

第六章

数据库中的数据交换

6.1 概述

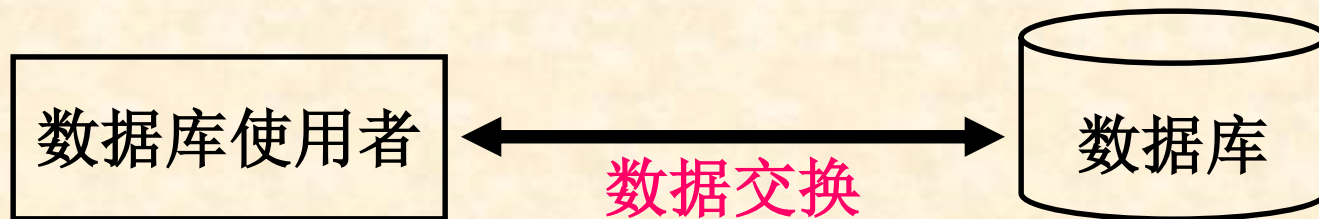
6.2 数据交换的管理

6.3 数据交换的流程

6.4 数据交换的四种方式

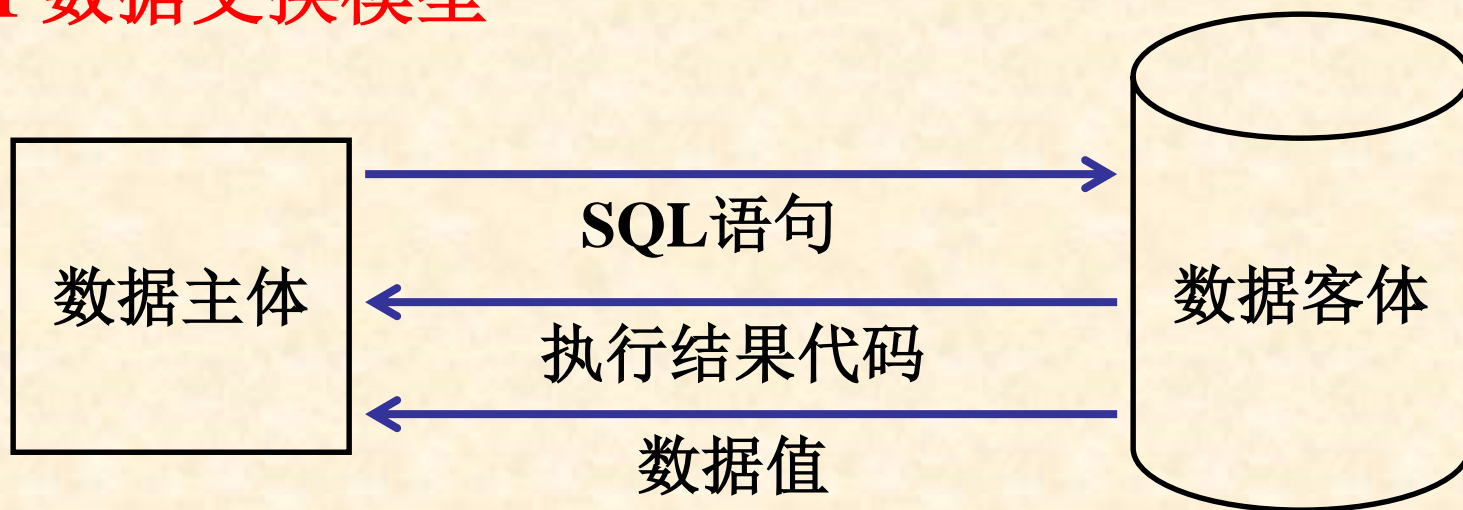
6.1 概述

□ 何为数据库中的‘数据交换’？



是数据库与其使用者间的数据交互过程

6.1.1 数据交换模型



6.1.2 数据交换的五种方式

□ 初级阶段

- 人机对话方式，交互方式

□ 中级阶段

- 嵌入式方式
- 自含方式
- 调用层接口（call level interface）方式

□ 近期阶段

- Web方式

6.1 概述

6.2 数据交换的管理

6.3 数据交换的流程

6.4 数据交换的四种方式

6.2 数据交换的管理

□ 数据交换的管理包括如下内容：

6.2.1 会话管理

6.2.2 连接管理

6.2.3 游标管理

6.2.4 诊断管理

6.2.5 动态SQL

6.2.1 会话管理

- 数据交换是两个数据体之间的会话过程，会话的进行须预先作环境的设定，这就是会话管理

- 会话管理的内容包括：
 - 会话的数据客体模式设定（网络环境、目录层、模式层）
 - 会话的语言模式设定（字符集）
 - 会话的时间模式设定（包括时区）
 - 会话的标识符设定（对所建立的‘会话’进行命名）

6.2.2 连接管理

- ❑ 连接管理负责在数据主、客体间建立实质性的关联，包括服务器指定、内存区域分配等。也可以断开两者之间的关联

- ❑ 连接语句

CONNECT TO <连接目标>

<连接目标> ::= <服务器名> [AS<连接名>][USER<用户名>]

- ❑ 断开连接语句

DISCONNECT <断开对象>

<断开对象> ::= <连接名> | ALL | CURRENT

6.2.3 游标管理

- ❑ 在数据交换中，数据库SQL中的变量是集合型的而应用程序的程序设计语言中的变量则是标量型，因此数据库中SQL变量不能直接供程序设计语言使用，而需要有一种机制将SQL变量中的集合量逐个取出后送入应用程序变量内供其使用，而提供此种机制方法是增加游标（**cursor**）语句

6.2.3 游标管理

❑ 游标（Cursor）操作

➤ declare a cursor

- 为某一映像语句（可能返回多个结果元组）的结果集合定义一个命名的游标

➤ open the cursor

- 执行相应的映像语句并打开获得的结果集，此时游标处于活动状态并指向结果集合的第一条记录的前面

➤ fetch a row by the cursor

- 将游标推向结果集合中的下一条记录，读出游标所指向记录的值并赋给对应的主语言变量
- One-Row-at-a-Time Principle

➤ close the cursor

- 关闭所使用的游标，释放相关的系统资源

6.2.3 游标管理

❑ The Declare Cursor Statement

```
EXEC SQL DECLARE cursor-name CURSOR FOR  
    subquery  
    [ ORDER BY ..... ]  
    [ FOR { READ ONLY |  
            UPDATE [ OF columnname, ..... ] } ] ;
```

- 如果查询语句的执行结果是一个元组的集合，那么需要使用游标来获取结果集合中的每一个元组
- 仅当用户确信只可能返回单个结果元组的情况下才可以使用SELECT.....INTO.....形式的嵌入式SQL查询语句

6.2.3 游标管理 – 定义游标

❑ declare a cursor

define the cursor name

```
EXEC SQL DECLARE agent_dollars CURSOR FOR  
  select aid, sum(dollars)  
  from  orders  
  where cid = :cust_id  
  group by aid ;
```

means multiple rows in result set

search by customer's id (stored in host variable cust_id) when open the cursor agent_dollars

6.2.3 游标管理 – 打开游标

□ open the cursor

before open the cursor, you must place cno value of customer's id in the host variable cust_id using in the declare statement of cursor agent_dollars.

```
.....  
EXEC SQL OPEN agent_dollars ;  
.....
```

execute the select statement

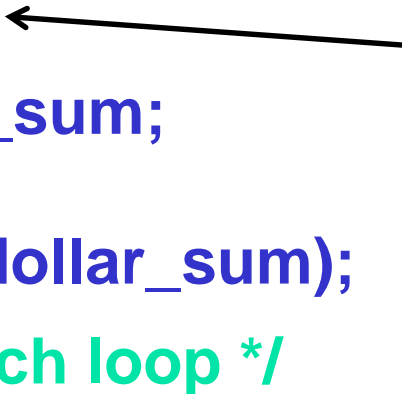
after open the cursor, the pointer of the cursor has been placed in the position before the first row in result set.

6.2.3 游标管理 – 取游标

❑ fetch the result rows

```
while (TRUE) {                               /* loop to fetch rows */
    exec sql fetch agent_dollars
        into :agent_id, :dollar_sum;

    printf("%s %11.2f\n", agent_id, dollar_sum);
}                                              /* end fetch loop */
```



- 1) move the pointer of cursor to the next row, then the next row is current row
- 2) fetch the current row's value into host variables: agent's id to agent_id, summation of dollars to dollar_sum

6.2.3 游标管理 – 取游标

❑ end fetch loop

```
exec sql whenever not found goto finish; ←
```

```
.....  
while (TRUE) {  
    exec sql fetch ..... into .....;  
    .....  
}
```

```
.....  
finish:  exec sql close agent_dollars;
```

declare 'not found' event processing

execute this statement after fetch loop when
'not found' event is occur

6.2.3 游标管理 – 关闭游标

❑ close the cursor

.....

```
EXEC SQL CLOSE agent_dollars ;
```

.....

- 1) close the cursor, and release the result set and other resource in DBMS
- 2) after close the cursor, it can be opened again

6.2.3 游标管理 - 可滚动游标

❑ Scrollable Cursors

```
EXEC SQL DECLARE cursor_name  
    [ INSENSITIVE ] [ SCROLL ]  
    CURSOR [ WITH HOLD ] FOR  
        subquery { UNION subquery }  
        [ ORDER BY ..... ]  
        [ FOR READ ONLY |  
          FOR UPDATE OF columnname ..... ];
```

```
EXEC SQL FETCH  
    [ { NEXT | PRIOR | FIRST | LAST |  
      { ABSOLUTE | RELATIVE } value_spec } FROM ]  
    cursor_name INTO .....;
```

6.2.4 诊断管理

- 在进行数据交换时，数据主体发出数据交换请求后，数据客体返回两种信息：
 - 所请求的数据值
 - 执行的状态值，而这种状态值又被称为诊断值，而生成、获取诊断值的管理称诊断管理

1. 诊断区域

2. 诊断操作

6.2.5 动态SQL

1. 什么是动态SQL?

- 在嵌入式SQL编程中，很多时候编程人员无法确定到底应该做什么工作，所使用的SQL语句也不能预先确定，需要根据程序的实际运行情况来决定，也就是根据实际情况来生成并调用SQL语句。这样的SQL语句被称为**动态SQL**

6.2.5 动态SQL

□ 动态SQL语句的可变性

- SQL语句正文动态可变
- 变量个数动态可变
- 类型动态可变
- SQL语句引用对象动态可变

□ 相对地，事先能够确定下来的嵌入式SQL语句又称为静态SQL

- 静态SQL与动态SQL的优缺点比较（next）

6.2.5 动态SQL

□ 静态SQL

- 在通过预编译时，SQL命令就被分析并为它们的执行作好了相应的准备工作。在程序运行时，只需要调用预先优化好的访问路径
 - 优点：性能好
 - 缺点：只能根据缺省参数值进行优化，其访问路径并非是最优访问路径

□ 动态SQL

- 程序在运行时动态生成的SQL命令
 - 优点：可以根据运行时的数据库最新情况选择最优访问路径
 - 缺点：动态地进行SQL语句的语法分析和访问路径选择

6.2.5 动态SQL

2. 在什么情况下需要使用动态SQL?

- 应用程序需要在执行过程中生成SQL语句;
- SQL语句用到的对象在预编译时不存在;
- 希望SQL语句的执行能够根据执行时的数据库系统内部的统计信息来采用最优的访问策略

3. 嵌入式动态SQL的语句

- 有关描述符区的操作语句
- 有关动态SQL的使用语句

❑ 描述符区（descriptor area）

- 应用程序与数据库需进行信息交互的区域

6.2.5 动态SQL

3. 有关动态SQL使用语句

- **Prepare语句**：为执行对数据库的访问操作而准备一个存储在主变量中的SQL语句

Prepare <语句名> [into <descriptor>]
FROM <主变量>;

【例】 prepare s1 from :mystatement;

【例】 prepare s2 into :mysqlda
from :myquery;

6.2.5 动态SQL

□ **Describe语句**：获得一个已准备好的SQL语句的结果集的描述信息

Describe <语句名> **into** <主变量>;

【例】 describe s1 into :mysqllda;

6.2.5 动态SQL

- ❑ **Execute语句**：执行一个已准备过的SQL语句（非select语句）

Execute <语句名> [**using** <主变量列表>]

【例】 **Execute s1**

【例】 **Execute s1 using :x, :y**

➤ 必需使用游标来处理动态SQL查询命令

- ❑ **Execute immediate**：立即执行一条SQL命令
 - 等价于 **prepare + execute**

6.2.5 动态SQL

4. 动态SQL的分类

□ **直接执行**：不带参数的非查询类动态SQL的执行。例：

.....

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
Char stmt[1024];
```

```
EXEC SQL END DECLARE SECTION;
```

```
do {
```

```
Printf(“请输入非查询类SQL语句: ”);
```

```
Scanf(“%s”, &stmt);
```

```
If (strcmp(stmt, “quit”) == 0) break;
```

```
EXEC SQL EXECUTE IMMEDIATE :stmt ;
```

```
} while (1);
```

.....

6.2.5 动态SQL

- **带动态参数**：在非查询类SQL语句中使用到一些未确定的变量（带参数）

【例】根据学生的姓名来删除学生

.....

```
Strcpy(stmt, "delete from S where Sn = ?");
```

```
/* ? 代表命令参数 */
```

```
EXEC SQL PREPARE s1 FROM :stmt ;
```

.....

```
/* 输入一个学生的姓名到主变量myname中 */
```

```
EXEC SQL EXECUTE s1 USING :myname ;
```

```
/* 用主变量 myname 的值代替原来的 ? */
```

.....

6.2.5 动态SQL

□ **动态查询**：需要查询结果的动态查询类SQL语句，且查询的结果属性不确定，一般情况下需要结合游标来使用

- 用prepare语句准备一条动态查询类SQL语句
- 利用准备好的语句定义游标

EXEC SQL DECLARE <游标名> CURSOR FOR <语句名> ;

- 打开游标

EXEC SQL OPEN <游标名> [USING <主变量列表>] ;

- 推进游标

**EXEC SQL FETCH
INTO <主变量列表> | USING DESCRIPTOR <主变量>;**

- 关闭游标

6.1 概述

6.2 数据交换的管理

6.3 数据交换的流程

6.4 数据交换的四种方式

6.3 数据交换的流程

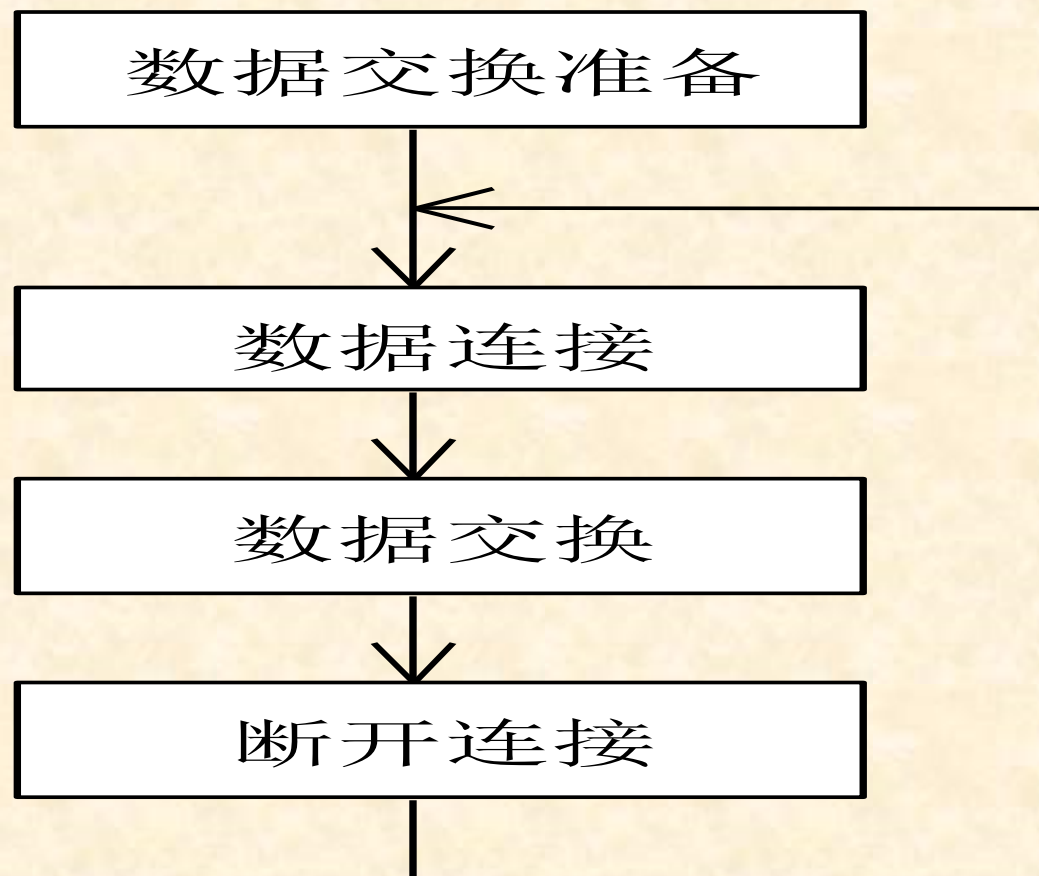


图 6.3 数据交换过程的流程图

6.1 概述

6.2 数据交换的管理

6.3 数据交换的流程

6.4 数据交换的四种方式



6.4 数据交换的四种方式

6.4.1 嵌入式SQL

6.4.2 自含式SQL

6.4.3 调用层接口

6.4.4 Web方式

6.4.1 嵌入式SQL

□ SQL语言的使用方式

➤ 交互式SQL

- 可独立运行，一般供临时用户操作访问数据库用

➤ 嵌入式SQL

- ‘SQL+主语言’ 的应用开发模式

➤ 自含式SQL

- 兼有SQL数据访问和高级程序设计语言的流程控制、简单数值处理功能
- 可在数据库服务器中独立运行
- 常用于编写存储过程，存储函数，触发器

6.4.1 嵌入式SQL

- 在嵌入式SQL中需要解决下面的四个问题：
 1. 主语言语句与SQL语句的区别
 2. 主语言程序与嵌入式SQL间的通讯
 3. 主语言变量与SQL变量的区别
 4. 主语言语句与SQL语句的数据交换

6.4.1 嵌入式SQL

1. 主语言语句与SQL语句的区别？

- 对嵌入在主语言中的SQL语句加前缀（EXEC SQL）和后缀（END_EXEC或；）

2. 主语言程序与ESQL间的通讯

EXEC SQL INCLUDE SQLCA;

- SQLCA是一个系统定义的全局变量，用于返回嵌入式SQL命令的执行状态及其结果信息。如：命令的执行是否成功？执行结果是否为空？等等

6.4.1 嵌入式SQL

3. 主语言变量与SQL变量的区别？

➤ 主变量

- 在嵌入式SQL语句中使用的主语言变量

➤ SQL变量

- SQL语句中的表名或属性名也可以看成是一个变量，我们称其为SQL变量

➤ 主变量是标量变量，而SQL变量则是集合变量

- 可以在嵌入式SQL语句中使用主语言变量，但必需在主语言变量前面加上一个前缀 ‘:’，以便与SQL语句中的表名或属性名区别开来

6.4.1 嵌入式SQL

- 可以通过主语言变量在嵌入式SQL语句与主语言语句之间交换数据。但一个主语言变量一次只能存储一个值
 - 可以通过主语言变量获取查询结果值，并用于主语言语句中
 - 也可以将保存在主语言变量中的值用于SQL语句的执行

6.4.1 嵌入式SQL

- ❑ 在嵌入式SQL语句中使用的主语言变量必需预先在DECLARE语句段中定义

EXEC SQL BEGIN DECLARE SECTION;

...

... /* 定义在嵌入式SQL语句中使用的主语言变量 */

...

EXEC SQL END DECLARE SECTION;

6.4.1 嵌入式SQL

4. 主语言语句与SQL语句的数据交换？

- SQL语句的处理对象与处理结果都是集合量，而主语言的语句只能处理标量值。因此在这两者之间需要有特殊的数据交换手段
 - **游标**（**cursor**）
- 通过游标机制可以将SQL变量中的集合量逐个取出送入主变量内，再供主程序使用。从而完成主语言程序与SQL语句之间的数据交换

嵌入式SQL语句的例子（1）

```
EXEC SQL      select count(*)  
                into :host_var  
                from customers ;
```

□ 与交互式SQL的区别

- 带有前缀 ‘EXEC SQL’ 和后缀 ‘;’
- 使用into子句来获取结果元组值
 - 该查询的结果集中只含有单个结果元组
- 用主语言变量 ‘:host_var’ 保存结果元组中的属性值
 - 通过前缀 ‘:’ 来区分主语言变量和SQL中的表名或属性名

嵌入式SQL语句的例子（2）

```
EXEC SQL select cname, discnt  
          into :cust_name, :cust_discnt  
          from customers  
          where cid = :cust_id ;
```

- ❑ 为了使用这些主语言变量，必须首先在 **DECLARE SECTION** 部分声明这些变量，**Why?**

❑ 主语言变量声明语句（**DECLARE SECTION**）作用

- 在编译时就可以检查主语言变量及其所对应的属性的数据类型是否一致
- 为接收从数据库返回的结果值而预先申请内存空间

6.4.1.2 嵌入式SQL程序的编制

程序结构

- ❑ **The Declare Section**
 - 定义主语言变量
- ❑ **Condition Handling**
 - 在嵌入式SQL语句执行出错或发生异常的情况下，对应用程序的执行流程进行控制
- ❑ **SQL Connect to Database**
 - 连接数据库
- ❑ **Main Body of Application Program**
 - 用主语言编写的应用程序，包括界面和数据处理过程
 - 用ESQL编写的数据库访问语句
- ❑ **SQL Disconnect**
 - 撤消与数据库的连接

The Declare Section

```
exec sql begin declare section;  
    char cust_id[5];  
    char cust_name[14];  
    float cust_discnt;  
    char user_name[20], user_pwd[20];  
exec sql end declare section;
```

**Begin declare
SQL host
variables**

**host variables
for cno, four
characters and
a null terminator**

**End of declare
section**

**host variables for
user name and
password**

Condition Handling



SQL Connect Statement

□ SQL99

```
EXEC SQL CONNECT TO target-server  
[AS connect-name] [USER username] ;
```

```
EXEC SQL CONNECT TO DEFAULT ;
```

➤ target-server

- 数据库名

➤ connect-name

- 本次连接（connect session）的名称
- 在一个程序中可以同时维持多个连接

➤ username

- 本次连接所使用的数据库用户的用户名

SQL Connect Statement

❑ Oracle

```
EXEC SQL CONNECT TO :user_name  
IDENTIFIED BY :user_pwd ;
```

➤ user_name

- Oracle数据库用户的用户名

➤ user_pwd

- 数据库用户的口令

➤ 在Oracle的数据库连接命令中，不需要给出需要连接的数据库名

交互式访问数据库的程序段

```
while (prompt(cid_prompt, 1, cust_id, 4) >= 0)
```

```
{
```

```
    exec sql select cname, discnt  
           into :cust_name, :cust_discnt  
    from customers  
    where cid = :cust_id;
```

```
    exec sql commit work;
```

```
    printf("CUSTOMER'S NAME IS %s AND DISC  
           cust_name, cust_discnt);
```

```
    continue;
```

```
notfound:
```

```
    printf("Can't find customer %s, continuing\n", cust_id);
```

```
}
```

获取用户输入的客户编号，并存储在主变量 **cust_id** 中

通过给定的主变量接收可能的单个结果元组中的属性值

SQL Disconnect Statement

❑ SQL99

EXEC SQL DISCONNECT connect-name ;

or

EXEC SQL DISCONNECT CURRENT ;

- 在撤消与数据库的连接之前，用户需要通过事务提交命令（Commit）来确认本次运行所执行的操作，也可以通过事务回滚（Rollback）命令来放弃本次运行已执行的操作

EXEC SQL COMMIT WORK ;

EXEC SQL ROLLBACK WORK ;

SQL Disconnect Statement

❑ Oracle

exec sql commit release ;

exec sql rollback release ;

a rollback statement followed by a disconnect statement, to undo any partial work in an unsuccessful task

a commit statement followed by a disconnect statement, for successful completion

6.4.1.3 嵌入式SQL的编译

❑ 嵌入式SQL应用程序的编译流程

➤ 预编译 (Precompiler)

- 使用DBMS所提供的预编译程序，将应用程序中的嵌入式SQL语句转换成相应的主语言调用函数

➤ 编译 (Compiler)

- 使用主语言的编译程序编译转换后的源程序

➤ 链接

- 生成可执行程序

➤ 执行

6.4.2 自含式SQL

❑ 服务器内的数据交换

6.4.2.1 自含式SQL的内容

- 传统的SQL
- 传统程序设计语言的主要成份，包括流程控制及循环语句、输出语句、调用语句以及服务性的函数库、类库等
- SQL中的数据交换，包括游标、诊断及动态SQL

6.4.2.2 自含式SQL的编程

- 自含式SQL编程的结构单位是 ‘块’，一个块一般包括三部分内容

DECLARE

..... /* 声明部分 */

BEGIN

.....

EXCEPTION

...../* 例外部分 */

END

/* 主体部分 */

6.4.2.2 自含式SQL的编程

- 自含式SQL相对于嵌入式SQL的优势
 - 不需要区分主变量与SQL变量
 - 编译过程极为简单

6.4.3 调用层接口

□ C/S方式下的数据库访问接口

- ISO接口：SQL/CLI
- 微软接口：ODBC
- SUN接口：JDBC

□ 数据交换的流程

- 连接阶段
- 数据交换阶段
- 断开连接阶段

6.4.3 调用层接口

□ 连接阶段

➤ 负责资源分配和建立连接

➤ 相关函数

- | | |
|----------------------|---------|
| ▪ AlloEnv | 分配SQL环境 |
| ▪ AlloStmt | 分配SQL语句 |
| ▪ AlloHandle | 分配SQL资源 |
| ▪ AlloConnect | 分配SQL连接 |
| ▪ Connect | 创建连接 |

6.4.3 调用层接口

□ 数据交换阶段

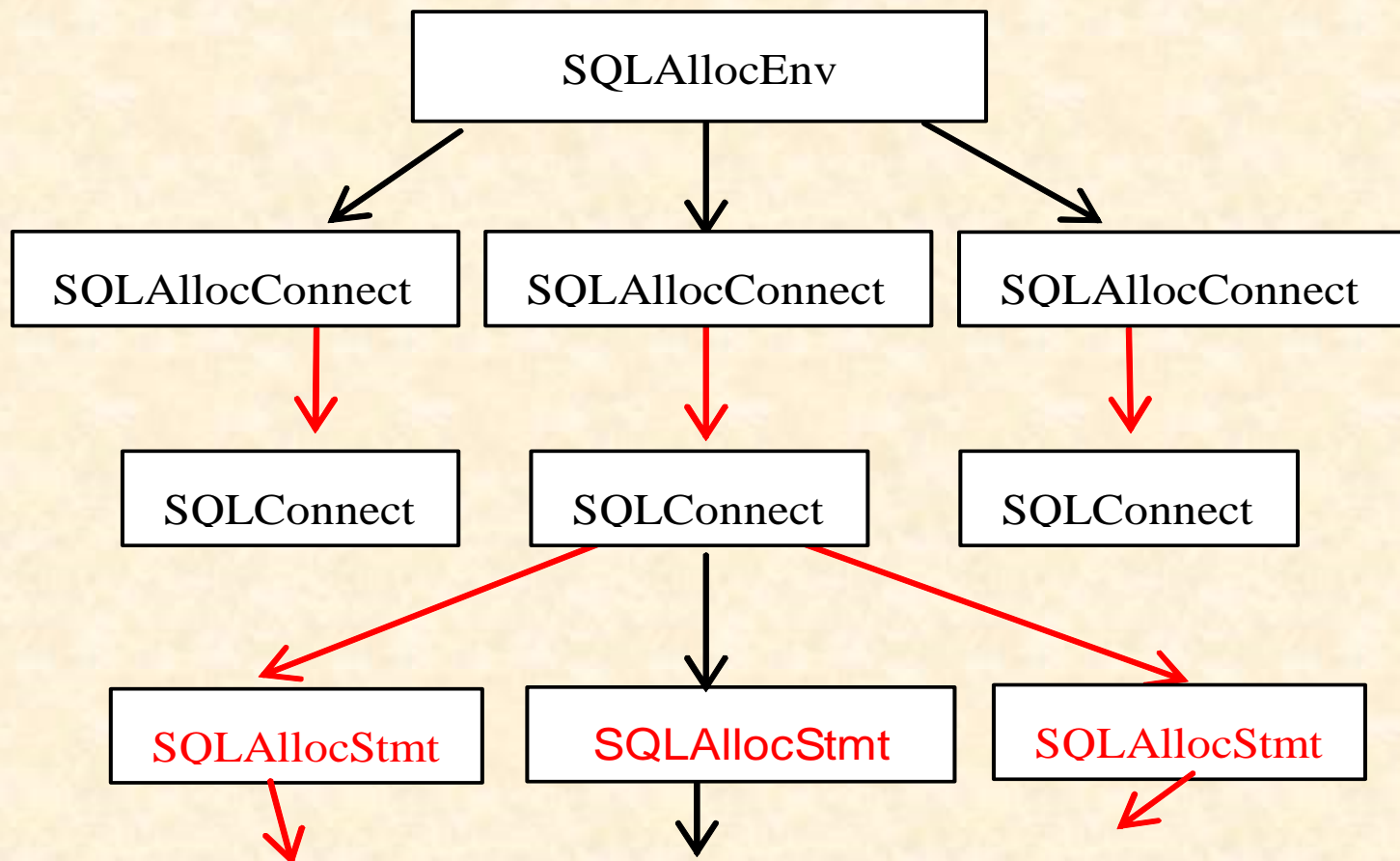
➤ 主要的函数

- 有关游标的函数
- 有关诊断的函数
- 有关动态SQL的函数
- 有关数据获取的函数
- 其它

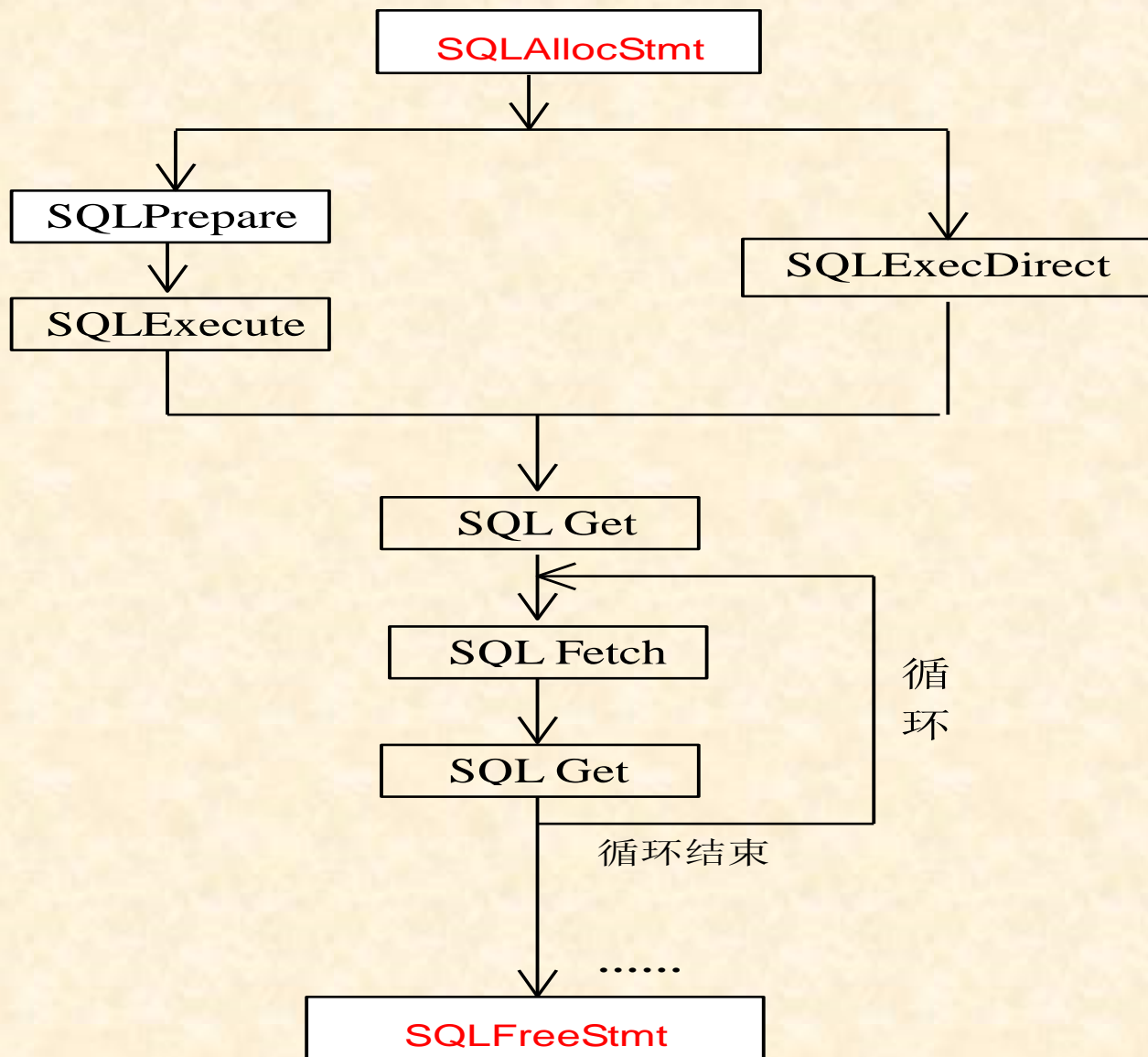
6.4.3 调用层接口

- ❑ 断开连接阶段
 - 断开连接并释放资源

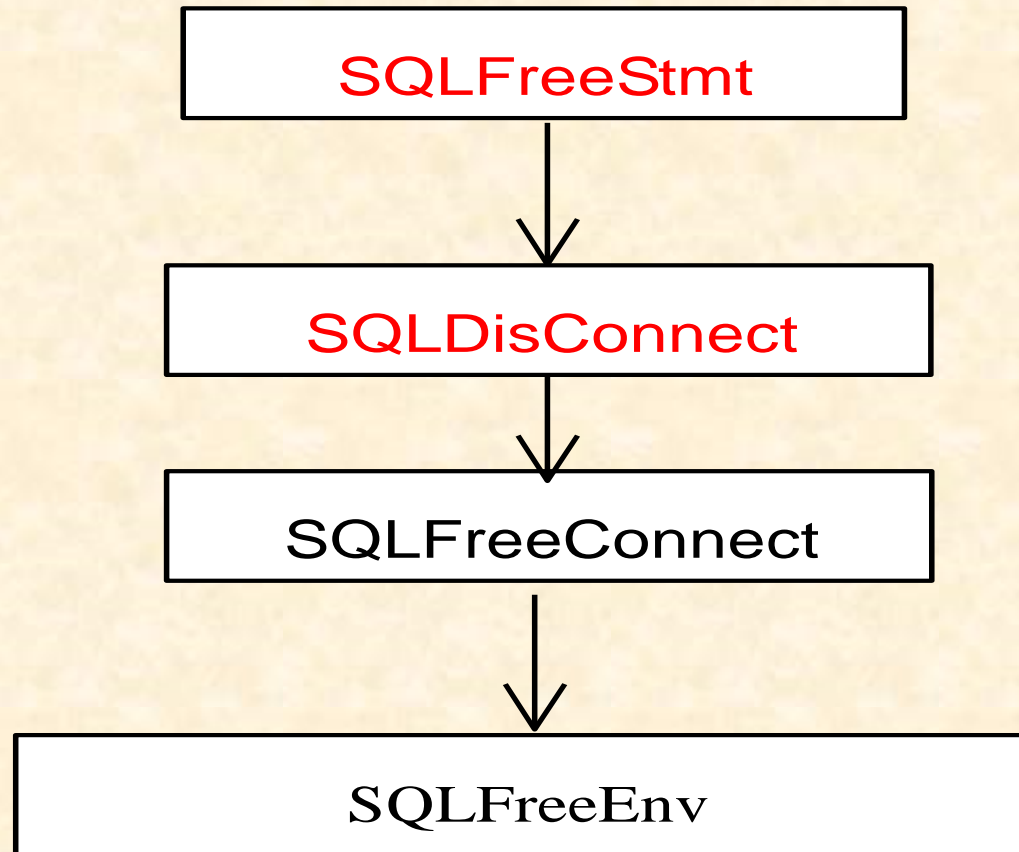
使用ODBC的程序结构1 – 连接阶段



使用ODBC的程序结构2 – 数据交换阶段



使用ODBC的程序结构3 – 断开连接阶段





6.4.4 Web方式

- ❑ **ASP**
- ❑ **JSP**
- ❑ **XML**

数据库连接池

- ❑ 数据库连接是一种关键的有限的昂贵的资源，这一点在多用户的网页应用程序中体现得尤为突出
- ❑ 对数据库连接的管理能显著影响到整个应用程序的伸缩性和健壮性，影响到程序的性能指标
- ❑ 数据库连接池正是针对这个问题提出来的

数据库连接池

- ❑ 数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而再不是重新建立一个
- ❑ 释放空闲时间超过最大空闲时间的数据库连接来避免因为没有释放数据库连接而引起的数据库连接遗漏
- ❑ 这项技术能明显提高对数据库操作的性能

- ❑ **Class.forName(“com.mysql.jdbc.Driver”);
Connection conn = DriverManager.getConnection(“url”, “user”, “password”);
conn.close();**
- ❑ **Statement, PreparedStatement & addBatch()**
- ❑ **ResultSet & ResultSetMetaData**
- ❑ **SQLException**

JDBC连接池

```
BasicDataSource dsObjectCoref = new BasicDataSource();  
dsObjectCoref.setDriverClassName(DBParam.DRV);  
dsObjectCoref.setUrl(DBParam.URL_ObjectCoref);  
dsObjectCoref.setUsername(DBParam.USR_ObjectCoref);  
dsObjectCoref.setPassword(DBParam.PWD_ObjectCoref);  
  
dsObjectCoref.setMaxActive(16);  
dsObjectCoref.setMaxIdle(2);  
dsObjectCoref.setDefaultTransactionIsolation(  
    Connection.TRANSACTION_SERIALIZABLE);  
  
dsObjectCoref.setTestOnBorrow(true);  
String sqlstr = "SHOW TABLES";  
dsObjectCoref.setValidationQuery(sqlstr);
```