{
    //finally block used to close resources
    try{
      if(conn!=null)
        conn.close();
    }catch(SQLException se){
      se.printStackTrace();
    }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，如果没有问题它将展示下面的结果，否则相应的错误将被捕获并会显示错误消息−

```
C:\>java JDBCExample
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
C:\>
```

通过传递错误的数据库名称或错误的用户名或错误的密码来测试上面的示例，并检查结果。

批处理

批处理是指你将关联的 SQL 语句组合成一个批处理，并将他们当成一个调用提交给数据库。

当你一次发送多个 SQL 语句到数据库时，可以减少通信的资源消耗，从而提高了性能。

- JDBC 驱动程序不一定支持该功能。你可以使用 *DatabaseMetaData.supportsBatchUpdates()* 方法来确定目标数据库是否支持批处理更新。如果你的JDBC驱动程序支持此功能，则该方法返回值为 true。
- Statement，PreparedStatement 和 CallableStatement 的 addBatch() 方法用于添加单个语句到批处理。
- executeBatch() 方法用于启动执行所有组合在一起的语句。
- executeBatch() 方法返回一个整数数组，数组中的每个元素代表了各自的更新语句的更新数目。
- 正如你可以添加语句到批处理中，你也可以用 clearBatch() 方法删除它们。此方法删除所有用 addBatch() 方法添加的语句。但是，你不能有选择性地选择要删除的语句。

批处理和 Statement 对象

使用 Statement 对象来使用批处理所需要的典型步骤如下所示−

- 使用 *createStatement()* 方法创建一个 Statement 对象。
- 使用 *setAutoCommit()* 方法将自动提交设为 false。
- 被创建的 Statement 对象可以使用 *addBatch()* 方法来添加你想要的所有SQL语句。
- 被创建的 Statement 对象可以用 *executeBatch()* 将所有的 SQL 语句执行。
- 最后，使用 *commit()* 方法提交所有的更改。

示例

下面的代码段提供了一个使用 Statement 对象批量更新的例子−

```
// Create statement object
Statement stmt = conn.createStatement();

// Set auto-commit to false
```

```
conn.setAutoCommit(false);

// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(200,'Zia', 'Ali', 30)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(201,'Raj', 'Kumar', 35)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "UPDATE Employees SET age = 35 " +
             "WHERE id = 100";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create an int[] to hold returned values
int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes
conn.commit();
```

批处理和 PrepareStatement 对象

使用 prepareStatement 对象来使用批处理需要的典型步骤如下所示−

- 使用占位符创建 SQL 语句。
- 使用任一 *prepareStatement()* 方法创建 prepareStatement 对象。
- 使用 *setAutoCommit()* 方法将自动提交设为 false。
- 被创建的 Statement 对象可以使用 *addBatch()* 方法来添加你想要的所有 SQL 语句。
- 被创建的 Statement 对象可以用 *executeBatch()* 将所有的 SQL 语句执行。
- 最后，使用 *commit()* 方法提交所有的更改。

下面的代码段提供了一个使用 PrepareStatement 对象批量更新的示例−

```
// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
```

```
          "VALUES(?, ?, ?, ?)";

// Create PrepareStatement object
PreparedStatemen pstmt = conn.prepareStatement(SQL);

//Set auto-commit to false
conn.setAutoCommit(false);

// Set the variables
pstmt.setInt( 1, 400 );
pstmt.setString( 2, "Pappu" );
pstmt.setString( 3, "Singh" );
pstmt.setInt( 4, 33 );
// Add it to the batch
pstmt.addBatch();

// Set the variables
pstmt.setInt( 1, 401 );
pstmt.setString( 2, "Pawan" );
pstmt.setString( 3, "Singh" );
pstmt.setInt( 4, 31 );
// Add it to the batch
pstmt.addBatch();

//add more batches
.
.
.
.
//Create an int[] to hold returned values
int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes
conn.commit();
```

存储过程

在前面的 JDBC-Statement 对象这章里，我们已经学习了如何在 JDBC 中使用**存储过程**。本章与该章有些类似，但本章将告诉你有关 JDBC 转义语法的额外信息。

正如一个 Connection 对象创建了 Statement 和 PreparedStatement 对象，它也创造了在数据库中被执行调用的 CallableStatement 对象。

创建 CallableStatement 对象

假设，你需要执行下面的 Oracle 存储过程-

```
CREATE OR REPLACE PROCEDURE getEmpName
  (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END;
```

注意：上面的存储过程是在 Oracle 使用的，但我们使用的是 MySQL 数据库，所以我们在 MySQL 的环境下需要重新写出相同功能的代码，下面的代码是在 EMP 数据库中创建相同功能的代码-

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
  (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END $$

DELIMITER ;
```

当前有三种类型的参数：IN，OUT 和 INOUT。PreparedStatement 对象只能使用 IN 参数。CallableStatement 对象可以使用所有的三种类型。

下面是三种类型参数的定义-

参数	描述
IN	当 SQL 语句创建的时候，该参数的值是未知的。你可以用 setXXX() 方法将值绑定到 IN 参数里。
OUT	该参数的值是由 SQL 语句的返回值。你可以用 getXXX() 方法从 OUT 参数中检索值。
INOUT	该参数同时提供输入和输出值。你可以用 setXXX() 方法将值绑定到 IN 参数里，并且也可以用 getXXX() 方法从 OUT 参数中检索值。

下面的代码片段展示了如何使用 Connection.prepareCall() 方法实现一个基于上述存储过程的 CallableStatement 对象−

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

字符串变量 SQL 使用参数占位符来表示存储过程。

使用 CallableStatement 对象就像使用 PreparedStatement 对象。在执行该语句前，你必须将值绑定到所有的参数，否则你将收到一个 SQL 异常。

如果你有 IN 参数，只要按照适用于 PreparedStatement 对象相同的规则和技巧；用 setXXX()方法来绑定对应的 Java 数据类型。

当你使用 OUT 和 INOUT 参数就必须采用额外的 CallableStatement 方法：registerOutParameter()。registerOutParameter() 方法将 JDBC 数据类型绑定到存储过程返回的数据类型。

一旦你调用了存储过程，你可以用适当的 getXXX()方法从 OUT 参数参数中检索数值。这种方法将检索出来的 SQL 类型的值映射到 Java 数据类型。

关闭 CallableStatement 对象

正如你关闭其它的 Statement 对象，出于同样的原因，你也应该关闭 CallableStatement 对象。

close()方法简单的调用就可以完成这项工作。如果你先关闭了 Connection 对象，那么它也会关闭 CallableStatement 对象。然而，你应该始终明确关闭 CallableStatement 对象，以确保该对象被彻底关闭。

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   cstmt.close();
}
```

JDBC 的 SQL 转义语法

转义语法使能够让你通过使用标准的 JDBC 方法和属性，来灵活的使用数据库的某些特定功能，而该特定功能对你来说本来是不可用的。

常用的 SQL 转义语法格式如下所示−

```
{keyword 'parameters'}
```

当你在编程的时候，你会发现以下的这些转义序列会非常有用的−

d，t，ts 关键字

它们能帮助确定日期，时间和时间戳的文字。众所周知，没有两个数据库管理系统的时间和日期的表现方式是相同的。该转义语法告诉驱动程序以目标数据库的格式来呈现日期或时间。例如−

```
{d 'yyyy−mm−dd'}
```

其中 yyyy = 年，mm = 月，DD = 日。使用这种语法 {d '2009−09−03'} 即是 2009 年 3 月 9 日。

下面是一个说明如何在表中插入日期的简单例子−

```
//Create a Statement object
stmt = conn.createStatement();
//Insert data ==> ID, First Name, Last Name, DOB
String sql="INSERT INTO STUDENTS VALUES" +
```

```
            "(100,'Zara','Ali', {d '2001-12-16'})";

stmt.executeUpdate(sql);
```

同样的，你可以使用以下两种语法之一，无论是 t 或 ts −

```
{t 'hh:mm:ss'}
```

其中 hh = 小时， mm = 分， ss = 秒。使用这种语法 {t '13:30:29'} 即是下午 1 点 30 分 29 秒。

```
{ts 'yyyy-mm-dd hh:mm:ss'}
```

这是用上述两种语法 'd' 和 't' 表示时间戳的组合语法。

escape 关键字

该关键字在 LIKE 子句中使用，来定义转义字符。当使用 SQL 通配符%，来匹配零个或多个字符时，该关键字就非常有用。例如−

```
String sql = "SELECT symbol FROM MathSymbols
            WHERE symbol LIKE '\%' {escape '\'}";
stmt.execute(sql);
```

如果你使用反斜杠字符（\）作为转义字符，你必须在 Java 字符串里使用两个反斜杠字符，因为反斜杠也是一个 Java 转义字符。

fn 关键字

该关键字代表在数据库管理系统中使用标量函数。例如，你可以使用 SQL 的 length 函数来计算字符串的长度−

```
{fn length('Hello World')}
```

这将返回11，也就是字符串'Hello World'的长度。

call 关键字

该关键字是用来调用存储过程的。例如，对于一个需要一个 IN 参数的存储过程，使用以下语法−

```
{call my_procedure(?)};
```

对于需要一个 IN 参数并返回一个 OUT 参数的存储过程，使用下面的语法−

```
{? = call my_procedure(?)};
```

oj 关键字

该关键字用来表示外部连接，其语法如下所示−

```
{oj outer−join}
```

其中 outer − join = 表 { LEFT | RIGHT | FULL } OUTER JOIN {表| outer − join }的搜索条件。例如−

```
String sql = "SELECT Employees
        FROM {oj ThisTable RIGHT
        OUTER JOIN ThatTable on id = '100'}";
stmt.execute(sql);
```

流数据

PreparedStatement 对象必须具备使用输入和输出流来提供参数数据的能力。这使你能够将整个文件存储到数据库列中，这样数据库就能存储大型数据，例如 CLOB 和 BLOB 数据类型。

用于流数据有下列几种方法−

- setAsciiStream(): 该方法是用来提供较大的 ASCII 值。
- setCharacterStream(): 该方法是用来提供较大的 UNICODE 值。
- setBinaryStream(): 该方法是用来提供较大的二进制值。

setXXXStream()方法需要一个额外的参数，该参数是除了参数占位符的文件大小。这个参数通知驱动程序通过使用流有多少数据被发送到数据库中。

示例

假如我们到要上传一个名为 XML_Data.xml 的 XML 文件到数据库的表中。下面是该 XML 文件的内容−

```
<?xml version="1.0"?>
<Employee>
<id>100</id>
<first>Zara</first>
<last>Ali</last>
<Salary>10000</Salary>
<Dob>18-08-1978</Dob>
<Employee>
```

将该 XML 文件和你要运行的示例保存在相同的目录的。

这个示例将创建一个数据库表 XML_Data ，然后 XML_Data.xml 将被上传到该表中。

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
// Import required packages
import java.sql.*;
import java.io.*;
import java.util.*;

public class JDBCExample {
  // JDBC driver name and database URL
```

```
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost/EMP";

//  Database credentials
static final String USER = "username";
static final String PASS = "password";

public static void main(String[] args) {
Connection conn = null;
PreparedStatement pstmt = null;
Statement stmt = null;
ResultSet rs = null;
try{
  // Register JDBC driver
  Class.forName("com.mysql.jdbc.Driver");

  // Open a connection
  System.out.println("Connecting to database...");
  conn = DriverManager.getConnection(DB_URL,USER,PASS);

  //Create a Statement object and build table
  stmt = conn.createStatement();
  createXMLTable(stmt);

  //Open a FileInputStream
  File f = new File("XML_Data.xml");
  long fileLength = f.length();
  FileInputStream fis = new FileInputStream(f);

  //Create PreparedStatement and stream data
  String SQL = "INSERT INTO XML_Data VALUES (?,?)";
  pstmt = conn.prepareStatement(SQL);
  pstmt.setInt(1,100);
  pstmt.setAsciiStream(2,fis,(int)fileLength);
  pstmt.execute();

  //Close input stream
  fis.close();

  // Do a query to get the row
  SQL = "SELECT Data FROM XML_Data WHERE id=100";
  rs = stmt.executeQuery (SQL);
  // Get the first row
  if (rs.next ()){
    //Retrieve data from input stream
```

```
        InputStream xmlInputStream = rs.getAsciiStream (1);
        int c;
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        while (( c = xmlInputStream.read ()) != -1)
          bos.write(c);
        //Print results
        System.out.println(bos.toString());
      }
      // Clean-up environment
      rs.close();
      stmt.close();
      pstmt.close();
      conn.close();
    }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
    }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
    }finally{
      //finally block used to close resources
      try{
        if(stmt!=null)
          stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
        if(pstmt!=null)
          pstmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
        if(conn!=null)
          conn.close();
      }catch(SQLException se){
        se.printStackTrace();
      }//end finally try
    }//end try
    System.out.println("Goodbye!");
  }//end main

  public static void createXMLTable(Statement stmt)
    throws SQLException{
    System.out.println("Creating XML_Data table..." );
    //Create SQL Statement
```

```
   String streamingDataSql = "CREATE TABLE XML_Data " +
                             "(id INTEGER, Data LONG)";
   //Drop table first if it exists.
   try{
      stmt.executeUpdate("DROP TABLE XML_Data");
   }catch(SQLException se){
   }// do nothing
   //Build table.
   stmt.executeUpdate(streamingDataSql);
}//end createXMLTable
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

``` C:>java JDBCExample Connecting to database... Creating XML_Data table... 100 Zara Ali 10000 1 8-08-1978 Goodbye! C:>

```

2

JDBC 示例

创建数据库实例

本教程介绍了如何使用 JDBC 应用程序创建数据库的示例。执行下面的示例之前，请确保你已做好以下工作−

- 在给定的模式下你应该有管理员权限去创建一个数据库。执行下面的例子，你需要用你实际的用户名和密码去代替 *username* 和 *password* 。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。 要创建一个新的数据库，你不用给任何数据库名，只需要准备下面示例中提到的数据库 URL。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句到数据库。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/";

  //  Database credentials
  static final String USER = "username";
```

```
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);

      //STEP 4: Execute a query
      System.out.println("Creating database...");
      stmt = conn.createStatement();

      String sql = "CREATE DATABASE STUDENTS";
      stmt.executeUpdate(sql);
      System.out.println("Database created successfully...");
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to database...
Creating database...
Database created successfully...
Goodbye!
C:\>
```

选择数据库实例

本章介绍了如何使用 JDBC 应用程序选择一个数据库的示例。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password*。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个连接对象，它代表与所选择数据库的物理连接。 用数据库 URL 来选择数据库。下面的例子会连接到 STUDENTS 数据库。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";

  public static void main(String[] args) {
  Connection conn = null;
```

```
  try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting to a selected database...");
    conn = DriverManager.getConnection(DB_URL, USER, PASS);
    System.out.println("Connected database successfully...");
  }catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
  }catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
  }finally{
    //finally block used to close resources
    try{
      if(conn!=null)
        conn.close();
    }catch(SQLException se){
      se.printStackTrace();
    }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Goodbye!
C:\>
```

删除数据库实例

本章介绍了如何使用 JDBC 应用程序来删除一个现有数据库的示例。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password* 。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

注意：这是一个重要的操作，在你删除数据库之前必须慎重考虑，因为一旦操作，在这个数据库里的所有数据都将删除。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。删除数据库不需要在你的数据库URL中出现数据库的名字。以下示例将删除 STUDENT 数据库。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句到数据库。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/";
```

```
// Database credentials
static final String USER = "username";
static final String PASS = "password";

public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
  //STEP 2: Register JDBC driver
  Class.forName("com.mysql.jdbc.Driver");

  //STEP 3: Open a connection
  System.out.println("Connecting to a selected database...");
  conn = DriverManager.getConnection(DB_URL, USER, PASS);
  System.out.println("Connected database successfully...");

  //STEP 4: Execute a query
  System.out.println("Deleting database...");
  stmt = conn.createStatement();

  String sql = "DROP DATABASE STUDENTS";
  stmt.executeUpdate(sql);
  System.out.println("Database deleted successfully...");
}catch(SQLException se){
  //Handle errors for JDBC
  se.printStackTrace();
}catch(Exception e){
  //Handle errors for Class.forName
  e.printStackTrace();
}finally{
  //finally block used to close resources
  try{
    if(stmt!=null)
      conn.close();
  }catch(SQLException se){
  }// do nothing
  try{
    if(conn!=null)
      conn.close();
  }catch(SQLException se){
    se.printStackTrace();
  }//end finally try
}//end try
System.out.println("Goodbye!");
```

```
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Deleting database...
Database deleted successfully...
Goodbye!
C:\>
```

创建表实例

本章介绍了如何使用 JDBC 应用程序来创建一个表的示例。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password* 。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句到被选择的数据库中去创建表。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";

  public static void main(String[] args) {
```

```
Connection conn = null;
Statement stmt = null;
try{
   //STEP 2: Register JDBC driver
   Class.forName("com.mysql.jdbc.Driver");

   //STEP 3: Open a connection
   System.out.println("Connecting to a selected database...");
   conn = DriverManager.getConnection(DB_URL, USER, PASS);
   System.out.println("Connected database successfully...");

   //STEP 4: Execute a query
   System.out.println("Creating table in given database...");
   stmt = conn.createStatement();

   String sql = "CREATE TABLE REGISTRATION " +
                "(id INTEGER not NULL, " +
                " first VARCHAR(255), " +
                " last VARCHAR(255), " +
                " age INTEGER, " +
                " PRIMARY KEY ( id ))";

   stmt.executeUpdate(sql);
   System.out.println("Created table in given database...");
}catch(SQLException se){
   //Handle errors for JDBC
   se.printStackTrace();
}catch(Exception e){
   //Handle errors for Class.forName
   e.printStackTrace();
}finally{
   //finally block used to close resources
   try{
      if(stmt!=null)
         conn.close();
   }catch(SQLException se){
   }// do nothing
   try{
      if(conn!=null)
         conn.close();
   }catch(SQLException se){
      se.printStackTrace();
   }//end finally try
}//end try
System.out.println("Goodbye!");
```

```
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating table in given database...
Created table in given database...
Goodbye!
C:\>
```

删除表实例

本章介绍了如何使用 JDBC 应用程序来删除一个表的示例。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password*。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

注意：这是一个重要的操作，在你删除表之前必须慎重考虑，因为一旦操作，在这个表里的所有数据都将删除。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句到被选择的数据库中去删除表。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";
```

```
  public static void main(String[] args) {
  Connection conn = null;
  Statement stmt = null;
  try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting to a selected database...");
    conn = DriverManager.getConnection(DB_URL, USER, PASS);
    System.out.println("Connected database successfully...");

    //STEP 4: Execute a query
    System.out.println("Deleting table in given database...");
    stmt = conn.createStatement();

    String sql = "DROP TABLE REGISTRATION ";

    stmt.executeUpdate(sql);
    System.out.println("Table  deleted in given database...");
  }catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
  }catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
  }finally{
    //finally block used to close resources
    try{
      if(stmt!=null)
         conn.close();
    }catch(SQLException se){
    }// do nothing
    try{
      if(conn!=null)
         conn.close();
    }catch(SQLException se){
      se.printStackTrace();
    }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Deleting table in given database...
Table  deleted in given database...
Goodbye!
C:\>
```

插入记录实例

本章介绍了如何使用 JDBC 应用程序在表中插入记录的示例。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password*。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。
- **执行查询**：需要使用声明的类型对象建立并提交一个 SQL 语句，这样可以在表中插入记录。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";

  public static void main(String[] args) {
```

```
Connection conn = null;
Statement stmt = null;
try{
   //STEP 2: Register JDBC driver
   Class.forName("com.mysql.jdbc.Driver");

   //STEP 3: Open a connection
   System.out.println("Connecting to a selected database...");
   conn = DriverManager.getConnection(DB_URL, USER, PASS);
   System.out.println("Connected database successfully...");

   //STEP 4: Execute a query
   System.out.println("Inserting records into the table...");
   stmt = conn.createStatement();

   String sql = "INSERT INTO Registration " +
                "VALUES (100, 'Zara', 'Ali', 18)";
   stmt.executeUpdate(sql);
   sql = "INSERT INTO Registration " +
                "VALUES (101, 'Mahnaz', 'Fatma', 25)";
   stmt.executeUpdate(sql);
   sql = "INSERT INTO Registration " +
                "VALUES (102, 'Zaid', 'Khan', 30)";
   stmt.executeUpdate(sql);
   sql = "INSERT INTO Registration " +
                "VALUES(103, 'Sumit', 'Mittal', 28)";
   stmt.executeUpdate(sql);
   System.out.println("Inserted records into the table...");

}catch(SQLException se){
   //Handle errors for JDBC
   se.printStackTrace();
}catch(Exception e){
   //Handle errors for Class.forName
   e.printStackTrace();
}finally{
   //finally block used to close resources
   try{
      if(stmt!=null)
         conn.close();
   }catch(SQLException se){
   }// do nothing
   try{
      if(conn!=null)
         conn.close();
```

```
    }catch(SQLException se){
       se.printStackTrace();
    }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Inserting records into the table...
Inserted records into the table...
Goodbye!
C:\>
```

查询记录实例

本章介绍了如何使用 JDBC 应用程序在表中查询或者提取记录的示例。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password*。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句，这样可以从表中查询或者提取记录。
- **提取数据**：当 SQL 语句被运行的时候，你就可以从表中提取记录了。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
```

```
static final String PASS = "password";

public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
  //STEP 2: Register JDBC driver
  Class.forName("com.mysql.jdbc.Driver");

  //STEP 3: Open a connection
  System.out.println("Connecting to a selected database...");
  conn = DriverManager.getConnection(DB_URL, USER, PASS);
  System.out.println("Connected database successfully...");

  //STEP 4: Execute a query
  System.out.println("Creating statement...");
  stmt = conn.createStatement();

  String sql = "SELECT id, first, last, age FROM Registration";
  ResultSet rs = stmt.executeQuery(sql);
  //STEP 5: Extract data from result set
  while(rs.next()){
    //Retrieve by column name
    int id  = rs.getInt("id");
    int age = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
  }
  rs.close();
}catch(SQLException se){
  //Handle errors for JDBC
  se.printStackTrace();
}catch(Exception e){
  //Handle errors for Class.forName
  e.printStackTrace();
}finally{
  //finally block used to close resources
  try{
    if(stmt!=null)
```

```
      conn.close();
   }catch(SQLException se){
   }// do nothing
   try{
     if(conn!=null)
       conn.close();
   }catch(SQLException se){
     se.printStackTrace();
   }//end finally try
 }//end try
 System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Goodbye!
C:\>
```

更新记录实例

本章介绍了如何使用 JDBC 应用程序在表中更新记录的示例。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password*。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句，这样可以更新表中的记录。这个 SQL 语句用 IN 和 WHERE 子句来更新符合条件的记录。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";
```

```
public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
   //STEP 2: Register JDBC driver
   Class.forName("com.mysql.jdbc.Driver");

   //STEP 3: Open a connection
   System.out.println("Connecting to a selected database...");
   conn = DriverManager.getConnection(DB_URL, USER, PASS);
   System.out.println("Connected database successfully...");

   //STEP 4: Execute a query
   System.out.println("Creating statement...");
   stmt = conn.createStatement();
   String sql = "UPDATE Registration " +
                "SET age = 30 WHERE id in (100, 101)";
   stmt.executeUpdate(sql);

   // Now you can extract all the records
   // to see the updated records
   sql = "SELECT id, first, last, age FROM Registration";
   ResultSet rs = stmt.executeQuery(sql);

   while(rs.next()){
      //Retrieve by column name
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");

      //Display values
      System.out.print("ID: " + id);
      System.out.print(", Age: " + age);
      System.out.print(", First: " + first);
      System.out.println(", Last: " + last);
   }
   rs.close();
}catch(SQLException se){
   //Handle errors for JDBC
   se.printStackTrace();
}catch(Exception e){
   //Handle errors for Class.forName
   e.printStackTrace();
```

```
  }finally{
     //finally block used to close resources
     try{
        if(stmt!=null)
           conn.close();
     }catch(SQLException se){
     }// do nothing
     try{
        if(conn!=null)
           conn.close();
     }catch(SQLException se){
        se.printStackTrace();
     }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating statement...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 101, Age: 30, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Goodbye!
C:\>
```

删除记录实例

本章介绍了如何使用 JDBC 应用程序在表中删除记录的示例。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password*。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.*就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句，这样可以更新表中的记录。这个 SQL 语句用 WHERE 子句来作为删除符合条件的记录。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";
```

```
public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
   //STEP 2: Register JDBC driver
   Class.forName("com.mysql.jdbc.Driver");

   //STEP 3: Open a connection
   System.out.println("Connecting to a selected database...");
   conn = DriverManager.getConnection(DB_URL, USER, PASS);
   System.out.println("Connected database successfully...");

   //STEP 4: Execute a query
   System.out.println("Creating statement...");
   stmt = conn.createStatement();
   String sql = "DELETE FROM Registration " +
                "WHERE id = 101";
   stmt.executeUpdate(sql);

   // Now you can extract all the records
   // to see the remaining records
   sql = "SELECT id, first, last, age FROM Registration";
   ResultSet rs = stmt.executeQuery(sql);

   while(rs.next()){
      //Retrieve by column name
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");

      //Display values
      System.out.print("ID: " + id);
      System.out.print(", Age: " + age);
      System.out.print(", First: " + first);
      System.out.println(", Last: " + last);
   }
   rs.close();
}catch(SQLException se){
   //Handle errors for JDBC
   se.printStackTrace();
}catch(Exception e){
   //Handle errors for Class.forName
   e.printStackTrace();
```

```
  }finally{
     //finally block used to close resources
     try{
        if(stmt!=null)
           conn.close();
     }catch(SQLException se){
     }// do nothing
     try{
        if(conn!=null)
           conn.close();
     }catch(SQLException se){
        se.printStackTrace();
     }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
Creating statement...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Goodbye!
C:\>
```

WHERE 子句实例

本章介绍了如何使用 JDBC 应用程序在表中查询记录的示例。在表中查询记录时，将通过 WHERE 子句来增加附加条件。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password*。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句，并依据提供的条件从表中提取记录。这个查询语句用 WHERE 子句来查询记录。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
```

```
static final String PASS = "password";

public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
  //STEP 2: Register JDBC driver
  Class.forName("com.mysql.jdbc.Driver");

  //STEP 3: Open a connection
  System.out.println("Connecting to a selected database...");
  conn = DriverManager.getConnection(DB_URL, USER, PASS);
  System.out.println("Connected database successfully...");

  //STEP 4: Execute a query
  System.out.println("Creating statement...");
  stmt = conn.createStatement();

  // Extract records without any condition.
  System.out.println("Fetching records without condition...");
  String sql = "SELECT id, first, last, age FROM Registration";
  ResultSet rs = stmt.executeQuery(sql);

  while(rs.next()){
    //Retrieve by column name
    int id  = rs.getInt("id");
    int age = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
  }

  // Select all records having ID equal or greater than 101
  System.out.println("Fetching records with condition...");
  sql = "SELECT id, first, last, age FROM Registration" +
            " WHERE id >= 101 ";
  rs = stmt.executeQuery(sql);

  while(rs.next()){
    //Retrieve by column name
```

```
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");

      //Display values
      System.out.print("ID: " + id);
      System.out.print(", Age: " + age);
      System.out.print(", First: " + first);
      System.out.println(", Last: " + last);
    }
    rs.close();
  }catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
  }catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
  }finally{
    //finally block used to close resources
    try{
      if(stmt!=null)
        conn.close();
    }catch(SQLException se){
    }// do nothing
    try{
      if(conn!=null)
        conn.close();
    }catch(SQLException se){
      se.printStackTrace();
    }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
```

```
Creating statement...
Fetching records without condition...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Fetching records with condition...
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Goodbye!
C:\>
```

LIKE 子句实例

本章介绍了如何使用 JDBC 应用程序在表中查询记录的示例。在表中查询记录时，将通过 LIKE 子句来增加附加条件。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password*。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句，并依据提供的条件从表中提取记录。这个查询语句用 LIKE 子句，将所有名字以 “Za” 开头的学生的记录查询出来。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
```

```
static final String PASS = "password";

public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
  //STEP 2: Register JDBC driver
  Class.forName("com.mysql.jdbc.Driver");

  //STEP 3: Open a connection
  System.out.println("Connecting to a selected database...");
  conn = DriverManager.getConnection(DB_URL, USER, PASS);
  System.out.println("Connected database successfully...");

  //STEP 4: Execute a query
  System.out.println("Creating statement...");
  stmt = conn.createStatement();

  // Extract records without any condition.
  System.out.println("Fetching records without condition...");
  String sql = "SELECT id, first, last, age FROM Registration";
  ResultSet rs = stmt.executeQuery(sql);

  while(rs.next()){
    //Retrieve by column name
    int id  = rs.getInt("id");
    int age = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
  }

  // Select all records having ID equal or greater than 101
  System.out.println("Fetching records with condition...");
  sql = "SELECT id, first, last, age FROM Registration" +
            " WHERE first LIKE '%za%' ";
  rs = stmt.executeQuery(sql);

  while(rs.next()){
    //Retrieve by column name
```

```
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");

      //Display values
      System.out.print("ID: " + id);
      System.out.print(", Age: " + age);
      System.out.print(", First: " + first);
      System.out.println(", Last: " + last);
    }
    rs.close();
  }catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
  }catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
  }finally{
    //finally block used to close resources
    try{
      if(stmt!=null)
        conn.close();
    }catch(SQLException se){
    }// do nothing
    try{
      if(conn!=null)
        conn.close();
    }catch(SQLException se){
      se.printStackTrace();
    }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample
Connecting to a selected database...
Connected database successfully...
```

```
Creating statement...
Fetching records without condition...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
Fetching records with condition...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 102, Age: 30, First: Zaid, Last: Khan
Goodbye!
C:\>
```

排序实例

本章介绍了如何使用 JDBC 应用程序将表中记录进行排序的示例。该示例将使用 asc 和 desc 作为关键字将记录按升序或降序排序。执行下面的示例之前，请确保你已做好以下工作−

- 在运行下面的例子之前，你需要用你实际的用户名和密码去代替 *username* 和 *password*。
- 你的 MySQL 或者其他数据库已经启动了并在运行中。

所需的步骤

用 JDBC 应用程序去创建一个新的数据库需要执行以下步骤−

- **导入包**：要求你包括含有需要进行数据库编程的 JDBC 类的包。大多数情况下，使用 *import java.sql.* 就足够了。
- **注册 JDBC 驱动程序**：要求你初始化驱动程序，这样你可以与数据库打开通信通道。
- **打开连接**：需要使用 *DriverManager.getConnection()* 方法创建一个 Connection 对象，它代表与数据库服务器的物理连接。
- **执行查询**：需要使用类型声明的对象建立并提交一个 SQL 语句，并依据提供的条件将表的记录进行排序。这个查询语句用 asc 和 desc 子句对数据进行升序或者降序排序。
- **清理环境**：依靠 JVM 垃圾收集器可以明确地回收所有的数据库资源。

示例代码

将下面的示例拷贝并粘帖到 JDBCExample.java 中，编译并运行它，如下所示−

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

  //  Database credentials
  static final String USER = "username";
```

```
static final String PASS = "password";

public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
  //STEP 2: Register JDBC driver
  Class.forName("com.mysql.jdbc.Driver");

  //STEP 3: Open a connection
  System.out.println("Connecting to a selected database...");
  conn = DriverManager.getConnection(DB_URL, USER, PASS);
  System.out.println("Connected database successfully...");

  //STEP 4: Execute a query
  System.out.println("Creating statement...");
  stmt = conn.createStatement();

  // Extract records in ascending order by first name.
  System.out.println("Fetching records in ascending order...");
  String sql = "SELECT id, first, last, age FROM Registration" +
           " ORDER BY first ASC";
  ResultSet rs = stmt.executeQuery(sql);

  while(rs.next()){
    //Retrieve by column name
    int id  = rs.getInt("id");
    int age = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
  }

  // Extract records in descending order by first name.
  System.out.println("Fetching records in descending order...");
  sql = "SELECT id, first, last, age FROM Registration" +
           " ORDER BY first DESC";
  rs = stmt.executeQuery(sql);

  while(rs.next()){
```

```
      //Retrieve by column name
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");

      //Display values
      System.out.print("ID: " + id);
      System.out.print(", Age: " + age);
      System.out.print(", First: " + first);
      System.out.println(", Last: " + last);
    }
    rs.close();
  }catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
  }catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
  }finally{
    //finally block used to close resources
    try{
      if(stmt!=null)
        conn.close();
    }catch(SQLException se){
    }// do nothing
    try{
      if(conn!=null)
        conn.close();
    }catch(SQLException se){
      se.printStackTrace();
    }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

现在，让我们用下面的命令编译上面的代码−

```
C:\>javac JDBCExample.java
C:\>
```

当你运行 JDBCExample 时，它将展示下面的结果−

```
C:\>java JDBCExample Connecting to a selected database... Connected database successfully... Creating statement... Fetching records in ascending order... ID: 103, Age: 28, First: Sumit, Last: Mittal ID: 102, Age: 3
```

```
0, First: Zaid, Last: Khan ID: 100, Age: 30, First: Zara, Last: Ali Fetching records in descending order... ID: 10
0, Age: 30, First: Zara, Last: Ali ID: 102, Age: 30, First: Zaid, Last: Khan ID: 103, Age: 28, First: Sumit, Last: M
ittal Goodbye! C:\>
```

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台

更多信息请访问

http://wiki.jikexueyuan.com/project/jdbc/