# 数据库系统概论
# An Introduction to Database System

## 第八章 数据库编程

中国人民大学信息学院

# 第八章 数据库编程

8.1 嵌入式 SQL

8.2 存储过程

8.3 ODBC 编程

# 8.3 ODBC 编程

- ODBC 优点：
  - 移植性好
  - 能同时访问不同的数据库
  - 共享多个数据资源

# 8.3 ODBC 编程

- 8.3.1 数据库互连概述
- 8.3.2 ODBC 工作原理概述
- 8.3.3 ODBC API 基础
- 8.3.4 ODBC 的工作流程
- 8.3.5 小结

# 8.3.1 数据库互连概述

- **ODBC 产生的原因：**
  - 由于不同的数据库管理系统的存在，在某个 RDBMS 下编写的应用程序就不能在另一个 RDBMS 下运行
  - 许多应用程序需要共享多个部门的数据资源，访问不同的 RDBMS

# 数据库互连概述（续）

- ODBC：
  - 是微软公司开放服务体系 (Windows Open Services Architecture， WOSA) 中有关数据库的一个组成部分
  - 提供了一组访问数据库的标准 API
- ODBC 约束力：
  - 规范应用开发
  - 规范 RDBMS 应用接口

# 8.3 ODBC 编程

- 8.3.1 数据库互连概述
- 8.3.2 ODBC 工作原理概述
- 8.3.3 ODBC API 基础
- 8.3.4 ODBC 的工作流程
- 8.3.5 小结

# 8.3.2 ODBC 工作原理概述

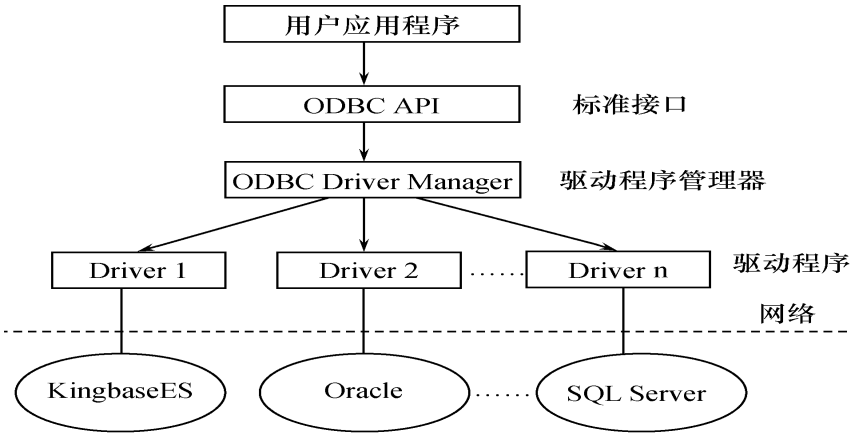
- ODBC 应用系统的体系结构：

一、 用户应用程序

二、 驱动程序管理器

三、 数据库驱动程序

四、 ODBC 数据源管理



# 一、 应用程序

- **ODBC 应用程序包括的内容：**
  - 请求连接数据库；
  - 向数据源发送 SQL 语句；
  - 为 SQL 语句执行结果分配存储空间，定义所读取的数据格式；
  - 获取数据库操作结果，或处理错误；
  - 进行数据处理并向用户提交处理结果；
  - 请求事务的提交和回滚操作；
  - 断开与数据源的连接。

# 二、驱动程序管理器

- 驱动程序管理器：用来管理各种驱动程序
  - 包含在 ODBC32.DLL 中
  - 管理应用程序和驱动程序之间的通信
  - 建立、配置或删除数据源并查看系统当前所安装的数据库 ODBC 驱动程序
  - 主要功能：
    - 装载 ODBC 驱动程序
    - 选择和连接正确的驱动程序
    - 管理数据源
    - 检查 ODBC 调用参数的合法性
    - 记录 ODBC 函数的调用等

# 三、数据库驱动程序

- ODBC 通过驱动程序来提供应用系统与数据库平台的独立性
- ODBC 应用程序不能直接存取数据库
  - 其各种操作请求由驱动程序管理器提交给某个 RDBMS 的 ODBC 驱动程序
  - 通过调用驱动程序所支持的函数来存取数据库。
  - 数据库的操作结果也通过驱动程序返回给应用程序。
  - 如果应用程序要操纵不同的数据库，就要动态地链接到不同的驱动程序上。

# 数据库驱动程序（续）

- ODBC 驱动程序类型：
  - 单束
    - 数据源和应用程序在同一台机器上
    - 驱动程序直接完成对数据文件的 I/O 操作
    - 驱动程序相当于数据管理器
  - 多束
    - 支持客户机 / 服务器、客户机 / 应用服务器 / 数据库服务器等网络环境下的数据访问
    - 由驱动程序完成数据库访问请求的提交和结果集接收
    - 应用程序使用驱动程序提供的结果集管理接口操纵执行后的结果数据

# 四、 ODBC 数据源管理

- 数据源：是最终用户需要访问的数据，包含了数据库位置和数据库类型等信息，是一种数据连接的抽象
- 数据源对最终用户是透明的
  - ODBC 给每个被访问的数据源指定唯一的数据源名（Data Source Name，简称 DSN），并映射到所有必要的、用来存取数据的低层软件
  - 在连接中，用数据源名来代表用户名、服务器名、所连接的数据库名等
  - 最终用户无需知道 DBMS 或其他数据管理软件、网络以及有关 ODBC 驱动程序的细节

# ODBC 数据源管理（续）

例如，假设某个学校在 MS SQL Server 和 KingbaseES 上创建了两个数据库：学校人事数据库和教学科研数据库。

- 学校的信息系统要从这两个数据库中存取数据
- 为方便与两个数据库连接，为学校人事数据库创建一个数据源名 PERSON，为教学科研数据库创建一个名为 EDU 的数据源。
- 当要访问每一个数据库时，只要与 PERSON 和 EDU 连接即可，不需要记住使用的驱动程序、服务器名称、数据库名

# 8.3 ODBC 编程

- 8.3.1 数据库互连概述
- 8.3.2 ODBC 工作原理概述
- 8.3.3 ODBC API 基础
- 8.3.4 ODBC 的工作流程
- 8.3.5 小结

# 8.3.3 ODBC API 基础

- ODBC 应用程序接口的一致性
  - API 一致性
    - API 一致性级别有核心级、扩展 1 级、扩展 2 级
  - 语法一致性
    - 语法一致性级别有最低限度 SQL 语法级、核心 SQL 语法级、扩展 SQL 语法级

# ODBC API 基础（续）

- 一、 函数概述
- 二、 句柄及其属性
- 三、 数据类型

# 一、 函数概述

- **ODBC 3.0 标准提供了 76 个函数接口：**
  - 分配和释放环境句柄、连接句柄、语句句柄；
  - 连接函数（ SQLDriverconnect 等）；
  - 与信息相关的函数（如获取描述信息函数 SQLGetinfo 、 SQLGetFuction ）；
  - 事务处理函数（如 SQLEndTran ）；
  - 执行相关函数（ SQLExecdirect 、 SQLExecute 等）；
  - 编目函数， ODBC 3.0 提供了 11 个编目函数如 SQLTables 、 SQLColumn 等，应用程序可以通过对编目函数的调用来获取数据字典的信息如权限、表结构等

# 函数概述（续）

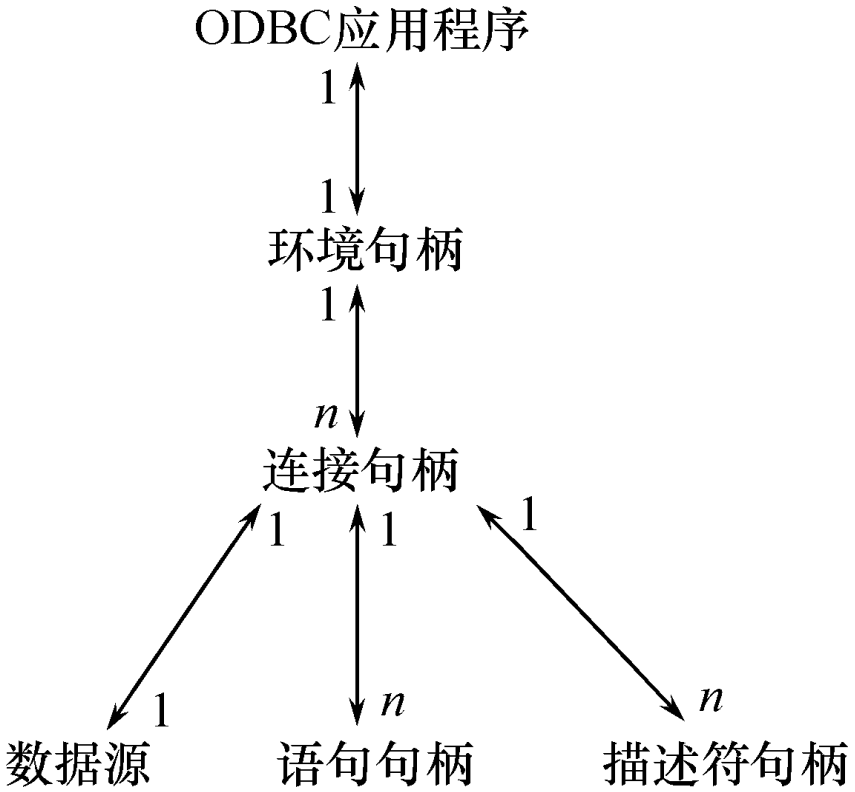
- ODBC 1.0 和 ODBC 2.x 、 ODBC 3.x 函数使用上有很多差异
- MFC ODBC 对较复杂的 ODBC API 进行了封装，提供了简化的调用接口

# 二、 句柄及其属性

- 句柄是 32 位整数值，代表一个指针
- ODBC 3.0 中句柄分类:
  - 环境句柄
  - 连接句柄
  - 语句句柄
  - 描述符句柄

# 句柄及其属性（续）

- 应用程序句柄之间的关系




应用程序句柄之间的关系

# 句柄及其属性（续）

1. 每个 ODBC 应用程序需要建立一个 ODBC 环境，分配一个环境句柄，存取数据的全局性背景如环境状态、当前环境状态诊断、当前在环境上分配的连接句柄等；
2. 一个环境句柄可以建立多个连接句柄，每一个连接句柄实现与一个数据源之间的连接；

# 句柄及其属性（续）

3. 在一个连接中可以建立多个语句句柄，它不只是一个SQL 语句，还包括 SQL 语句产生的结果集以及相关的信息等；

4. 在 ODBC 3.0 中又提出了描述符句柄的概念，它是描述SQL 语句的参数、结果集列的元数据集合。

# 三、 数据类型

- ODBC 数据类型:
  - SQL 数据类型: 用于数据源
  - C 数据类型： 用于应用程序的 C 代码

- 应用程序可以通过 SQLGetTypeInfo 来获取不同的驱动程序对于数据类型的支持情况

# 数据类型（续）

SQL 数据类型和 C 数据类型之间的转换规则

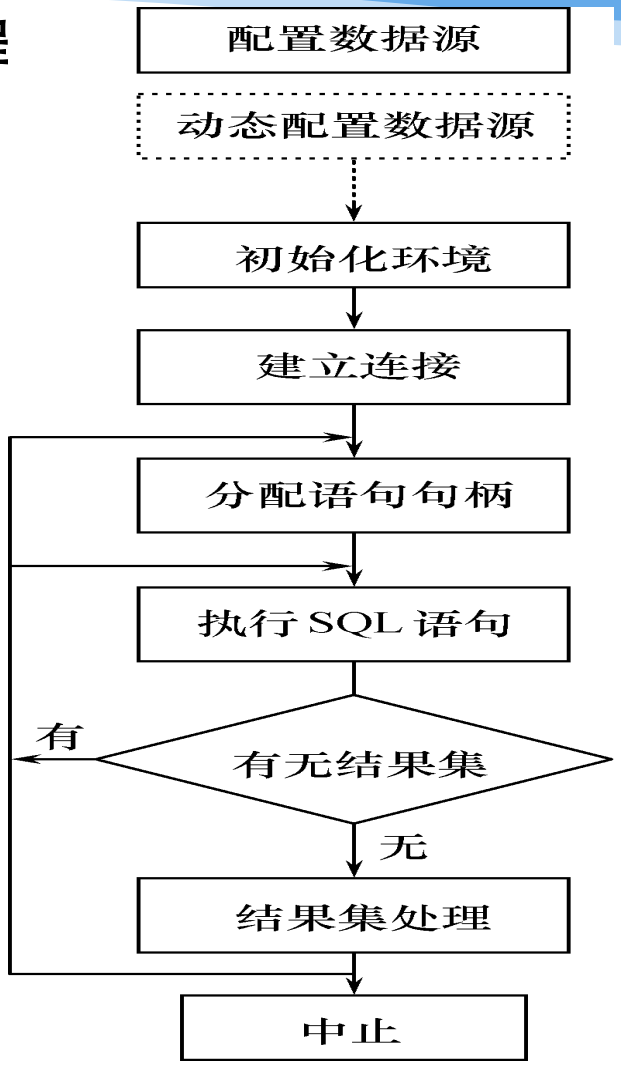
| SQL | □□□□ | C □□□□ |
| --- | --- | --- |
| SQL □□□□ | □□□□□□□ | □□□□□□□□□□□<br>□□□ SQLBindparameter □ |
| C □□□□ | □□□□□□□□□□□<br>□□□□□ SQLBindcol □ | □□□□□□□□□□ |

# 8.3 ODBC 编程

- 8.3.1 数据库互连概述
- 8.3.2 ODBC 工作原理概述
- 8.3.3 ODBC API 基础
- 8.3.4 ODBC 的工作流程
- 8.3.5 小结

# 8.3.4 ODBC 的工作流程

- ODBC 的工作流程



# ODBC 的工作流程（续）

［例 13 ］将 KingbaseES 数据库中 Student 表的数据备份到 SQL SERVER 数据库中。

- 该应用涉及两个不同的 RDBMS 中的数据源
- 使用 ODBC 来开发应用程序，只要改变应用程序中连接函数（ SQLConnect ）的参数，就可以连接不同 RDBMS 的驱动程序，连接两个数据源

# ODBC 的工作流程（续）

- 在应用程序运行前，已经在 KingbaseES 和 SQL SERVER 中分别建立了 STUDENT 关系表

```
CREATE TABLE Student
    （ Sno   CHAR （9）  PRIMARY KEY ,
      Sname  CHAR （20）  UNQUE
      Ssex   CHAR （2）,
      Sage   SMALLINT ,
      Sdept  CHAR （20）
    ）；
```

# ODBC 的工作流程（续）

- 应用程序要执行的操作是:
  - 在 KingbaseES 上执行 SELECT * FROM STUDENT ;
  - 把获取的结果集，通过多次执行

    INSERT INTO STUDENT

    （Sno， Sname， Ssex， Sage， Sddept） VALUES

    （？， ？， ？， ？， ？）；
  - 插入到 SQL SERVER 的 STUDENT 表中

# ODBC 的工作流程（续）

- 操作步骤：

  一、 配置数据源

  二、 初始化环境

  三、 建立连接

  四、 分配语句句柄

  五、 执行 SQL 语句

  六、 结果集处理

  七、 中止处理

# 一、 配置数据源

- 配置数据源两种方法：

  ⑴ 运行数据源管理工具来进行配置；

  ⑵ 使用 Driver Manager 提供的 ConfigDsn 函数来增加、修

  改或删除数据源

- 在［例 13 ］中，采用了第一种方法创建数据源。因为要同时用到 KingbaseES 和 SQL Server ， 所以分别建立两个数据源，将其取名为 KingbaseES ODBC 和 SQLServer 。

# 配置数据源（续）

［例 13 ］创建数据源的详细过程

```
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <Sqltypes.h>
#define SNO_LEN 30
#define NAME_LEN 50
#define DEPART_LEN 100
#define SSEX_LEN 5
```

# 配置数据源（续）

［例 13 ］创建数据源 --- 第一步：定义句柄和变量

```
int main()
{
    /* Step 1  定义句柄和变量  */
    // 以 king 开头的表示的是连接 KingbaseES 的变量
    // 以 server 开头的表示的是连接 SQLSERVER 的变量
    SQLHENV    kinghenv ,   serverhenv;                    // 环境句柄
    SQLHDBC                 kinghdbc ,  serverhdbc;                 // 连接句柄
    SQLHSTMT                kinghstmt ,  serverhstmt;   // 语句句柄
    SQLRETURN               ret;
    SQLCHAR  sName  [ NAME_LEN ] ,  sDepart  [ DEPART_LEN ] ,
    sSex  [ SSEX_LEN ] ,   sSno  [ SNO_LEN ] ;
    SQLINTEGER   sAge;
    SQLINTEGER  cbAge=0 ,   cbSno=SQL_NTS ,   cbSex=SQL_NTS ,
    cbName=SQL_NTS ,  cbDepart=SQL_NTS;
```

# 二、初始化环境

- 没有和具体的驱动程序相关联，由 Driver Manager 来进行控制，并配置环境属性
- 应用程序通过调用连接函数和某个数据源进行连接后， Driver Manager 才调用所连的驱动程序中的 SQLAllocHandle，来真正分配环境句柄的数据结构

# 初始化环境代码

［例 13 ］创建数据源 --- 第二步：初始化环境

```
/* Step 2  初始化环境  */
    ret=SQLAllocHandle(SQL_HANDLE_ENV ,  SQL_NULL_HANDLE
    ,   &kinghenv);
    ret=SQLAllocHandle(SQL_HANDLE_ENV ,  SQL_NULL_HANDLE
    ,   &serverhenv);
    ret=SQLSetEnvAttr(kinghenv ,        SQL_ATTR_ODBC_VERSION
    ,               (void*)SQL_OV_ODBC3 ,   0);
    ret=SQLSetEnvAttr(serverhenv ,      SQL_ATTR_ODBC_VERSION
    ,               (void*)SQL_OV_ODBC3 ,   0);
```

# 三、 建立连接

- 应用程序调用 SQLAllocHandle 分配连接句柄，通过 SQLConnect 、 SQLDriverConnect 或 SQLBrowseConnect 与数据源连接
- SQLConnect 连接函数，输入参数为：
  - 配置好的数据源名称
  - 用户 ID
  - 口令
- [ 例 13] 中 KingbaseES ODBC 为数据源名字， SYSTEM 为用户名， MANAGER 为用户密码

# 建立连接代码

［例 13 ］创建数据源 --- 第三步：建立连接

```
/* Step 3 : 建立连接  */
ret=SQLAllocHandle(SQL_HANDLE_DBC ,  kinghenv ,  &kinghdbc);
ret=SQLAllocHandle(SQL_HANDLE_DBC ,  serverhenv ,
    &serverhdbc);
ret=SQLConnect(kinghdbc ,  "KingbaseES ODBC" ,  SQL_NTS ,
    "SYSTEM" ,  SQL_NTS ,  "MANAGER" ,  SQL_NTS);
if (!SQL_SUCCEEDED(ret))// 连接失败时返回错误值
    return-1;
ret=SQLConnect(serverhdbc ,  "SQLServer" ,  SQL_NTS ,  "sa" ,
    SQL_NTS ,  "sa" ,  SQL_NTS);
if (!SQL_SUCCEEDED(ret) )                    // 连接失败时返回错误值
    return -1;
```

# 四、分配语句句柄

- 处理任何 SQL 语句之前，应用程序还需要首先分配一个语句句柄
- 语句句柄含有具体的 SQL 语句以及输出的结果集等信息
- [ 例 13] 中分配了两个语句句柄:
  - 一个用来从 KingbaseES 中读取数据产生结果集（ kinghstmt ）
  - 一个用来向 SQLSERVER 插入数据（ serverhstmt ）
- 应用程序还可以通过 SQLtStmtAttr 来设置语句属性（也可以使用默认值）

# 分配语句句柄代码

［例 13 ］创建数据源 --- 第四步

```
/* Step 4 : 初始化语句句柄 */
ret=SQLAllocHandle(SQL_HANDLE_STMT ,  kinghdbc ,
    &kinghstmt);
ret=SQLSetStmtAttr(kinghstmt ,  SQL_ATTR_ROW_BIND_TYPE ,
    (SQLPOINTER)
   SQL_BIND_BY_COLUMN ,  SQL_IS_INTEGER );
ret=SQLAllocHandle(SQL_HANDLE_STMT ,  serverhdbc ,
    &serverhstmt);
```

# 五、 执行 SQL 语句

- 应用程序处理 SQL 语句的两种方式：
  - 预处理（SQLPrepare 、 SQLExecute 适用于语句的多次执行）
  - 直接执行（SQLExecdirect ）
- 如果 SQL 语句含有参数，应用程序为每个参数调用 SQLBindParameter ， 并把它们绑定至应用程序变量
- 应用程序可以直接通过改变应用程序缓冲区的内容从而在程序中动态的改变 SQL 语句的具体执行

# 执行 SQL 语句（续）

- 应用程序根据语句的类型进行的处理
  - 有结果集的语句（ select 或是编目函数），则进行结果集处理。
  - 没有结果集的函数，可以直接利用本语句句柄继续执行新的语句或是获取行计数（本次执行所影响的行数）之后继续执行。
- 在 [ 例 13] 中，使用 SQLExecdirect 获取 KingbaseES 中的结果集，并将结果集根据各列不同的数据类型绑定到用户程序缓冲区
- 在插入数据时，采用了预编译的方式，首先通过 SQLPrepare 来预处理 SQL 语句，将每一列绑定到用户缓冲区
- 应用程序可以直接修改结果集缓冲区的内容

# 程序源码

［例 13 ］创建数据源 --- 第五步：执行 SQL 语句

```
/* Step 5 : 两种方式执行语句 */
/* 预编译带有参数的语句 */
ret=SQLPrepare(serverhstmt , "INSERT INTO
    STUDENT(SNO , SNAME , SSEX , SAGE , SDEPT) VALUES
    (? , ? , ? , ? , ?)" , SQL_NTS);
if (ret==SQL_SUCCESS || ret==SQL_SUCCESS_WITH_INFO)
{
ret=SQLBindParameter(serverhstmt , 1 , SQL_PARAM_INPUT , SQL_C_CHA
    R , SQL_CHAR , SNO_LEN , 0 , sSno , 0 , &cbSno);
ret=SQLBindParameter(serverhstmt , 2 , SQL_PARAM_INPUT ,
    SQL_C_CHAR , SQL_CHAR , NAME_LEN , 0 , sName , 0 ,
    &cbName);
ret=SQLBindParameter(serverhstmt , 3 , SQL_PARAM_INPUT , SQL_C_CHA
    R , SQL_CHAR , 2 , 0 , sSex , 0 , &cbSex);
ret=SQLBindParameter(serverhstmt , 4 , SQL_PARAM_INPUT ,
SQL_C_LONG , SQL_INTEGER , 0 , 0 , &sAge , 0 , &cbAge);
```

# 程序源码

```
ret=SQLBindParameter(serverhstmt , 5 , SQL_PARAM_INPUT , SQL_C_CH
    AR ,
SQL_CHAR , DEPART_LEN , 0 , sDepart , 0 , &cbDepart);
}
  /* 执行 SQL 语句 */
ret=SQLExecDirect(kinghstmt , "SELECT * FROM STUDENT" , SQL_NTS);
if (ret==SQL_SUCCESS || ret==SQL_SUCCESS_WITH_INFO)
{
    ret=SQLBindCol(kinghstmt , 1 ,
       SQL_C_CHAR , sSno , SNO_LEN , &cbSno);
    ret=SQLBindCol(kinghstmt , 2 , SQL_C_CHAR , sName , NAME_LEN
       , &cbName);
    ret=SQLBindCol(kinghstmt , 3 , SQL_C_CHAR , sSex , SSEX_LEN
       , &cbSex);
    ret=SQLBindCol(kinghstmt , 4 , SQL_C_LONG , &sAge , 0 ,
       &cbAge);
    ret=SQLBindCol(kinghstmt , 5 , SQL_C_CHAR , sDepart ,
       DEPART_LEN , &cbDepart);
}
```

# 六、结果集处理

- 应用程序可以通过 SQLNumResultCols 来获取结果集中的列数
- 通过 SQLDescribeCol 或是 SQLColAttrbute 函数来获取结果集每一列的名称、数据类型、精度和范围

# 结果集处理（续）

- ODBC 中使用游标来处理结果集数据
- ODBC 中游标类型：
  - forward-only 游标，是 ODBC 的默认游标类型
  - 可滚动（ scroll ）游标：
    - 静态（ static ）
    - 动态（ dynamic ）
    - 码集驱动（ keyset-driven ）
    - 混合型（ mixed ）

# 结果集处理（续）

- 结果集处理步骤：
  - ODBC 游标打开方式不同于嵌入式 SQL，不是显式声明而是系统自动产生一个游标（Cursor），当结果集刚刚生成时，游标指向第一行数据之前
  - 应用程序通过 SQLBindCol，把查询结果绑定到应用程序缓冲区中，通过 SQLFetch 或是 SQLFetchScroll 来移动游标获取结果集中的每一行数据
  - 对于如图像这类特别的数据类型当一个缓冲区不足以容纳所有的数据时，可以通过 SQLGetdata 分多次获取
  - 最后通过 SQLClosecursor 来关闭游标

# 程序源码代码

［例 13 ］创建数据源 --- 第六步：结果集处理

```
/* Step 6 ： 处理结果集并执行预编译后的语句 */
 while ( (ret=SQLFetch(kinghstmt) ) !=SQL_NO_DATA_FOUND)
 {
     if(ret==SQL_ERROR)  printf("Fetch error ＼ n");
     else
             ret=SQLExecute(serverhstmt);
}
```

# 七、中止处理

- **应用程序中止步骤：**
  - 首先释放语句句柄
  - 释放数据库连接
  - 与数据库服务器断开
  - 释放 ODBC 环境

# 中止处理代码

［例 13 ］创建数据源 --- 第七步： 中止处理

```
    /* Step 7  中止处理 */
    SQLFreeHandle(SQL_HANDLE_STMT ,  kinghstmt);
    SQLDisconnect(kinghdbc);
    SQLFreeHandle(SQL_HANDLE_DBC ,  kinghdbc);
    SQLFreeHandle(SQL_HANDLE_ENV ,  kinghenv);
    SQLFreeHandle(SQL_HANDLE_STMT ,  serverhstmt);
    SQLDisconnect(serverhdbc);
    SQLFreeHandle(SQL_HANDLE_DBC ,  serverhdbc);
    SQLFreeHandle(SQL_HANDLE_ENV ,  serverhenv);
    return 0;
}
```

# 8.3 ODBC 编程

- 8.3.1 数据库互连概述
- 8.3.2 ODBC 工作原理概述
- 8.3.3 ODBC API 基础
- 8.3.4 ODBC 的工作流程
- 8.3.5 小结

# 8.3.5 小结

- ODBC 目的：为了提高应用系统与数据库平台的独立性，使得应用系统的移植变得容易
- ODBC 优点：
  - 使得应用系统的开发与数据库平台的选择、数据库设计等工作并行进行
  - 方便移植
  - 大大缩短整个系统的开发时间

# 下课了。。。


休息一会儿。。。