華中科技大學

数据库系统原理

第8章　数据库编程

分级通关平台，QQ群

基础篇

第1章 绪论
第2章 关系数据库*3
第3章 标准语言SQL*3
第4章 数据库安全性
第5章 数据库完整性
实验1

设计与应用开发篇

第6章 关系数据理论*2
第7章 数据库设计
实验2
第8章 数据库编程

系统篇

第9章 *关系查询处理和优化
实验3
第10章 数据库恢复技术
第11章 并发控制
实验4

内容提要

- ✓ 嵌入式SQL
- ✓ 过程化SQL
- ✓ 存储过程和函数
- ✓ ODBC编程

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--必要的扩充
  - 事务处理应用需要高级语言
- 嵌入式SQL是将SQL语句嵌入程序设计语言（宿主语言，简称主语言）中，如C、C++、Java。
- 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，C语句格式：
  - EXEC SQL <SQL语句>;

嵌入式SQL—实例

- [例8.1] 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[20];
    char Hsno[9];
    char Hsname[20];
    char Hssex[2];
    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*/
```

嵌入式SQL—实例

- 将SQL嵌入到高级语言中混合编程，如何交互数据？
  - 1）向主语言传递SQL语句的执行状态信息，使主语言能够据此控制程序流程，主要用SQL通信区实现
  - 2）主语言向SQL语句提供参数，主要用主变量实现
  - 3）将SQL语句查询数据库的结果交主语言处理，主要用主变量和游标实现

✓ SQLCA： SQL Communication Area，SQLCA是一个数据结构

➢ 描述系统当前工作状态， 描述运行环境

- 定义SQLCA： 用EXEC SQL INCLUDE SQLCA定义
- 使用SQLCA：SQLCA中有一个存放每次执行SQL语句后返回代码的变量SQLCODE，是否SUCCESS。

嵌入式SQL—实例

```
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SX*/
                  SELECT Sno,Sname,Ssex,Sage       /*SX对应的语句*/
                  FROM Student
                  WHERE SDept = :deptname;
        EXEC SQL OPEN SX;          /*打开游标SX，指向查询结果的第一行*/
```

嵌入式SQL—实例

- 主变量
  - 嵌入式SQL语句中可以使用主语言的程序变量来输入或输出数据
  - 简称为主变量（Host Variable）
  - 在SQL语句中使用主变量时，为了与数据库对象名（表名、视图名、列名等）区别，SQL语句中的主变量名前要加冒号（:）作为标志
- 建立数据库连接
  - EXEC SQL CONNECT TO target[AS connection-name][USER user-name];
    - target是要连接的数据库服务器
    - 常见的服务器标识串，如<dbname>@<hostname>:<port>
- 关闭数据库连接 ： EXEC SQL DISCONNECT [connection];

嵌入式SQL—实例

- 游标
  - 游标是系统是一个数据缓冲区，存放SQL语句的执行结果
  - 每个游标区都有一个名字，用SQL语句逐一从游标中获取记录，并赋给主变量，交由主语言进一步处理
  - SQL语言是面向集合的，一条SQL语句可以产生或处理多条记录
  - 主语言是面向记录的，一组主变量一次只能存放一条记录
  - 引入了游标的概念，来满足SQL语句与应用程序输入/输出数据

```
EXEC SQL DECLARE SX CURSOR FOR      /*定义游标SX*/
        SELECT Sno,Sname,Ssex,Sage   FROM Student
                WHERE SDept = :deptname;
EXEC SQL OPEN SX;       /*打开游标SX，指向查询结果的第一行*/
```

嵌入式SQL—实例

```
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');
```

嵌入式SQL—实例

```
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更新语句*/
                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*/
}
```

嵌入式SQL—不使用游标的SQL语句

1. 查询结果为单记录的SELECT语句
2. 非CURRENT形式的增删改语句

- [例8.2] 根据学生号码查询学生信息。

```
EXEC SQL SELECT Sno,Sname,Ssex,Sage,Sdept
         INTO:Hsno,:Hname,:Hsex,:Hage,:Hdept
        FROM  Student
         WHERE Sno=:givensno;  /*使用主变量givensno的值进行查询*/
```

- [例8.4] 修改某个学生选修1号课程的成绩。

```
EXEC SQL UPDATE SC     SET Grade=:newgrade
        WHERE Sno=:givensno;      /*学号赋给主变量：givensno*/
```

嵌入式SQL—不使用游标的SQL语句

- [例8.5] 某个学生新选修了某门课程，将有关记录插入SC表中。假设插入的学号已赋给主变量stdno，课程号已赋给主变量couno。

```
gradeid=-1;                          /*gradeid为指示变量，赋为负值*/
EXEC SQL INSERT    INTO SC(Sno,Cno,Grade)
   VALUES(:stdno,:couno,:gr :gradeid);    /*:stdno,:couno,:gr为主变量*/
```

  - 由于该学生刚选修课程，成绩应为空，所以要把指示变量赋为负值

- 指示变量（Indicator Variable）
  - 是一个整型变量，用来“指示”所指主变量的值或条件
  - 一个主变量可以附带一个指示变量指示变量的用途
    - 指示输入主变量是否为空值
    - 检测输出变量是否为空值，值是否被截断

嵌入式SQL—使用游标的SQL语句

1. 查询结果为多条记录的SELECT语句
2. CURRENT形式的UPDATE和DELETE语句

- 使用游标的步骤

（1）说明游标 EXEC SQL DECLARE <游标名> CURSOR
FOR <SELECT语句>;

（2）打开游标 EXEC SQL OPEN <游标名>;

（3）推进游标指针并取当前记录

EXEC SQL FETCH <游标名> INTO <主变量>[<指示变量>]
[,<主变量>[<指示变量>]]...;

（4）关闭游标 EXEC SQL CLOSE <游标名>;

嵌入式SQL—使用游标的SQL语句

1. 查询结果为多条记录的SELECT语句
2. CURRENT形式的UPDATE和DELETE语句

```
EXEC SQL UPDATE Student          /*嵌入式SQL更新语句*/
     SET Sage = :NEWAGE     WHERE CURRENT OF SX;
                    /*对当前游标指向的学生年龄进行更新*/
```

- 不能使用CURRENT形式的UPDATE语句和DELETE语句
  - 当游标定义中的SELECT语句带有UNION或ORDER BY子句
  - 该SELECT语句相当于定义了一个不可更新的视图

嵌入式SQL—动态SQL

- 动态嵌入式SQL
  - 允许在程序运行过程中临时“组装”SQL语句
  - 支持动态组装SQL语句和动态参数两种形式
- [例8.6] 创建基本表TEST。

```
EXEC SQL BEGIN DECLARE SECTION;
        const char *stmt="CREATE TABLE test(a int);";
        /*SQL语句主变量，内容是创建表的SQL语句*/
EXEC SQL END DECLARE SECTION;
...
EXEC SQL EXECUTE IMMEDIATE :stmt;
        /*执行动态SQL语句*/
```

嵌入式SQL—动态SQL

- [例8.7] 向TEST中插入元组。

```
EXEC SQL BEGIN DECLARE SECTION;
        const char *stmt = "INSERT INTO test VALUES(?);";
                                    /*声明SQL主变量内容是INSERT语句 */
EXEC SQL END DECLARE SECTION;
...
EXEC SQL PREPARE mystmt FROM :stmt; /*准备语句*/
...
EXEC SQL EXECUTE mystmt USING 100;   /*设定INSERT语句插入值100 */
EXEC SQL EXECUTE mystmt USING 200;   /* 设定INSERT语句插入值200 */
```

  - 动态参数
    - 使用参数符号（?）表示该位置的数据在运行时设定
    - 通过 PREPARE语句准备主变量和EXECUTE绑定数据或主变量来完成

嵌入式SQL—过程化的SQL块

- 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
- 过程化SQL功能

1. 条件控制：IF-THEN，IF-THEN-ELSE和嵌套的IF语句
2. 循环控制：LOOP，WHILE-LOOP和FOR-LOOP
3. 错误处理：BEGIN
   SQL语句、过程化SQL的流程控制语句
   EXCEPTION
   异常处理部分
   END;

嵌入式SQL—存储过程

- 存储过程：由过程化SQL语句编写，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。
- 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

- 存储过程的用户接口

  1）创建存储过程

  2）执行存储过程

  3）修改存储过程

  4）删除存储过程

嵌入式SQL—存储过程

- [例8.8] 利用存储过程来实现下面的应用：
  从账户1转指定数额的款项到账户2中。

```
CREATE OR REPLACE PROCEDURE TRANSFER(inAccount INT,outAccount
INT,amount FLOAT)
  /*定义存储过程TRANSFER，其参数为转入账户、转出账户、转账额度*/
AS DECLARE                    /*定义变量*/
      totalDepositOut Float;
      totalDepositIn Float;
      inAccountnum INT;
```

嵌入式SQL—存储过程

```
BEGIN                                   /*检查转出账户的余额 */
   SELECT Total INTO totalDepositOut FROM Accout
         WHERE accountnum=outAccount;
   IF totalDepositOut IS NULL THEN   /*如果转出账户不存在或账户中没有存款*/
         ROLLBACK; RETURN;          /*回滚事务*/
   END IF;
   IF totalDepositOut< amount THEN          /*如果账户存款不足*/
         ROLLBACK; RETURN;         /*回滚事务*/
   END IF;
   SELECT Accountnum INTO inAccountnum FROM Account
        WHERE accountnum=inAccount;
   IF inAccount IS NULL THEN        /*如果转入账户不存在*/
         ROLLBACK; RETURN;          /*回滚事务*/
   ENDIF;
```

嵌入式SQL—存储过程

```
    UPDATE Account SET total=total-amount
        WHERE accountnum=outAccount;      /* 修改转出账户余额，减去转出额 */
    UPDATE Account SET total=total + amount
        WHERE   accountnum=inAccount;     /* 修改转入账户余额，增加转入额 */
    COMMIT;                               /* 提交转账事务 */
END;
```

- [例8.9] 从账户01003815868转10000元到01003813828账户中

  CALL PROCEDURE TRANSFER(01003813828,01003815868,10000);

- 修改存储过程

  ALTER PROCEDURE 过程名1 RENAME TO 过程名2;

- 删除存储过程

  DROP PROCEDURE 过程名();

嵌入式SQL—函数

1. 函数的定义语句格式

CREATE OR REPLACE FUNCTION 函数名 ([参数1,参数2,…])
RETURNS <类型> AS <过程化SQL块>;

2. 函数的执行语句格式

CALL/SELECT 函数名 ([参数1,参数2,…]);

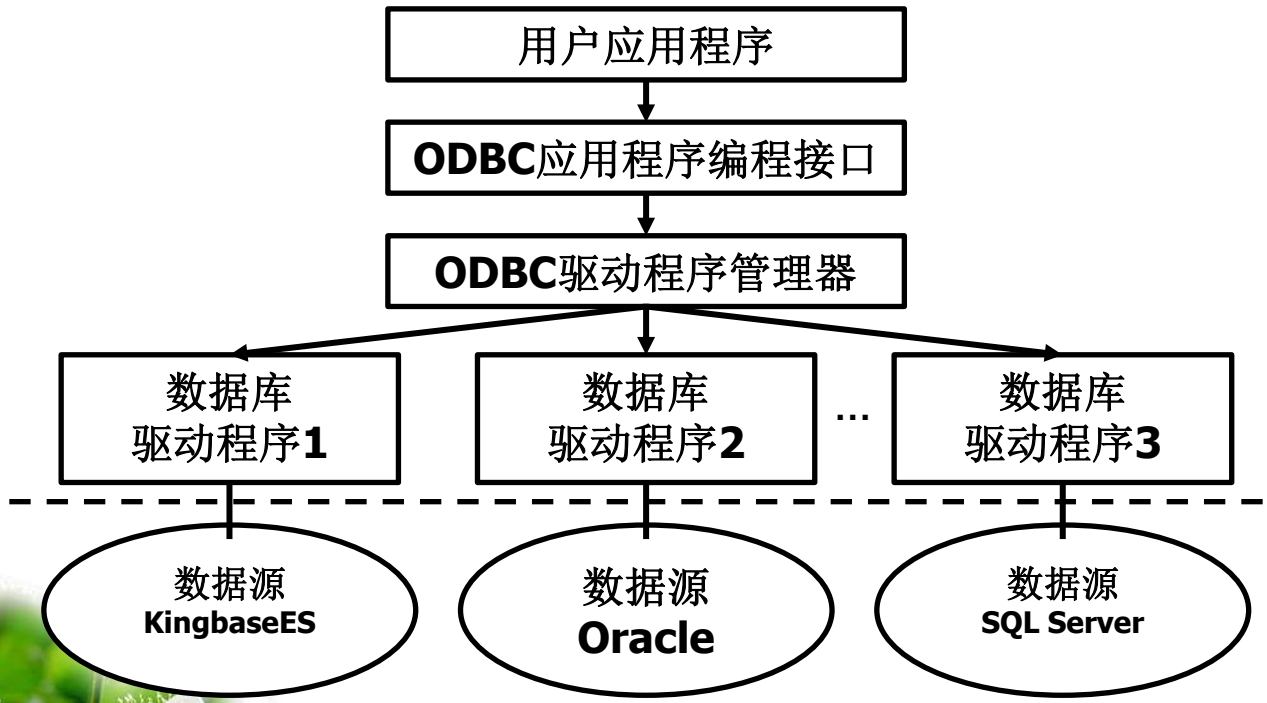
3. 修改函数

- 重命名 ALTER FUNCTION 过程名1 RENAME TO 过程名2;
- 重新编译 ALTER FUNCTION 过程名 COMPILE;

- 函数和存储过程的异同
  - 同：都是持久性存储模块
  - 异：函数必须指定返回的类型

ODBC编程

- 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
- 规范关系数据库管理系统应用接口



ODBC编程

- 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
- ODBC应用程序不能直接存取数据库
  - 其各种操作请求由驱动提交给关系数据库管理系统的ODBC驱动程序所支持的函数来存取数据库

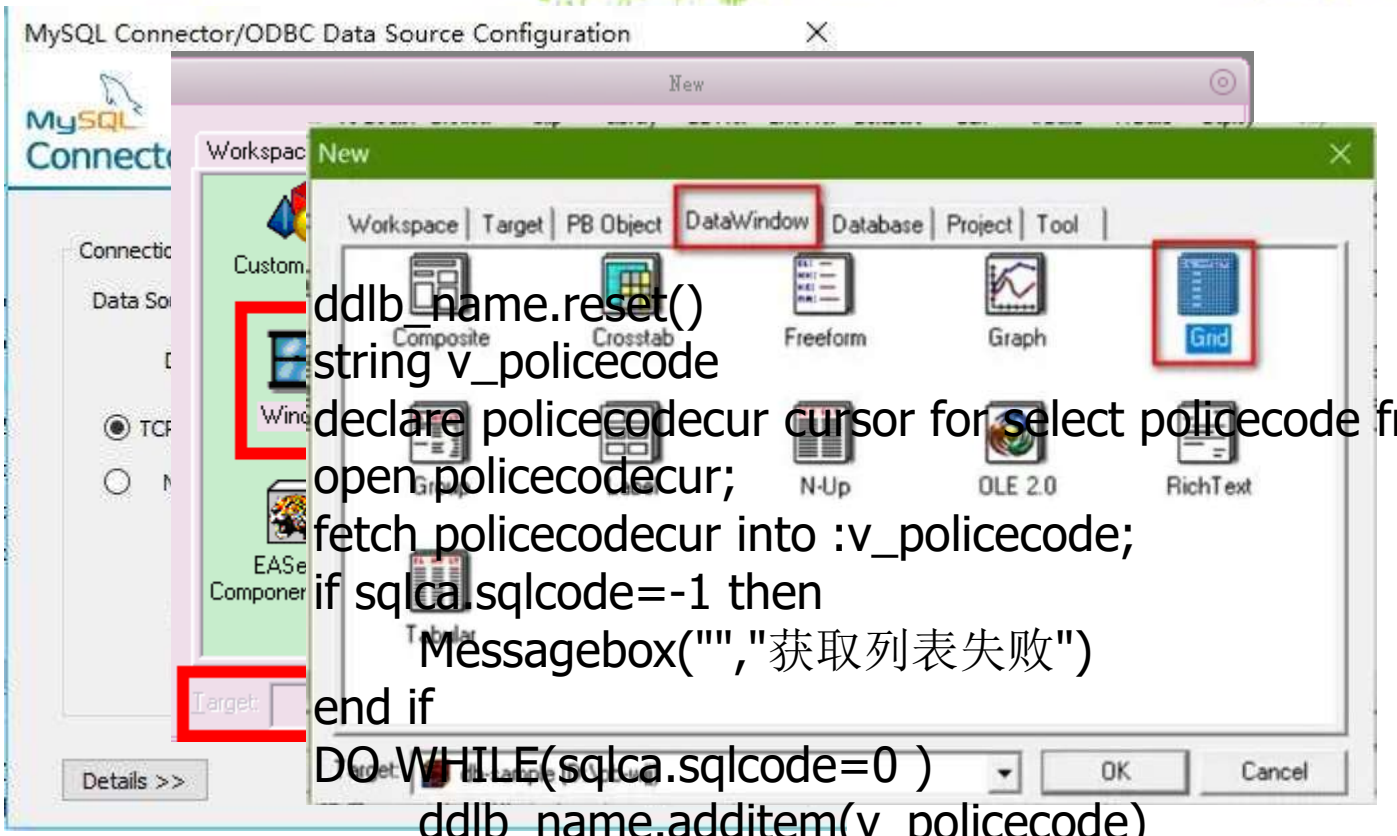
ODBC编程

- 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**(32位整数，指针)**
  - 连接函数（SQLDriverconnect等）
  - 与信息相关的函数（SQLGetinfo、SQLGetFuction等）
  - 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
- ODBC数据源
  - 指定唯一的数据源名（Data Source Name，简称DSN），并映射到用户名、服务器名、所连接的数据库名等

ODBC编程—工作流程

- ret=SQLConnect(kinghdbc,"KingbaseES ODBC", SQL_NTS,"SYSTEM", SQL_NTS, "MANAGER",SQL_NTS);
- ret=SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&kinghenv);
- ret=SQLConnect(serverhdbc, "SQLServer", SQL_NTS, "sa",SQL_NTS, "sa",SQL_NTS);
- ret=SQLAllocHandle(SQL_HANDLE_STMT,kinghdbc, &kinghstmt);
- ret=SQLExecDirect(kinghstmt,"SELECT * FROM STUDENT",SQL_NTS);
  ret=SQLBindCol(kinghstmt,4,SQL_C_LONG,&sAge,0,&cbAge);
- while ((ret=SQLFetch(kinghstmt))!=SQL_NO_DATA_FOUND) {
  if(ret==SQL_ERROR) printf("Fetch error\n");
  else ret=SQLExecute(serverhstmt);
  }
- SQLFreeHandle(SQL_HANDLE_STMT,kinghstmt);
  SQLDisconnect(serverhdbc);

1. 配置数据源
2. 初始化环境
3. 建立连接
4. 分配语句句柄
5. 执行SQL语句
6. 结果集处理
7. 中止处理




```
ddlb_name.reset()
string v_policecode
declare policecodecur cursor for select policecode from biz_ code;
open policecodecur;
fetch policecodecur into :v_policecode;
if sqlca.sqlcode=-1 then
      Messagebox("","获取列表失败")
end if
DO WHILE(sqlca.sqlcode=0 )
      ddlb_name.additem(v_policecode)
      fetch policecodecur into :v_policecode;
LOOP
close policecodecur;
```

小结

- 嵌入式SQL把SQL语句嵌入到高级语言中
  - ✓ SQL与主语言具有不同的数据处理方式
- 嵌入式SQL
- 过程化SQL
- 存储过程和函数
- ODBC编程