数据库系统概论

An Introduction to Database System

第八章 数据库编程

第八章 数据库编程

嵌入式SQL

存储过程和触发器

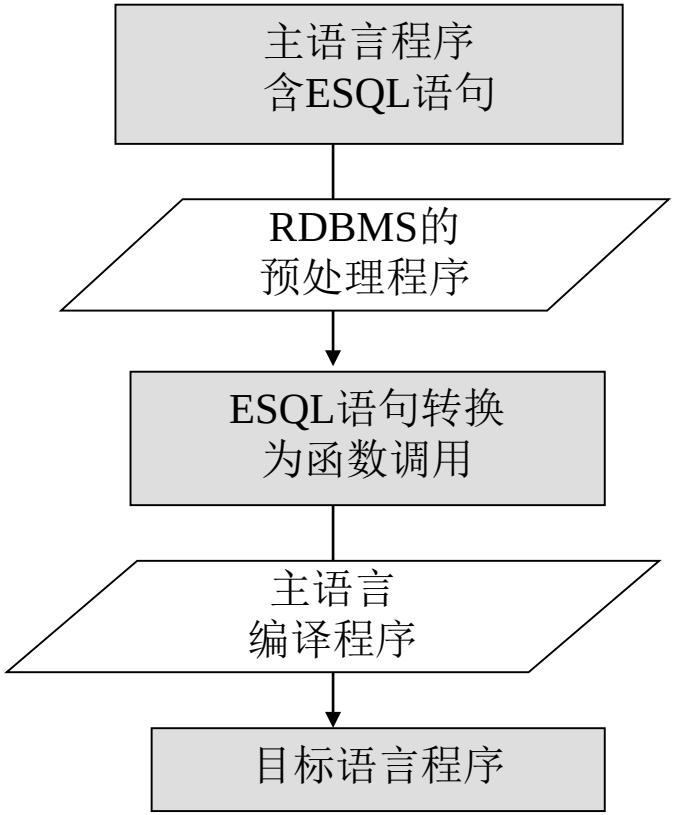
JDBC编程

8.1 嵌入式SQL

- SQL语言提供了两种不同的使用方式：
  - 交互式
  - 嵌入式
- 为什么要引入嵌入式SQL
  - SQL语言是非过程性语言
  - 事务处理应用需要高级语言
- 这两种方式细节上有差别，在程序设计的环境下，SQL语句要做某些必要的扩充

8.1.1 嵌入式SQL的处理过程

- 主语言
  - 嵌入式SQL是将SQL语句嵌入程序设计语言中，被嵌入的程序设计语言，如C、C++、Java，称为宿主语言，简称主语言。
- 处理过程预编译方法



ESQL基本处理过程

嵌入式SQL的处理过程（续）

- 为了区分SQL语句与主语言语句， 所有SQL语句必须加前缀EXEC SQL，以(;)结束:

  EXEC SQL <SQL语句>;

五、程序实例

[例1]依次检查某个系的学生记录，交互式更新某些学生年龄。

```
EXEC SQL BEGIN DECLARE SECTION;  /*主变量说明开始*/
    char deptname[64];
    char HSno[64];
    char HSname[64];
    char HSsex[64];
    int     HSage;
    int     NEWAGE;
EXEC SQL END DECLARE SECTION;    /*主变量说明结束*/
long SQLCODE;
EXEC SQL INCLUDE sqlca;              /*定义SQL通信区*/
```

程序实例（续）

```
int main(void)                          /*C语言主程序开始*/
{
    int     count = 0;
    char  yn;                           /*变量yn代表yes或no*/
    printf("Please choose the department name(CS/MA/IS): ");
    scanf("%s", deptname);      /*为主变量deptname赋值*/
    EXEC SQL CONNECT TO TEST@localhost:54321 USER
     "SYSTEM" /"MANAGER";   /*连接数据库TEST*/
    EXEC SQL DECLARE SX CURSOR FOR   /*定义游标*/
            SELECT Sno, Sname, Ssex, Sage   /*SX对应语句的执行结果*/
            FROM Student
            WHERE SDept = :deptname;
    EXEC SQL OPEN SX;      /*打开游标SX便指向查询结果的第一行*/
```

程序实例（续）

```
for ( ; ; )                          /*用循环结构逐条处理结果集中的记录*/
{
    EXEC SQL FETCH SX INTO :HSno, :HSname, :HSsex,:HSage;
                                     /*推进游标，将当前数据放入主变量*/
    if (sqlca.sqlcode != 0)          /* sqlcode != 0,表示操作不成功*/
          break;                     /*利用SQLCA中的状态信息决定何时退出循环*/
    if (count++ == 0)                /*如果是第一行的话，先打出行头*/
     printf ("\n%-10s %-20s %-10s %-10s\n", "Sno", "Sname", "Ssex", "Sage");
     printf ("%-10s %-20s %-10s %-10d\n", HSno, HSname, HSsex, HSage);
                                          /*打印查询结果*/
     printf ("UPDATE AGE(y/n)?"); /*询问用户是否要更新该学生的年龄*/
    do {
          scanf("%c",&yn);
     }
     while (yn != 'N' && yn != 'n' && yn != 'Y' && yn != 'y');
```

程序实例（续）

```
    if (yn == 'y' || yn == 'Y')              /*如果选择更新操作*/
          {
           printf("INPUT NEW AGE:");
           scanf("%d",&NEWAGE);          /*用户输入新年龄到主变量中*/
           EXEC SQL UPDATE Student          /*嵌入式SQL*/
                    SET Sage = :NEWAGE
                    WHERE CURRENT OF SX ;
          }              /*对当前游标指向的学生年龄进行更新*/
}

    EXEC SQL CLOSE SX;          /*关闭游标SX不再和查询结果对应*/
    EXEC SQL COMMIT WORK;                    /*提交更新*/
    EXEC SQL DISCONNECT TEST;                /*断开数据库连接*/
}
```

第八章 数据库编程

8.1 嵌入式SQL

8.2 存储过程和触发器

8.3 JDBC编程

8.2 存储过程

8.2.1 PL/SQL的块结构

8.2.2 变量常量的定义

8.2.3 控制结构

8.2.4 存储过程

8.2.5 小结

8.2.1 PL/SQL的块结构

- PL/SQL：
  - SQL的扩展
  - 增加了过程化语句功能
  - 基本结构是块
    - 块之间可以互相嵌套
    - 每个块完成一个逻辑操作

PL/SQL的块结构（续）

- **PL/SOL块的基本结构：**

1.定义部分

DECLARE

------变量、常量、游标、异常等

- 定义的变量、常量等只能在该基本块中使用
- 当基本块执行结束时，定义就不再存在

PL/SQL的块结构（续）

- **PL/SOL块的基本结构(续)：**

2.执行部分

BEGIN

------SQL语句、PL/SQL的流程控制语句

EXCEPTION

------异常处理部分

END;

8.2.2 变量常量的定义

1. **PL/SQL中定义变量的语法形式是:**

变量名 数据类型 ［ ［NOT NULL］:=初值表达式］或

变量名 数据类型 ［ ［NOT NULL］初值表达式］

2. **常量的定义类似于变量的定义:**

常量名 数据类型 CONSTANT :=常量表达式

常量必须要给一个值，并且该值在存在期间或常量的作用域内不能改变。如果试图修改它，PL/SQL将返回一个异常。

3. **赋值语句**

变量名称:=表达式

8.2.3 控制结构

- **PL/SQL 功能：**
  - 一、条件控制语句
  - 二、循环控制语句
  - 三、错误处理

控制结构（续）

一、 条件控制语句

IF-THEN， IF-THEN-ELSE和嵌套的IF语句

1. IF *condition* THEN
   Sequence_of_statements;
   END IF
2. IF condition THEN
   Sequence_of_statements1;
   ELSE
   Sequence_of_statements2;
   END IF;
3. 在THEN和ELSE子句中还可以再包括IF语句，即IF语句可以嵌套

控制结构（续）

二、循环控制语句

LOOP， WHILE-LOOP和FOR-LOOP

1. 最简单的循环语句LOOP

```
LOOP
    Sequence_of_statements;
END LOOP;
```

多数数据库服务器的PL/SQL都提供EXIT、BREAK或LEAVE等循环结束语句，保证LOOP语句块能够结束。

控制结构（续）

2. WHILE-LOOP

WHILE condition LOOP

Sequence_of_statements;

END LOOP;

- 每次执行循环体语句之前，首先对条件进行求值
- 如果条件为真，则执行循环体内的语句序列。
- 如果条件为假，则跳过循环并把控制传递给下一个语句

3. FOR-LOOP

FOR count IN ［REVERSE］*bound1* … *bound2* LOOP

Sequence_of_statements;

END LOOP;

控制结构（续）

三、错误处理

- 如果PL/SQL在执行时出现异常，则应该让程序在产生异常的语句处停下来，根据异常的类型去执行异常处理语句
- SQL标准对数据库服务器提供什么样的异常处理做出了建议，要求PL/SQL管理器提供完善的异常处理机制

8.2 存储过程

- 8.2.1 PL/SQL的块结构
- 8.2.2 变量常量的定义
- 8.2.3 控制结构
- 8.2.4 存储过程
- 8.2.5 小结

8.2.4 存储过程

- PL/SQL块类型：
  - 命名块：编译后保存在数据库中，可以被反复调用，运行速度较快。存储过程和函数是命名块
  - 匿名块：每次执行时都要进行编译，它不能被存储到数据库中，也不能在其他的PL/SQL块中调用

存储过程（续）

- 存储过程：由PL/SQL语句书写的过程，经编译和优化后存储在数据库服务器中，使用时只要调用即可。
- 一、存储过程的优点

1. 运行效率高
2. 降低了客户机和服务器之间的通信量
3. 方便实施企业规则

二、 存储过程的用户接口

1. 创建存储过程：

CREATE Procedure 过程名（［参数1，参数2，...］）

AS

<PL/SQL块>；

- 过程名：数据库服务器合法的对象标识
- 参数列表：用名字来标识调用时给出的参数值，必须指定值的数据类型。参数也可以定义输入参数、输出参数或输入/输出参数。默认为输入参数。
- 过程体：是一个<PL/SQL块>。包括声明部分和可执行语句部分

存储过程的用户接口（续）

［例11］ 利用存储过程来实现下面的应用: 从一个账户转指定数额的款项到另一个账户中。

```
CREATE PROCEDURE TRANSFER(inAccount INT， outAccount INT， amount FLOAT)
  AS DECLARE
       totalDeposit FLOAT;
  BEGIN                          /* 检查转出账户的余额 */
     SELECT total INTO totalDeposit
     FROM ACCOUNT WHERE ACCOUNTNUM=outAccount;
     IF totalDeposit IS NULL THEN    /* 账户不存在或账户中没有存款 */
        ROLLBACK;
        RETURN;
     END IF;
```

存储过程的用户接口（续）

```
IF totalDeposit < amount THEN        /* 账户账户存款不足 */
    ROLLBACK;
    RETURN;
END IF;
UPDATE account SET total=total-amount
WHERE ACCOUNTNUM=outAccount;
                                     /* 修改转出账户，减去转出额 */
UPDATE account SET total=total + amount WHERE
ACCOUNTNUM=inAccount;
                                     /* 修改转入账户，增加转出额 */
COMMIT;                              /* 提交转账事务 */
END;
```

存储过程的用户接口（续）

2. 执行存储过程

CALL/PERFORM Procedure 过程名(［参数1，参数2，...］)；

- 使用CALL或者PERFORM等方式激活存储过程的执行。
- 在PL/SQL中，数据库服务器支持在过程体中调用其他存储过程

［例12］从账户01003815868转一万元到01003813828账户中。

CALL Procedure TRANSFER(01003813828，01003815868，10000)；

触发器

- 触发器（Trigger）是用户定义在关系表上的一类由事件驱动的特殊过程
  - 由服务器自动激活
  - 可以进行更为复杂的检查和操作，具有更精细和更强大的数据控制能力

定义触发器

- CREATE TRIGGER语法格式

CREATE TRIGGER <触发器名>

{ BEFORE | AFTER } <触发事件> ON <表名>

FOR EACH { ROW | STATEMENT }

[WHEN <触发条件>]

<触发动作体>

定义触发器(续)

- 定义触发器的语法说明:
  - 1. 创建者：表的拥有者
  - 2. 触发器名
  - 3. 表名：触发器的目标表
  - 4. 触发事件：INSERT、DELETE、UPDATE
  - 5. 触发器类型
    - 行级触发器（FOR EACH ROW）
    - 语句级触发器（FOR EACH STATEMENT）

定义触发器(续)

- 例如,假设在TEACHER表上创建了一个AFTER UPDATE触发器。如果表TEACHER有1000行，执行如下语句：

```
UPDATE TEACHER SET Deptno=5;
```

  - 如果该触发器为语句级触发器，那么执行完该语句后，触发动作只发生一次
  - 如果是行级触发器，触发动作将执行1000次

定义触发器(续)

- 触发条件
  - 触发条件为真时触发动作体才执行
  - 可以省略WHEN触发条件
- 触发动作体
  - 触发动作体可以是一个匿名PL/SQL过程块
  - 也可以是对已创建存储过程的调用

定义触发器(续)

［例18］ 定义一个BEFORE行级触发器，为教师表Teacher定义完整性规则“教授的工资不得低于4000元，如果低于4000元，自动改为4000元”。

```
CREATE TRIGGER Insert_Or_Update_Sal
BEFORE INSERT OR UPDATE ON Teacher
                                    /*触发事件是插入或更新操作*/
FOR EACH ROW                        /*行级触发器*/
AS  BEGIN                           /*定义触发动作体，是PL/SQL过程块*/
    IF (new.Job='教授') AND (new.Sal < 4000) THEN
    new.Sal :=4000;
    END IF;
END;
```

定义触发器(续)

［例19］定义AFTER行级触发器，当教师表Teacher的工资发生变化后就自动在工资变化表Sal_log中增加一条相应记录

首先建立工资变化表Sal_log

```
CREATE TABLE Sal_log
  (Eno    NUMERIC(4)  references teacher(eno)，
   Sal     NUMERIC(7，2)，
   Username  char(10)，
   Date   TIMESTAMP
  )；
```

定义触发器(续)

[例19]（续）

```
CREATE TRIGGER Insert_Sal
AFTER INSERT ON Teacher        /*触发事件是INSERT*/
FOR EACH ROW
AS BEGIN
   INSERT INTO Sal_log VALUES( new.Eno，new.Sal，
                              CURRENT_USER，CURRENT_TIMESTAMP);
END;

CREATE TRIGGER Update_Sal
AFTER UPDATE ON Teacher        /*触发事件是UPDATE */
FOR EACH ROW
AS BEGIN
  IF (new.Sal <> old.Sal) THEN INSERT INTO Sal_log VALUES(
                  new.Eno，new.Sal，CURRENT_USER，CURRENT_TIMESTAMP);
  END IF;
END;
```

激活触发器

- 触发器的执行，是由触发事件激活的，并由数据库服务器自动执行
- 一个数据表上可能定义了多个触发器
  - 同一个表上的多个触发器激活时遵循如下的执行顺序：

    ❿执行该表上的BEFORE触发器；

    ❿激活触发器的SQL语句；

    ❿执行该表上的AFTER触发器。

激活触发器(续)

［例20］执行修改某个教师工资的SQL语句，激活上述定义的触发器。

UPDATE Teacher SET Sal=800 WHERE Ename='陈平';

执行顺序是：

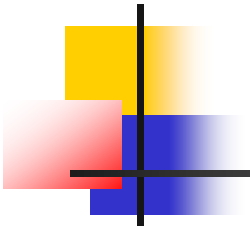
- 执行触发器Insert_Or_Update_Sal
- 执行SQL语句“UPDATE Teacher SET Sal=800 WHERE Ename='陈平';”
- 执行触发器Update_Sal

第八章 数据库编程

8.1 嵌入式SQL

8.2 存储过程和触发器

8.3 JDBC编程



JDBC

- JDBC基础
- JDBC驱动程序
- JDBC编程
- 示例

JDBC

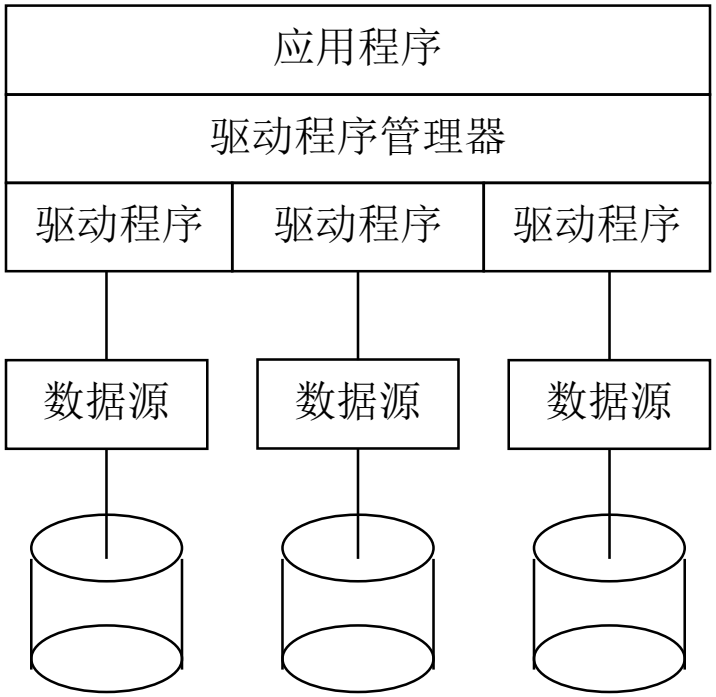
——基础

- JDBC(Java Database Connectivity)是一个独立于特定数据库管理系统的、通用的SQL数据库存取和操作的公共接口（一组API），定义了用来访问数据库的标准Java类库，使用这个类库可以以一种标准的方法、方便地访问数据库资源（在java.sql类包中）。
- JDBC为访问不同的数据库提供了一种统一的途径，象**ODBC(Open Database Connectivity)**一样，JDBC对开发者屏蔽了一些细节问题。
- JDBC的目标是使应用程序开发人员使用JDBC可以连接任何提供了JDBC驱动程序的数据库系统，这样就使得程序员无需对特定的数据库系统的特点有过多的了解，从而大大简化和加快了开发过程。

JDBC

——基础

开放数据库互连（ODBC）是为了实现异构数据库互连而由Microsoft公司推出的一种标准，它是一个单一的、公共的编程接口。ODBC提供不同的程序以存取不同的数据库，但只提供一种应用编程接口（API）给应用程序。



ODBC

ODBC的体系结构含有四个部件：

应用程序（Application）：执行ODBC函数的调用和处理，提交SQL语句并检索结果。

驱动程序管理器（Driver Manager）：为应用程序装载驱动程序。

驱动程序（Driver）：驱动程序是实现ODBC函数调用和同数据源交互作用的动态连接库，它执行ODBC函数调用，提交SQL请求到指定的数据源，并把结果返回给应用程序。如果需要，驱动程序也可改变应用程序的请求，以和特定的DBMS的语法匹配。

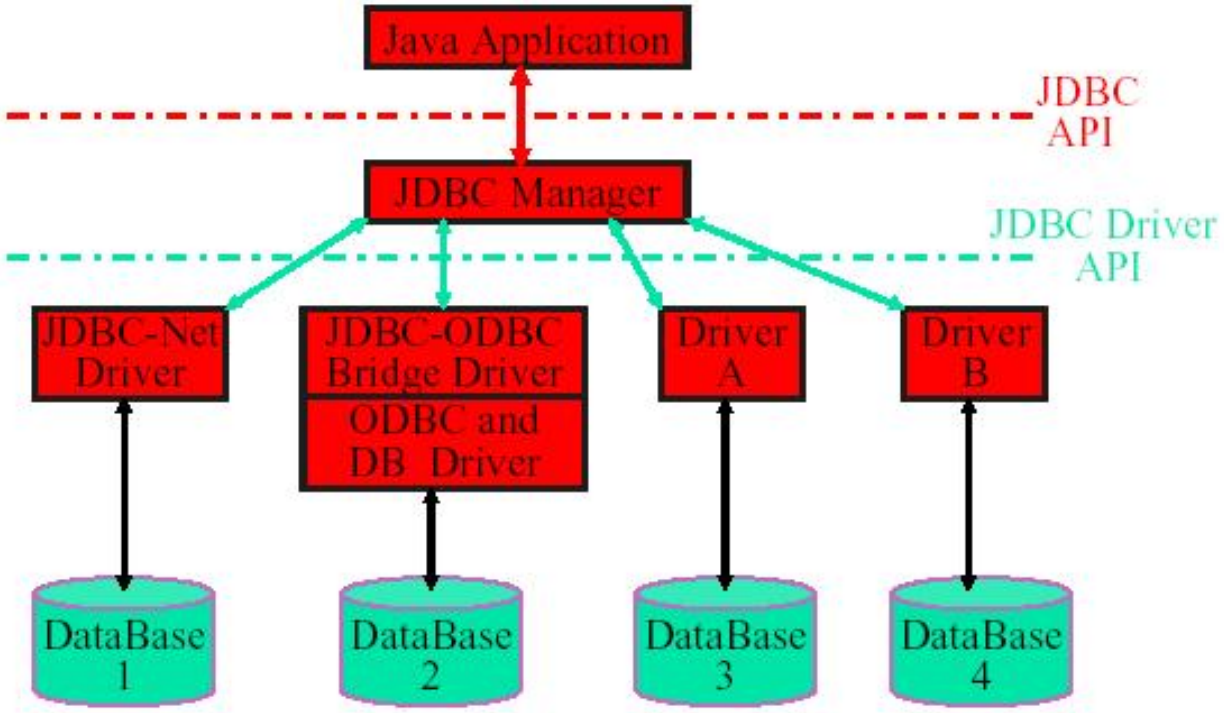
数据源（Data Source）：由用户需要存取的数据和与之相连的操作系统、DBMS及存取DBMS的网络平台组成。

JDBC

——基础

与ODBC相类似，JDBC接口（API）也包括两个层次：

- **面向应用的API**：Java API，抽象接口，供应用程序开发人员使用（连接数据库，执行SQL语句，获得结果）。
- **面向数据库的API**：Java Driver API，供开发商开发数据库驱动程序用。



与ODBC相比，JDBC没有了定制的“数据源”的概念，而是直接在应用程序中加载驱动程序并连接特定的数据库。

JDBC

——驱动程序

JDBC支持四种类型的驱动程序：

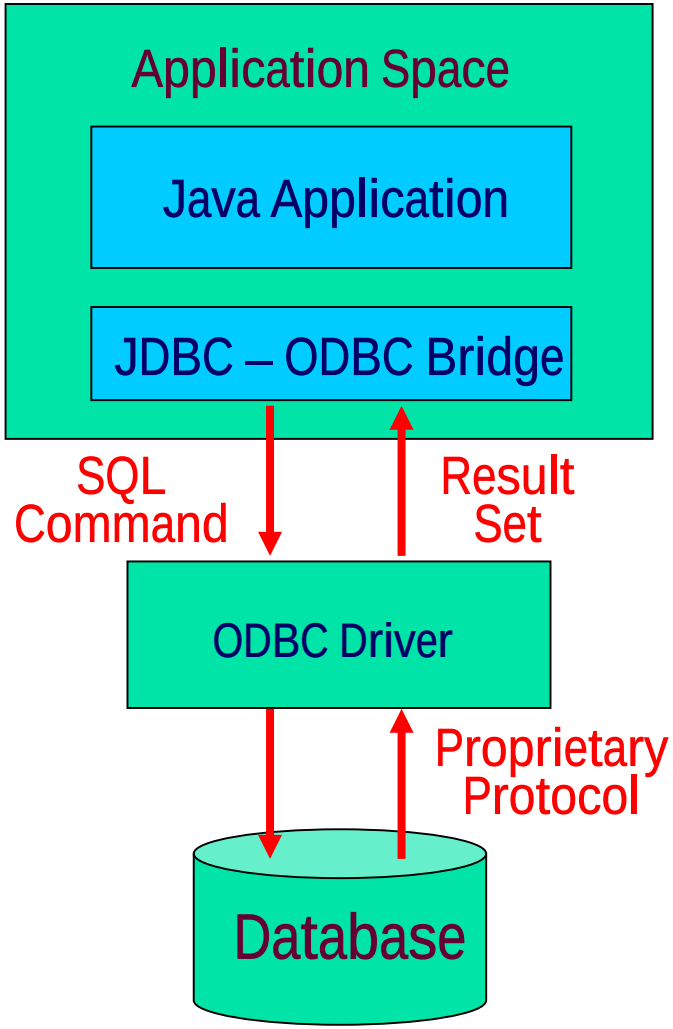
- JDBC-ODBC Bridge, plus ODBC driver (Type 1)
  - Simplest
  - JDBC methods -> Translate JDBC methods to ODBC methods -> ODBC to native methods -> Native methods API
- Native-API, partly Java driver (Type 2)
  - JDBC methods -> Map JDBC methods to **native methods** (calls to vendor library) -> Native methods API (vendor library)
- JDBC-net, pure Java driver (Type 3)
  - JDBC methods -> Translate to **Native API methods** through TCP/IP network -> Native API methods
- Native-protocol, pure Java driver (Type 4)
  - Java methods -> **Native methods** in Java

JDBC

——驱动程序

JDBC-ODBC Bridge, plus ODBC driver (Type 1)

- 由 Sun的Java2 JDK提供 (sun.jdbc.odbc.JdbcOdbcDriver)
- 通过ODBC驱动程序来获得对数据库的JDBC访问
- 必须先安装ODBC驱动程序和配置ODBC数据源。
- 仅当特定的数据库系统没有相应的JDBC驱动程序时使用。

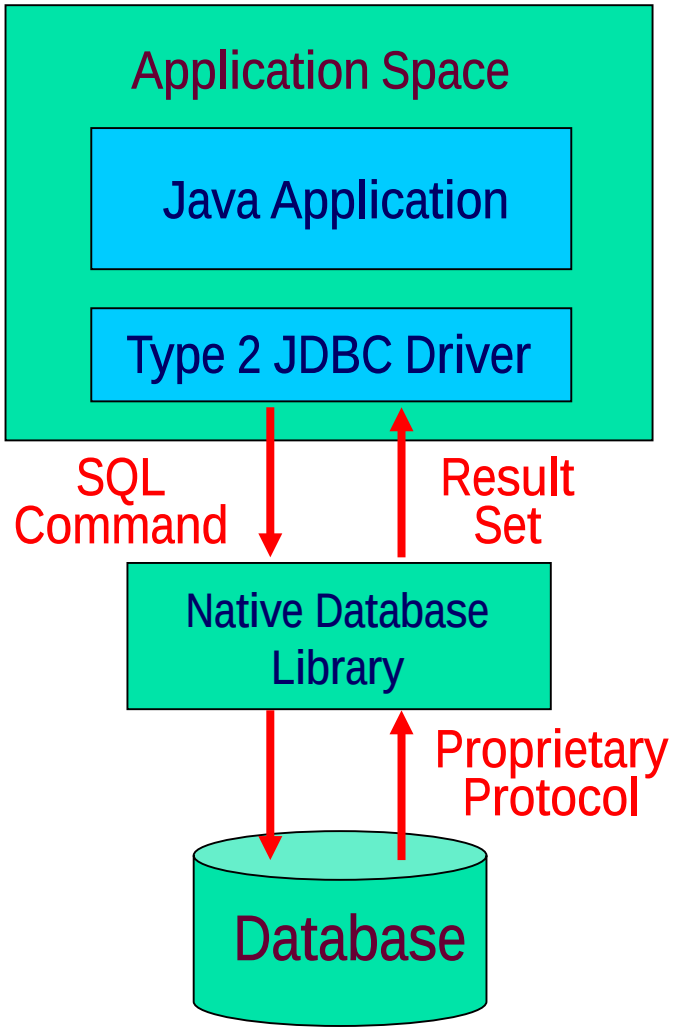


JDBC

——驱动程序

Native-API, partly Java driver (Type 2)

- Native-API driver 将JDBC命令转换为特定数据库系统的本地库方法。
- 与Type1相类似，必须先安装特定数据库的库方法（二进制代码，非Java）。

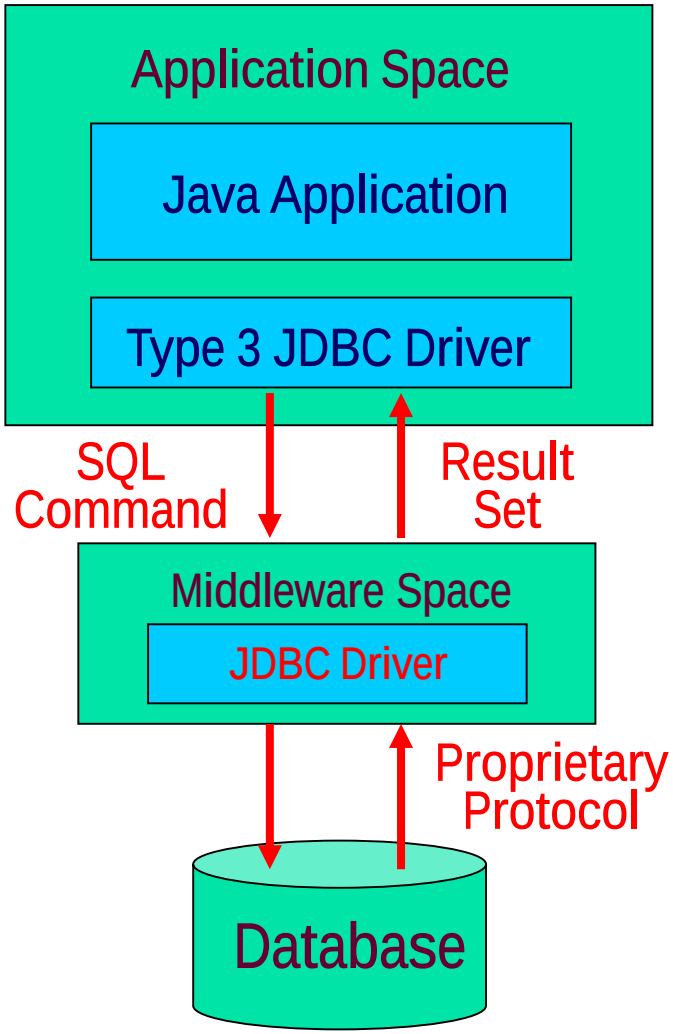


JDBC

——驱动程序

JDBC-net, pure Java driver (Type 3)

- 将JDBC命令转换为与数据库系统无关的网络协议，并发送给一个中间件服务器。
- 中间件服务器再将数据库系统无关的网络协议转换为特定数据库系统的协议，并发送给数据库系统。
- 从数据库系统获得的结果先发送给中间件服务器，并进而返回给应用程序。

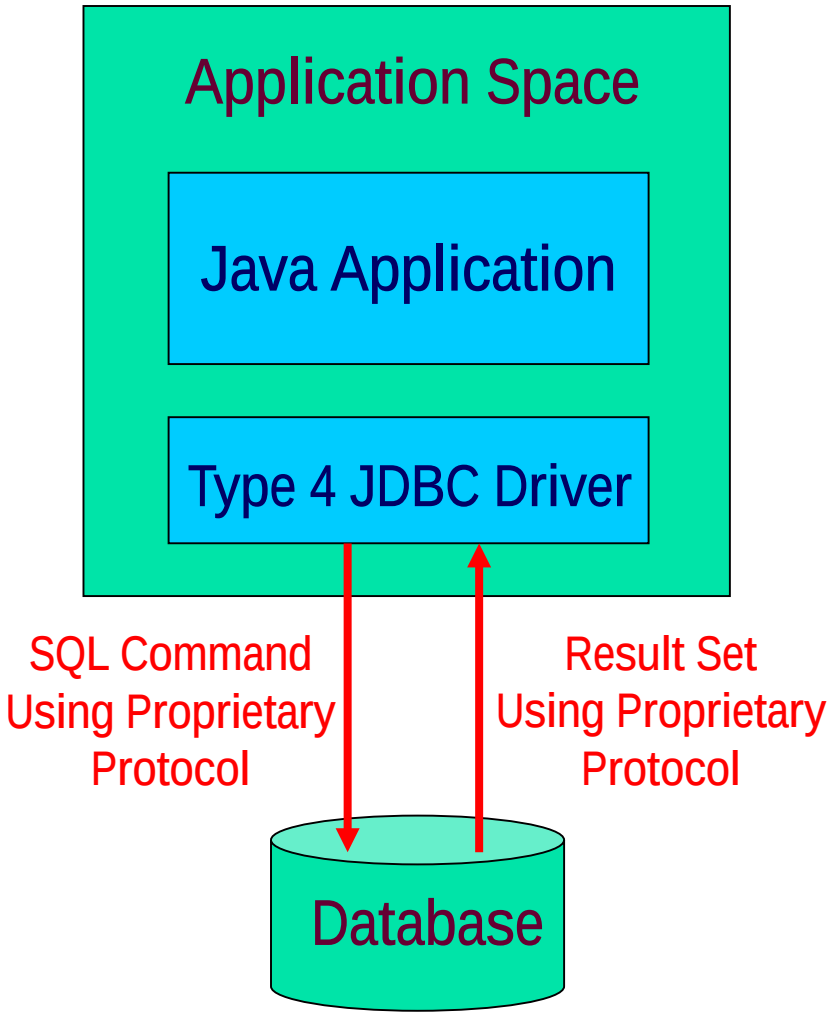


JDBC

——驱动程序

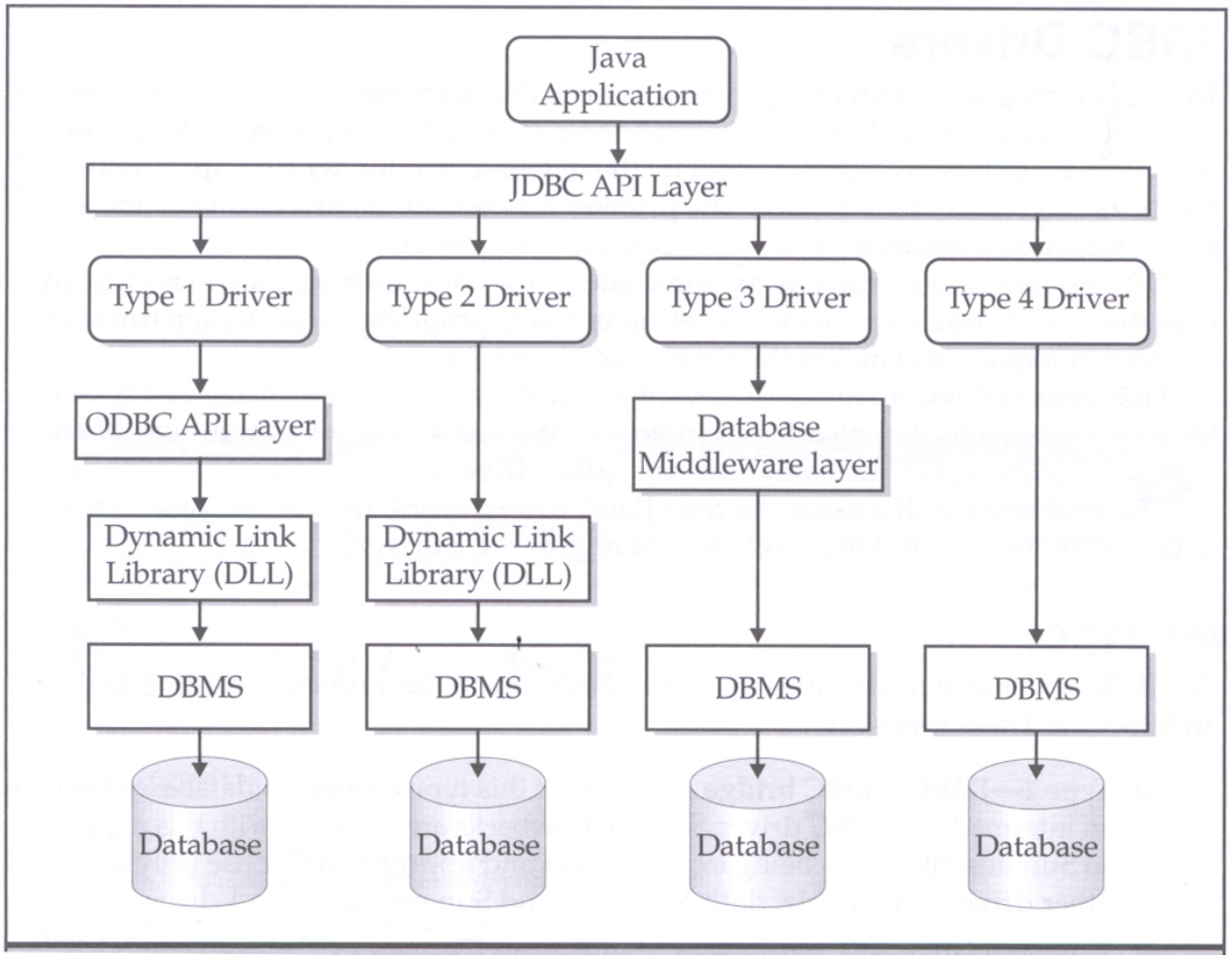
Native-protocol, pure Java driver (Type 4)

- 纯Java的驱动程序，直接与特定的数据库系统通信。
- 直接将JDBC命令转换为数据库系统的本地协议。
- 优点：没有中间的转换或者是中间件。
- 通常用于提高数据库访问的性能。

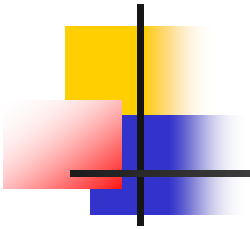


JDBC

——驱动程序


Java Application
JDBC API Layer
Type 1 Driver
Type 2 Driver
Type 3 Driver
Type 4 Driver
ODBC API Layer
Database Middleware layer
Dynamic Link Library (DLL)
Dynamic Link Library (DLL)
DBMS
DBMS
DBMS
DBMS
Database
Database
Database
Database

JDBC vs ODBC



1. ODBC是用C语言编写的，不是面向对象的；而JDBC是用Java编写的，是面向对象的。
2. ODBC难以学习，因为它把简单的功能与高级功能组合在一起，即便是简单的查询也会带有复杂的任选项；而JDBC的设计使得简单的事情用简单的做法来完成。
3. ODBC是局限于某一系统平台的，而JDBC提供Java与平台无关的解决方案。
4. 但也可以通过Java来操作ODBC，这可以采用JDBC-ODBC桥接方式来实现。

JDBC

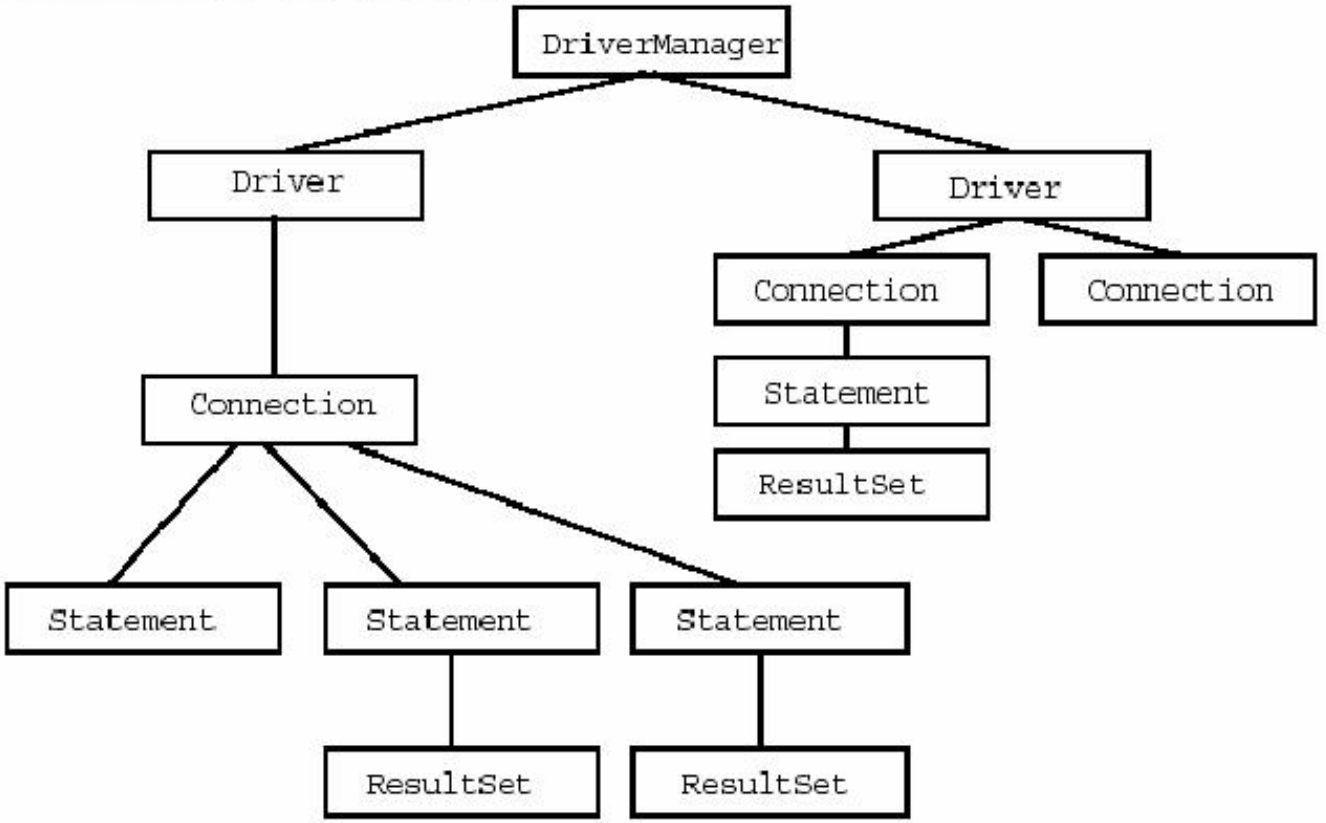
——基础：主要概念

- **Driver Manager（java.sql.DriverManager）**
  - 装载驱动程序，管理应用程序与驱动程序之间的连接。
- **Driver（由驱动程序开发商提供）**
  - 将应用程序的API请求转换为特定的数据库请求。
- **Connection（java.sql.Connection）**
  - 将应用程序连接到特定的数据库
- **Statement（java.sql.Statement）**
  - 在一个给定的连接中，用于执行一个静态的数据库SQL语句。
- **ResultSet（java.sql.ResultSet）**
  - SQL语句中心完后，返回的数据结果集（包括行、列）。
- **Metadata（java.sql.DatabaseMetadata; java.sql. ResultSetMetadata）**
  - 关于查询结果集、数据库和驱动程序的元数据信息。

JDBC

——编程

JDBC API 调用流程关系图



JDBC Flowchart

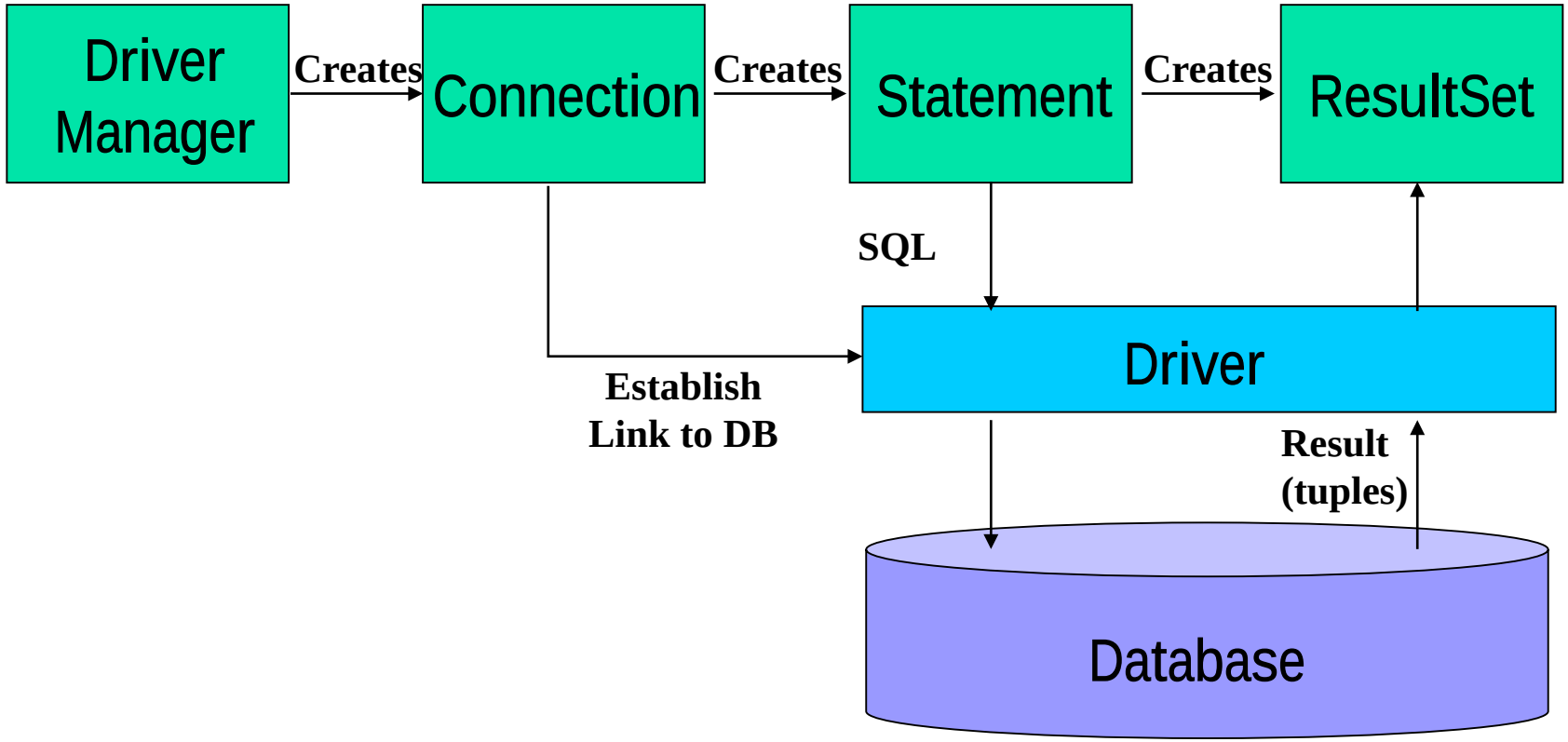
JDBC

——基础：基本工作步骤

- Import the necessary classes
- Load the JDBC driver
- Identify the database source
- Allocate a “Connection” object (create)
- Allocate a “Statement” object (create)
- Execute a query using the “Statement” object
- Retrieve data from the returned “ResultSet” object
- Close the “ResultSet” object
- Close the “Statement” object
- Close the “Connection” object

JDBC

——基础：基本工作步骤



JDBC

——基础：基本工作步骤

1. Load the JDBC driver class:

```
Class.forName("driverName");
```

2. Open a database connection:

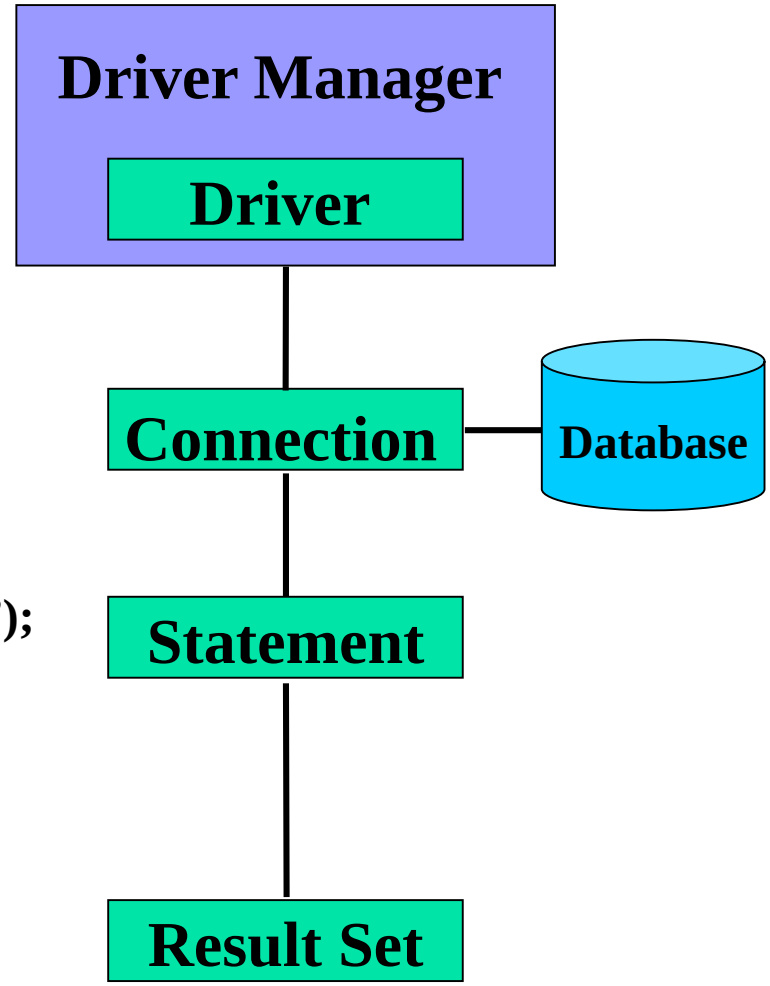
```
DriverManager.getConnection
("jdbc:xxx:datasource");
```

3. Issue SQL statements:

```
stmt = con.createStatement();
stmt.executeQuery ("Select * from myTable");
```

4. Process result set:

```
while (rs.next()) {
name = rs.getString("name");
amount = rs.getInt("amt"); }
```



JDBC

——编程

任何一个JDBC应用程序，都需要以下四个步骤：

- 加载JDBC驱动程序
- 建立与数据库的连接
- 进行数据库操作
- 关闭相关连接

JDBC

——编程：加载JDBC驱动程序

在应用程序中，有三种方法可以加载驱动程序：

- 利用System类的静态方法setProperty()

System.setProperty(“jdbc.drivers”,“sun.jdbc.odbc.JdbcOdbcDriver”);

- 利用Class类的静态方法forName()

Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);

Class.forName(“oracle.jdbc.driver.OracleDriver”);

- 直接创建一个驱动程序对象

new sun.jdbc.odbc.JdbcOdbcDriver();

JDBC

——编程：建立与数据库的连接

利用DriverManager类的静态方法getConnection()来获得与特定数据库的连接实例（Connection实例）。

- ✓ **Connection conn = DriverManager.getConnection(source);**
- ✓ **Connection conn = DriverManager.getConnection(source, user, pass);**

这三个参数都是String类型的，使用不同的驱动程序与不同的数据库建立连接时，source的内容是不同的，但其格式是一致的，都包括三个部分：

jdbc:*driverType*:*dataSource*

对于JDBC-ODBC Bridge，*driverType*为**odbc**, *dataSource*则为ODBC数据源：“jdbc:odbc:myDSN”。

对于其他类型的驱动程序，根据数据库系统的不同*driverType*和*dataSource*有不同的格式和内容。

JDBC

——编程：进行数据库操作

每执行一条SQL语句，都需要利用Connetcion实例的createStatement()方法来创建一个Statement实例。Statement的常用方法包括：

- 执行SQL INSERT, UPDATE 或 DELETE 等语句
  - int executeUpdate(String sql)
- 执行SQL SELECT语句
  - ResultSet executeQuery(String sql)
- 执行一个可能返回多个结果的SQL语句
  - boolean execute(String sql) (与其他方法结合起来来获得结果)

- Statement 中还有其他的方法来执行SQL语句。

JDBC

——编程：进行数据库操作

通过ResultSet来获得查询结果：

- ResultSet实例最初定位在结果集的第一行（记录）
- ResultSet提供了一些在结果集中定位的方法，如next()等。
- ResultSet提供了一些方法来获得当前行中的不同字段的值，getXXX()。

- ResultSet中还提供了有关方法，来修改结果集，并提交到数据库中去。

JDBC

——编程：进行数据库操作

ResultSet常用getXXX方法

返回值类型	方法名称
boolean	getBoolean()
byte	getByte()
byte[]	getBytes()
java.sql.Date	getDate()
double	getDouble()
float	getFloat()
int	getInt()
long	getLong()
Object	getObject()
short	getShort()
java.lang.String	getString()
java.sql.Time	getTime()

参数：

int colIndex

或

String colName

JDBC

——编程：进行数据库操作

通过ResultSetMetadata来获得查询结果的元数据信息：

- ResultSet提供了一个方法getMetadata()来获得结果集的元数据信息，它返回的是一个ResultSetMetadata实例。
- 通过ResultSetMetadata实例，就可以获得结果集中字段的详细信息，如字段总数，每个字段的名称、类型等。
  - **getColumnCount() // # of columns in the row**
  - **getColumnName(*i*) // returns column name**
  - **getColumnType (*i*) // returns column data type**
  - **getColumnLabel (*i*) //suggested label for a column when print**
  - **getTableName() //returns the name of the table**
- 确定了字段类型，获取字段数据时，就可以明确如何使用ResultSet中的getXXX()方法了。

JDBC

——编程：进行数据库操作

通过DatabaseMetadata来获得数据库的元数据信息：

- Connection提供了一个方法getMetadata()来获得数据库的元数据信息，它返回的是一个DatabaseMetadata实例。
- 通过DatabaseMetadata实例，就可以获得数据库的各种信息，如数据库厂商信息、版本信息、数据表数目、每个数据表名称等。
  - **getDatabaseProductName()**
  - **getDatabaseProductVersion()**
  - **getDriverName()**
  - **getTables()**