

# 《数据库系统原理实验》

## 指导书

### 一、实验目的

《数据库系统原理》是软件工程、计算机专业本科生限选的实践性专业基础课。通过实验深入理解和掌握《数据库系统原理》课程教学中的基本概念、基本理论。本课程的任务是使学生在掌握数据库的基本概念、基本理论的基础上，使用一种数据库系统和一种开发语言，完成一个实用数据库应用系统的设计与开发。本课程的目标是通过实验使得学生具有使用 SQL 语言建立和操作数据库，使用 JDBC 或 ODBC 接口实现与数据库的连接，以及使用开发语言编写数据库应用程序的能力，并最终设计和开发完整的数据库应用系统，从而提高开发实际数据库应用的能力。

### 二、实验内容

本实验由五个实验项目组成，内容为数据库系统的使用（SQL SERVER2000 或 MYSQL 或 ORACLE 等）、SQL 语言的使用、数据库编程、数据库访问接口、实用数据库应用系统的设计与开发等。

实验序号	实验项目内容	学时	选做/必做	实验类型
实验 1	数据库系统的使用（SQL SERVER2000 或 MYSQL 或 ORACLE 等）	2	选做	验证型
实验 2	SQL 语言的使用	2	必做	验证型
实验 3	数据库编程	2	选做	设计型
实验 4	数据库访问接口 ODBC	6	必做	设计型
实验 5	实用数据库应用系统的设计与开发	20	必做	综合型
合计		32		

## 实验 1：数据库系统的使用（SQL SERVER

## 或 MYSQL 或 ORACLE 等)

实验目的：通过该实验了解数据库管理系统，掌握利用 SQL SERVER 2015 或 MYSQL 或 ORACLE 等对数据库、数据库表、数据库用户授权的设置。

实验背景：必须学完 SQL 命令 DDL 语句的语法。

实验设备：一台 PC 机

实验条件：PC 机上必须安装 SQL SERVER 2015 或 MYSQL 8.0 或 ORACLE Database 12c 等数据库。

实验学时：2 学时

实验要求：(1) 利用查询分析器创建数据库，掌握表（关系）和索引的建立方法；  
(2) 利用查询分析器掌握表结构（关系模式）的修改方法；  
(3) 实践 DBMS 提供的数据库完整性功能，加深对数据库完整性的理解。  
(4) 创建一个数据库用户和数据库角色，使之拥有对某个数据库的权力。

实验内容：

1. 在 studentdb 数据库中利用查询分析器创建以下 3 个表，同时完成数据库完整性的定义（实体完整性、参照完整性和用户定义的域完整性）：

student（学生信息表）：

主码	列名	数据类型	宽度	小数位	空否	取值范围	备 注
Pk	sno	char	5		N		学号
	sname	char	10		N		姓名
	ssex	char	2		Y		性别
	sage	smallint			Y	不小于 12	年龄
	sdept	char	15		Y		系名

course（课程表）：

主码	列名	数据类	宽度	小数位	空否	备 注
----	----	-----	----	-----	----	-----

		型				
Pk	cno	Char	2		N	课程号
	cname	Char	20		Y	课程名称
	cpno	Char	2		Y	先行课号
	ccredit	smallint			Y	学分

sc (学生选课表):

主 码	列名	数据类 型	宽 度	小 数	空 否	外 码	参照关 系	取值范 围	备 注
Pk	sno	Char	5		N	Fk	student		学号
	cno	Char	2		N	Fk	course		课程号
	grade	Decimal	5	1	Y			0≤x≤100	成绩

2. 修改表结构，具体要求如下：

- (1) 将表 course 的 cname 列的数据类型改为 varchar(40).
- (2) 为表 student 增加一个新列: birthday(出生日期), 类型为 datetime, 默认为空值.
- (3) 将表 sc 中的 grade 列的取值范围改为小于等于 150 的正数.
- (4) 为 Student 表的“Sex”字段创建一个缺省约束, 缺省值为'男'.
- (5) 为“Sdept”字段创建一个检查约束, 使得所在系必须是'计算机'、'数学'或'信息'之一。
- (6) 为 Student 表的“Sname”字段增加一个唯一性约束
- (7) 为 SC 表建立外键,依赖于 Student 表的 fk\_S\_c 约束。
- (8) 禁止启用 Student 表的“Sdept”的 CHECK 约束 ck\_student。

3. 分别建立以下索引(如果不能成功建立,请分析原因)

- (1) 在 student 表的 sname 列上建立普通降序索引.
- (2) 在 course 表的 cname 列上建立唯一索引.
- (3) 在 sc 表的 sno 列上建立聚集索引.
- (4) 在 spj 表的 sno(升序), pno(升序)和 jno(降序)三列上建立一个普通索引.

4. 创建一个数据库用户和数据库角色，使之拥有对数据库 StudentDB 的一切权力。
5. 完成如下的数据授权操作（要求学生分组共同完成本实验）：
  - (1) 数据对象操作授权：将对表和视图的查询及其 INSERT、UPDATE、DELETE 操作权限分不同情况授与其他用户，同时取得其他用户的授权，体会安全控制的作用。
  - (2) 在授权过程中，体会 GRANT 语句中 WITH GRANT OPTION 短语的作用。
  - (3) 分情况收回授权，并体会 REVOKE 语句中 GRANT OPTION FOR 和 CASCADE 短语的作用。

## 实验 2：SQL 语句的使用

**实验目的：**通过该实验熟悉和掌握 SQL 语句的语法。

**实验背景：**必须学完 SQL 命令 DDL、DML 语句的语法。

**实验设备：**一台 PC 机

**实验条件：**PC 机上必须安装 SQL SERVER 2015 或 MYSQL 8.0 或 ORACLE Database 12c 等数据库，并且已经执行脚本 oracle\_home\rdbms\admin\bdemobld.sql，为当前的 scott 用户增加可用的练习表。

**实验学时：**2 学时

- 实验要求：**
- (1) 要求用交互式方式输入 DDL 命令建立实验指导书中规定的表结构、视图、序列生成器等对象，并且要定义约束条件保证数据的完整性。
  - (2) 要求用交互式方式输入 DML 命令完成对表的插入、更新、删除、以及各种条件的查询操作；

**实验内容：**

1. 按下列图创建相关基表

MEMBER

列名	MEMBER_ID	LAST_NAME	FIRST_NAME	ADDRESS	CITY	PHONE	JOIN_DATE
----	-----------	-----------	------------	---------	------	-------	-----------

Key Type	PK						
Null/Unique	NN,U	NN					NN
Default Value							SYSTEM DATE
Data Type	NUMBER	Varchar2	Varchar2	Varchar2	Varchar2	Varchar2	DATE
Length	10	25	25	100	30	15	

#### TITLE

列名	TITLE _ID	TITLE	DESCRIPTION	RATING	CITY	RELEASE _DATE
Key Type	PK					
Null/Unique	NN,U	NN				
Default Value				G,PG,R, NC17,NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMENT- TARY	
Data Type	NUMBER	Varchar2	Varchar2	Varchar2	Varchar2	DATE
Length	10	60	400	4	20	

#### TITLE\_COPY

列名	COPY _ID	TITLE_ID	STATUS
Key Type	PK	PK,FK	
Null/Unique	NN,U	NN,U	
Check			AVAILABLE, DESTROYED,

			RENTED, RESERVED
FK Ref Table		TITLE	
FK Ref Col		TITLE_ID	
Data Type	NUMBER	NUMBER	VARCHAR2
Length	10	10	155

#### RENTAL

列名	BOOK_ DATE	MEMBER_ID	COPY_ID	ACT_RET _DATE	EXP_RET _DATE	TITLE_ID
Key Type	PK	PK,FK1	PK,FK2			PK,FK2
Null/Unique	NN,U	NN				
FK Ref Table		MEMBER	TITLE_COPY			TITLE_COPY
FK Ref Col		MEMBER_ID	COPY_ID			TITLE_ID
Default Value	System Date				System Date + 2days	
Data Type	DATE	NUMBER	NUMBER	DATE	DATE	NUMBER
Length		10	10			10

#### RESERVATION

列名	RES_ DATE	MEMBER_ID	TITLE_ID
Key Type	PK	PK,FK1	PK,FK2
Null/Unique	NN,U	NN,U	NN
FK Ref Table		MEMBER	TITLE

FK Ref Col		MEMBER_ID	TITLE_ID
Data Type	DATE	NUMBER	NUMBER
Length		10	10

2. 从数据字典中确认上述创建命令，检查相关约束条件是否正确建立

3. 分别为 MEMBER 表、TITLE 表创建序列号

MEMBER\_ID\_SEQ: 从 101 开始，使用时不放入 cache

TITLE\_ID\_SEQ: 从 92 开始，使用时不放入 cache

查看相应的序列号信息

4. 将下列数据分别加入相应表中

TITLE(TITLE\_ID 列由序列号产生)

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends make a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-1995
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-1995
The Glob	A meteor crashes near a small American town and unleashes carnivorous goo in this classic.	NR	SCIFI	12-AUG-1995
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York.	PG	COMEDY	12-JUL-1995
Miracles on	A six-year-old has doubts	PG	DRAMA	12-SEP-1995

Ice	about Santa Claus, but she discovers that miracles really do exist.			
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-1995

#### MEMBER

First_Name	Last_Name	Address	City	Phone	Join_Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-1990
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-1990
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-1991
Mark	Quick-to-See	6921 King Way	Lagos	63-559-7777	07-APR-1990
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-1991
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-1991

#### TITLE\_COPY

Title	Copy_Id	Status
Willie and Christmas Too	1	AVAILABLE
Alien Again	1	AVAILABLE



	2	RENTED
The Glob	1	AVAILABLE
My Day Off	1	AVAILABLE
	2	AVAILABLE
	3	RENTED
Miracles on Ice	1	AVAILABLE
Soda Gang	1	AVAILABLE

#### RENTAL

<b>Title_ Id</b>	<b>Copy_ Id</b>	<b>Member _Id</b>	<b>Book_date</b>	<b>Exp_Ret_Date</b>	<b>Act_Ret_Date</b>
92	1	101	3 days ago	1 day ago	2 days ago
93	2	101	1 day ago	1 day from now	
95	3	102	2 days ago	Today	
97	1	106	4 days ago	2 days ago	2 days ago

5. 创建视图 TITLE\_AVAIL (包含 title、copy\_id、status、exp\_ret\_d 列)，列出电影名称、各拷贝的情况，如果已借出，则列出归还日期，并从中查询数据。

#### 6. 修改表结构和数据

(1) 向 TITLE 表中增加记录

TITLE: 'Interstellar Wars',

RATING: 'PG',

CATEGORY: 'scifi',

RELEASE DATE: 07—JUL—1977,

DESCRIPTION: 'Futuristic interstellar action movie.Can the rebels save the humans from the evil empire?'

(2) 向 RESERVATION 表增加记录

Carmen Velasquez, 预定租借'Interstellar Wars'

Mark Quick-to-See, 预定租借'Soda Gang'

(3) 修改表, 向 TITLE 增加列 PRICE NUMBER(8,2)

按下表修改 TITLE 中数据

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	30
Soda Gang	35
Interstellar Wars	29

为 PRICE 列增加 NOT NULL 约束

7. 执行脚本 oracle\_home\rdbms\admin\bdemobld.sql, 为当前的 scott 用户增加可用的练习表, 并完成下列查询语句。

(1) 列出 1986 年之后雇佣的员工

(2) 列出有提成的员工的 last name, job, salary, commission, 按照工资降序排序

(3) 如果将没有提成的员工的工资增长 10%, 其工资为多少, 请列出 (增长后的工资需四舍五入)

(4) 截至当前, 请将每个员工的工作年限、工作月数列出

(5) 请将 last name 以字母 J、K、L、M 开头的员工的姓名信息列出

(6) 请列出所有雇员信息, 包括 last name, salary, commission, 对于其中 commission 列, 如果员工有提成, 则显示“有”, 否则显示“无”

(7) 请将工作于纽约的员工的信息显示出来, 包括 department name,last name,job\_id,salary

(8) 以两种方法显示 last name 以 n 结尾的员工信息

(9) 列出部门的名称、location\_id 及每个部门的雇员数, 要将没有员工的部门也包括在内

(10) 列出员工上司有关信息, 包括员工姓名, 上司姓名, 上司工资, 上司的工

资级别。

## 实验 3：数据库编程

**实验目的：**通过该实验熟悉和掌握存储过程和触发器的开发步骤和调用或触发方法。

**实验背景：**必须学完 SQL 命令 DDL、DML 语句的语法，以及 ORACLE 的 PL/SQL 语言或 SQLSERVER 的 T-SQL 语言。

**实验设备：**一台 PC 机

**实验条件：**PC 机上必须安装 SQL SERVER2000 或 MYSQL 或 ORACLE9i 等数据库，并且已经执行脚本 oracle\_home\rdbms\admin\bdemobld.sql，为当前的 scott 用户增加可用的练习表。

**实验学时：**2 学时

**实验要求：**(1) 要求用 ORACLE 的 PL/SQL 语言或 SQLSERVER 的 T-SQL 语言编写存储过程、触发器、包。

(2) 要求查询相应的数据字典确认存储过程和触发器是否建立成功。

(3) 调用存储过程和包、触发触发器，检查执行结果。

**实验内容：**

1. 编写程序块，根据外部变量给出的部门编号，返回相应部门的员工数

其中：部门编号为外部变量在 iSQLPLUS 中使用 DEFINE 定义

输出信息为：部门 xx 有 xx 名员工

2. 编写程序块，根据外部变量给定的员工姓名，判断员工工资，如果工资<2500，则更新员工工资信息，salary+500，并且给出信息：xx 工资上涨 500；否则，给出信息：xx 工资达到标准

其中：员工姓名为外部变量在 iSQLPLUS 中使用 DEFINE 定义

3. 游标的使用

编写程序块，定义游标 emp\_cur，包含员工姓名、工资、雇佣日期

根据游标内记录判断，如果员工工资高于 3000，且于 1986 年 1 月 1 日后加入公司，则显示相应信息：xx 于 xx 年 xx 月 xx 日加入公司，现工资为 xx

4. 编写存储过程 Add\_dept，利用此过程向表 department 中添加记录，其中部门

编号、部门名称及部门所在地点编号都从外部变量输入，执行此过程向 `department` 表中添加数据，查询表中数据的改变

#### 5. 编写函数 `Get_service_years`，利用此过程获取指定雇员的工作年限 –19

输入参数为雇员编号，返回参数为工作年限

调用函数，获取有关雇员的工作信息

提示：

调用方法 `EXECUTE DBMS_OUTPUT.PUT_LINE('员工 7788 工作'||get_service_years(7788)||'年')`

#### 6. 存储单元中异常处理的练习

##### （1）处理系统预定义的异常

编写程序块，根据替换变量提供的工资信息获取雇员名称，如果根据输入工资返回的记录超过一条或者无返回记录，应该引发相应的异常处理，这两种异常是系统预定义的异常，分别为 `TOO_MANY_ROWS` 和 `NO_DATA_FOUND`

利用不同替换变量值，反应出两个异常出现的情况

##### （2）处理系统尚未定义的异常

编写存储，根据替换变量，修改 `employee` 表中雇员的部门编号

输入参数为：`empno`，`deptno`

定义异常 `e_constraint` 对应系统异常 2291（违反一致性约束）

在异常处理部分处理此异常

当新给出的部门编号不包含于 `department` 表中时，引发异常

##### （3）处理用户定义的异常

编写存储过程，利用输入参数向 `employee` 添加记录

输入参数为 `empno`，`ename`，`salary`，`job_id`，`department_id`

自定义异常：`invalid_salary`

判断输入的 `salary`，如果高于 5000，则引发异常 `invalid_salary`

#### 7. 编写程序包

程序包同过程与函数的编写略有不同，程序包被分成两个组成部分，包头和包体。其中包头中保护游客被外部直接调用的程序包的函数、过程的定义及可被外部调用的变量、异常等的定义，即公共部分的定义。包体则包括公共的过程、函数的

具体实现代码及私有的（即只在程序包内部有效的）过程、函数、变量、异常的代码。

范例：

编写程序包 JOB\_PACK，包含过程 ADD\_JOB，UPD\_JOB，DEL\_JOB 及函数 Q\_JOB

分别实现增加工作、更新工作、删除工作，查询工作  
调用格式为：

JOB\_PACK.ADD\_JOB(JOB\_ID,JOB\_NAME)

JOB\_PACK.UPD\_JOB(JOB\_ID,JOB\_NAME)

JOB\_PACK.DEL\_JOB(JOB\_ID,JOB\_NAME)

JOB\_NAME=JOB\_PACK.Q\_JOB(JOB\_ID)

```
CREATE OR REPLACE PACKAGE JOB_PACK IS
```

```
PROCEDURE ADD_JOB
```

```
(P_JOBID IN JOB.JOB_ID%TYPE,
```

```
P_JOBNAME IN JOB.FUNCTION%TYPE);
```

```
PROCEDURE UPD_JOB
```

```
(P_JOBID IN JOB.JOB_ID%TYPE,
```

```
P_JOBNAME IN JOB.FUNCTION%TYPE);
```

```
PROCEDURE DEL_JOB
```

```
(P_JOBID IN JOB.JOB_ID%TYPE);
```

```
FUNCTION Q_JOB
```

```
(P_JOBID IN JOB.JOB_ID%TYPE)
```

```
RETURN VARCHAR2;
```

```
END JOB_PACK;
```

```
CREATE OR REPLACE PACKAGE BODY JOB_PACK IS
```

```
PROCEDURE ADD_JOB
```

```
(P_JOBID IN JOB.JOB_ID%TYPE,P_JOBNAME IN JOB.FUNCTION%TYPE)
```

```
IS
```

```

BEGIN
INSERT INTO JOB VALUES(P_JOBID,P_JOBNAME);
END ADD_JOB;
PROCEDURE UPD_JOB
(P_JOBID IN JOB.JOB_ID%TYPE,P_JOBNAME IN JOB.FUNCTION%TYPE)
IS
BEGIN
UPDATE JOB SET FUNCTION=P_JOBNAME WHERE JOB_ID=P_JOBID;
IF SQL%NOTFOUND THEN
RAISE_APPLICATION_ERROR(-20202,'NO JOB UPDATE.');
```

```

END IF;
END UPD_JOB;
PROCEDURE DEL_JOB
(P_JOBID IN JOB.JOB_ID%TYPE)
IS
BEGIN
DELETE FROM JOB
WHERE JOB_ID=P_JOBID;
IF SQL%NOTFOUND THEN
RAISE_APPLICATION_ERROR(-20203,'NO JOB DELETE.');
```

```

END IF;
END DEL_JOB;
FUNCTION Q_JOB
(P_JOBID IN JOB.JOB_ID%TYPE)
RETURN VARCHAR2
IS
V_JOBNAME JOB.FUNCTION%TYPE;
BEGIN
SELECT FUNCTION INTO V_JOBNAME
WHERE JOB_ID=P_JOBID;
```

```
RETURN (V_JOBNAME);  
END Q_JOB;  
END JOB_PACK;
```

执行程序包 JOB\_PACK 中的过程、函数，观察返回结果

(1) 创建程序包 EMP\_MNG，其中包含过程 ADD\_EMP,UPD\_SAL,DEL\_EMP 和函数 EMP\_COUNT

说明，利用过程 ADD\_EMP 向 employee 增加记录，输入参数为 employee\_id,last\_name,first\_name,salary,job\_id,deptno,hiredate

利用过程 UPD\_SAL 修改员工工资信息，输入参数为 employee\_id,new\_sal，如果输入的员工 id 不存在，应进行相应的异常处理，提示没有此雇员

利用过程 DEL\_EMP 删除员工信息，输入参数为 employee\_id，如果输入的员工 id 不存在，应进行相应的异常处理，提示没有此雇员

利用函数 EMP\_COUNT，统计给定部门的雇员员工数，输入参数为部门编号 department\_id,如果输入部门编号不正确，请进行相应的异常处理，返回参数为员工数。

(2) 调用程序包内的过程、函数，获取。

## 实验 4：数据库访问接口 ODBC

实验目的：通过该实验熟悉和掌握前端开发语言与后台数据库连接。

实验背景：必须学完 JDBC 或 ODBC 接口。

实验设备：一台 PC 机

实验条件：PC 机上必须安装 SQL SERVER 2015 或 MYSQL 8.0 或 ORACLE Database 12c 等数据库，

实验学时：6 学时

实验要求：(1) 配置 SQL Server ODBC 驱动程序 (ODBC)

(2) 创建与删除 ODBC 数据源、分配 ODBC 句柄、设置特性

(3) 与 SQL Server 实例连接、执行查询以及处理结果

实验内容：

## 1、配置 SQL Server ODBC 驱动程序 (ODBC)

在使用 Microsoft® SQL Server™ 的 ODBC 应用程序之前，必须知道如何升级 SQL Server 早期版本中的目录存储过程的版本，以及如何添加、删除和测试数据源。

## 2、如何添加数据源 (ODBC)

可通过使用 ODBC 管理器、编程方式（使用 **SQLConfigDataSource**）或创建文件的方法添加数据源。

### （1）使用 ODBC 管理器添加数据源

- 在"开始"菜单中指向"设置"子菜单，然后单击"控制面板"命令。
- 双击"ODBC"。
- 单击"用户 DSN"、"系统 DSN"或"文件 DSN"选项卡，然后单击"添加"按钮。
- 单击 **SQL Server**，然后单击"完成"按钮。
- 完成向 SQL Server 新建数据源向导中的步骤。

### （2）编程方式添加数据源

- 调用 **SQLConfigDataSource**，调用时将 *fOption* 设置为 ODBC\_ADD\_DSN 或 ODBC\_ADD\_SYS\_DSN。

### （3）添加文件数据源

- 调用 **SQLDriverConnect**，调用时连接字符串中带参数 `SAVEFILE=file_name`。如果该连接成功，ODBC 驱动程序将在 SAVEFILE 参数所指向的位置创建带连接参数的文件数据源。

## 示例

### A. 使用 **SQLConfigDataSource** 创建数据源



```

#include <stdio.h>
#include <windows.h>
#include "sql.h"
#include <odbcinst.h>

int main()
{
    RETCODE retcode;

    UCHAR    *szDriver = "SQL Server";
    UCHAR    *szAttributes =
        "DSN=MyDSN\0DESCRIPTION=SQLConfigDSN Sample\0"
        "SERVER=MySQL\0ADDRESS=MyServer\0NETWORK=dbmssocn\0"
        "DATABASE=pubs\0";

    retcode = SQLConfigDataSource(NULL,
                                   ODBC_ADD_DSN,
                                   szDriver,
                                   szAttributes);
}

```

## B. 创建文件数据源

在 **SQLDriverConnect** 中使用 **SAVEFILE** 关键字创建文件数据源，然后使用 **SQLDriverConnect** 与该文件数据源进行连接。这是删除错误处理后的简化示例。

```

#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <odbcss.h>

#define MAXBUFLEN    255

```

```

SQLHENV      henv = SQL_NULL_HENV;
SQLHDBC      hdbc1 = SQL_NULL_HDBC;

int main() {

    RETCODE    retcode;

    // This format of the SAVEFILE keyword saves a successful
    // connection as the file Myfiledsn.dsn in the ODBC default
    // directory for file DSNs.
    SQLCHAR    szConnStrIn[MAXBUFLEN] =
                "SAVEFILE=MyFileDSN;DRIVER={SQL
Server};SERVER=MySQL;"
                "NETWORK=dbmssocn;UID=sa;PWD=MyPassWord;";

    SQLCHAR    szConnStrOut[MAXBUFLEN];
    SQLSMALLINT    cbConnStrOut = 0;

    // Allocate the ODBC Environment and save handle.
    retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);

    //Notify ODBC that this is an ODBC 3.0 application.
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                            (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_INTEGER);

    // Allocate an ODBC connection handle and connect.
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
    retcode = SQLDriverConnect(hdbc1,          // Connection handle
                               NULL,           // Window handle
                               szConnStrIn,     // Input connect string
                               SQL_NTS,        // Null-terminated string
                               szConnStrOut,    // Addr of output buffer
                               MAXBUFLEN,      // Size of output buffer
                               &cbConnStrOut,  // Address of output length

```

```

        SQL_DRIVER_NOPROMPT);

// Disconnect, set up a new connect string, and then test file DSN.
SQLDisconnect(hdbc1);
strcpy(szConnStrIn, "FILEDSN=MyFileDSN;UID=sa;PWD=MyPassWord;");
retcode = SQLDriverConnect(hdbc1,          // Connection handle
                           NULL,          // Window handle
                           szConnStrIn,    // Input connect string
                           SQL_NTS,       // Null-terminated string
                           szConnStrOut,    // Addr of output buffer
                           MAXBUFLen,     // Size of output buffer
                           &cbConnStrOut, // Address of output length
                           SQL_DRIVER_NOPROMPT);

/* Clean up. */
SQLDisconnect(hdbc1);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return(0);
}

```

### 3、如何删除数据源 (ODBC)

可通过使用 ODBC 管理器、编程方式（使用 **SQLConfigDataSource**）或删除文件来删除数据源。

#### (1) 使用 ODBC 管理器删除数据源

- 在"开始"菜单中指向"设置"子菜单，然后单击"控制面板"命令。
- 双击"32 位 ODBC"。
- 单击"用户 DSN"、"系统 DSN"或"文件 DSN"选项卡。
- 单击要删除的数据源。

- 单击"删除"按钮并确认删除。

## (2) 编程方式删除用户或系统数据源

- 调用 **SQLConfigDataSource**，调用时将 **fOption** 参数设置为 ODBC\_REMOVE\_DSN 或 ODBC\_REMOVE\_SYS\_DSN。

## (3) 删除文件数据源

- 在"开始"菜单中指向"设置"子菜单，然后单击"控制面板"命令。
- 双击"32 位 ODBC"。
- 单击"文件 DSN"。
- 单击要删除的文件 DSN。
- 单击"删除"按钮。

## 示例

此示例展示使用 **SQLConfigDataSource** 删除数据源。已经通过删除错误检查对其进行了简化。

```
#include <stdio.h>
#include <windows.h>
#include "sql.h"
#include <odbcinst.h>
```

```
int main() {
```

```
    RETCODE retcode;
```

```
    UCHAR    *szDriver = "SQL Server";
```

```
    UCHAR    *szAttributes = "DSN=MyFileDSN";
```

```
retcode = SQLConfigDataSource(NULL,  
                                ODBC_REMOVE_DSN,  
                                szDriver,  
                                szAttributes);
```

#### 4、与 SQL Server (ODBC) 连接

初始化 ODBC 应用程序包括分配环境和连接句柄，设置句柄特性以适应驱动程序和服务器的行为，然后与 Microsoft® SQL Server™ 2000 连接。

##### (1) 分配句柄并与 SQL Server 连接

- 加入 ODBC 头文件 `Sql.h`、`Sqllex.h` 和 `Sqltypes.h`。
- 加入 Microsoft® SQL Server™ 2000 驱动程序专用的头文件 `Odbcss.h`。
- 调用 **SQLAllocHandle**，调用时将 *HandleType* 设置为 `SQL_HANDLE_ENV` 以初始化 ODBC 并分配环境句柄。
- 调用 **SQLSetEnvAttr**，调用时将 *Attribute* 设置为 `SQL_ATTR_ODBC_VERSION` 并将 *ValuePtr* 设置为 `SQL_OV_ODBC3`，以表明该应用程序将使用 ODBC 3.x 格式的函数调用。
- 可以调用 **SQLSetEnvAttr** 设置其它环境选项或调用 **SQLGetEnvAttr** 获得环境选项（可选）。
- 调用 **SQLAllocHandle**，调用时将 *HandleType* 设置为 `SQL_HANDLE_DBC` 以分配连接句柄。
- 调用 **SQLSetConnectAttr** 设置连接选项或调用 **SQLGetConnectAttr** 获得连接选项（可选）。
- 调用 **SQLConnect** 使用现有数据源与 SQL Server 连接。

(2) 调用 **SQLDriverConnect** 使用连接字符串与 SQL Server 连接。

SQL Server 最小的完整连接字符串应具有下列两种形式之一：

**DSN=dsn\_name;UID=login\_id;PWD=password;**

**DRIVER={SQL Server};SERVER=server;UID=login\_id;PWD=password;**

如果连接字符串不完整，**SQLDriverConnect** 可提示所需信息。这由为 *DriverCompletion* 参数指定的值控制。

(3) 以迭代方式多次调用 **SQLBrowseConnect** 以生成连接字符串并与 SQL Server 连接。

- 调用 **SQLGetInfo** 获得驱动程序特性和 SQL Server 数据源的行为（可选）。
- 分配和使用语句。
- 调用 **SQLDisconnect** 与 SQL Server 断开，使该连接句柄可用于新连接。
- 调用 **SQLFreeHandle**，调用时将 *HandleType* 设置为 **SQL\_HANDLE\_DBC** 以释放该连接句柄。
- 调用 **SQLFreeHandle**，调用时将 *HandleType* 设置为 **SQL\_HANDLE\_ENV** 以释放该环境句柄。

## 示例

### A. 分配句柄，然后使用 **SQLConnect** 进行连接

此示例显示分配一个环境句柄和一个连接句柄，然后再使用 **SQLConnect** 进行连接的情况。已经通过删除许多错误检查对其进行了简化。

```
#include <stdio.h>
```

```

#include <string.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>
#include <odbcss.h>

SQLHENV      henv = SQL_NULL_HENV;
SQLHDBC      hdbc1 = SQL_NULL_HDBC;
SQLHSTMT     hstmt1 = SQL_NULL_HSTMT;

int main() {
    RETCODE retcode;
    UCHAR    szDSN[SQL_MAX_DSN_LENGTH+1] = "MyDSN",
            szUID[MAXNAME] = "sa",
            szAuthStr[MAXNAME] = "MyPassword";

    // Allocate the ODBC Environment and save handle.
    retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);

    // Notify ODBC that this is an ODBC 3.0 application.
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                            (SQLPOINTER)SQL_OV_ODBC3,
                            SQL_IS_INTEGER);

    // Allocate an ODBC connection handle and connect.
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
    retcode = SQLConnect(hdbc1, szDSN, (SQLINTEGER)strlen(szDSN),
                        szUID, (SQLINTEGER)strlen(szUID),
                        szAuthStr, (SQLINTEGER)strlen(szAuthStr));
    if ( (retcode != SQL_SUCCESS) &&
        (retcode != SQL_SUCCESS_WITH_INFO) ) {
        // Connect failed, call SQLGetDiagRec for errors.
    }
}

```

```

else {
    // Connects to SQL Server always return
    // informational messages. These messages can be
    // retrieved by calling SQLGetDiagRec.
}

// Allocate statement handles and do ODBC processing.

/* Clean up. */
SQLDisconnect(hdbc1);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return(0);
}

```

## B. 与 SQL Server 连接而不使用现有的 ODBC 数据源

此示例显示在不需要现有 ODBC 数据源的情况下调用 **SQLDriverConnect** 与 SQL Server 实例连接：

```

#define MAXBUFLen 255

SQLHENV      henv = SQL_NULL_HENV;
SQLHDBC      hdbc1 = SQL_NULL_HDBC;
SQLHSTMT     hstmt1 = SQL_NULL_HSTMT;

SQLCHAR      ConnStrIn[MAXBUFLen] =
    "DRIVER={SQL Server};SERVER=MyServer;"
    "UID=sa;PWD=MyPassWord;DATABASE=pubs;";

SQLCHAR      ConnStrOut[MAXBUFLen];
SQLSMALLINT  cbConnStrOut = 0;

// Make connection without data source. Ask that driver not
// prompt if insufficient information. Driver returns

```



```

// SQL_ERROR and application prompts user
// for missing information. Window handle not needed for
// SQL_DRIVER_NOPROMPT.
retcode = SQLDriverConnect(hdbc1,          // Connection handle
                           NULL,           // Window handle
                           ConnStrIn,      // Input connect string
                           SQL_NTS,        // Null-terminated string
                           ConnStrOut,     // Address of output buffer
                           MAXBUFLen,     // Size of output buffer
                           &cbConnStrOut, // Address of output length
                           SQL_DRIVER_NOPROMPT);

```

## 5、执行查询 (ODBC)

在 ODBC 应用程序中执行 SQL 语句需要分配语句句柄、设置语句特性以及准备和执行该 SQL 语句。

### (1) 如何使用语句 (ODBC)

- 调用 **SQLAllocHandle**，调用时将 *HandleType* 设置为 **SQL\_HANDLE\_STMT** 以分配语句句柄。
- 调用 **SQLSetStmtAttr** 设置语句选项或调用 **SQLGetStmtAttr** 获得语句特性（可选）。

若要使用服务器游标，必须将游标特性设置为其默认值以外的值。

- 如果要多次执行该语句，也可以用 **SQLPrepare** 准备要执行的语句（可选）。
- 如果该语句已绑定参数标记，也可以用 **SQLBindParameter** 将这些参数标记绑定到程序变量中（可选）。如果已准备好该语句，则可以调用 **SQLNumParams** 和 **SQLDescribeParam** 查找参数个数和参数特征。
- 使用 **SQLExecDirect** 直接执行语句。

(2) 如果已准备好语句，则可用 **SQLExecute** 多次执行该语句。

(3) 调用返回结果的方法有：

A. 将结果集列绑定到程序变量，使用 **SQLGetData** 将数据从结果集列移至程

序变量或者这两种方法的组合。 每次在语句的结果集内提取一行。

B. 使用块状光标，每次在该结果集内提取多行。

C. 调用 **SQLRowCount** 以确定受 INSERT、UPDATE 或 DELETE 语句影响的行数。

如果该 SQL 语句可以有多个结果集，则在每个结果集的结尾调用 **SQLMoreResults** 以查看是否有其它要处理的结果集。

(4) 处理完结果后，可能需要下列操作才能使得该语句句柄在执行新语句时可用：

- 如果在返回 **SQL\_NO\_DATA** 前未调用 **SQLMoreResults**，请调用 **SQLCloseCursor** 关闭该游标。
- 如果已将参数标记绑定到程序变量，请在调用 **SQLFreeStmt** 时将 *Option* 设置为 **SQL\_RESET\_PARAMS** 以释放绑定参数。
- 如果已将结果集列绑定到程序变量，请在调用 **SQLFreeStmt** 时将 *Option* 设置为 **SQL\_UNBIND** 以释放绑定列。
- 若要再次使用该语句句柄，请转到上述第2步骤。

(5) 调用 **SQLFreeHandle**

调用时将 *HandleType* 设置为 **SQL\_HANDLE\_STMT** 以释放该语句句柄。

## 6、如何设置游标选项 (ODBC)

## (1) 设置游标选项

- 调用 **SQLSetStmtAttr** 设置控制游标行为的语句选项，或调用 **SQLGetStmtAttr** 获得控制游标行为的语句选项。

<i>Foption</i>	指定
SQL_ATTR_CURSOR_TYPE	只进、静态、动态或键集驱动游标类型
SQL_ATTR_CONCURRENCY	只读、锁定、乐观使用时间戳或乐观使用值并发控制选项
SQL_ATTR_ROW_ARRAY_SIZE	每次提取所检索的行数
SQL_ATTR_CURSOR_SENSITIVITY	游标显示或不显示其它连接对游标行所做的更新
SQL_ATTR_CURSOR_SCROLLABLE	游标可以向前和向后滚动

- 这些特性的默认值（只进、只读、行集大小为 1）不使用服务器游标。若要使用服务器游标，则必须将这些特性中的至少一个特性设置为默认值之外的值，而且所执行的语句必须是单个 **SELECT** 语句，或者是包含单个 **SELECT** 语句的存储过程。使用服务器游标时，**SELECT** 语句无法使用服务器游标不支持的子句：**COMPUTE**、**COMPUTE BY**、**FOR BROWSE** 和 **INTO**。
- 通过设置 **SQL\_ATTR\_CURSOR\_TYPE** 和 **SQL\_ATTR\_CONCURRENCY**，或设置 **SQL\_ATTR\_CURSOR\_SENSITIVITY** 和 **SQL\_ATTR\_CURSOR\_SCROLLABLE**，可以控制所用游标的类型。不应将这两种指定游标行为的方法混合起来。

## 示例

A. 分配语句句柄，用行版本控制乐观并发设置动态游标类型，然后执行

## SELECT

```
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);
retcode = SQLSetStmtAttr(hstmt1, SQL_ATTR_CURSOR_TYPE,
                        (SQLPOINTER)SQL_CURSOR_DYNAMIC,
                        SQL_IS_INTEGER);
retcode = SQLSetStmtAttr(hstmt1, SQL_ATTR_CONCURRENCY,
                        (SQLPOINTER)SQL_CONCUR_ROWVER,
                        SQL_IS_INTEGER);
retcode = SQLExecDirect(hstmt1,
                        "SELECT au_lname FROM authors",
                        SQL_NTS);
```

### B. 分配语句句柄，设置可滚动、感知游标，然后执行 SELECT

```
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);
// Set the cursor options and execute the statement.
retcode = SQLSetStmtAttr(hstmt1, SQL_ATTR_CURSOR_SCROLLABLE,
                        (SQLPOINTER)SQL_SCROLLABLE,
                        SQL_IS_INTEGER);
retcode = SQLSetStmtAttr(hstmt1, SQL_ATTR_CURSOR_SENSITIVITY,
                        (SQLPOINTER)SQL_INSENSITIVE,
                        SQL_IS_INTEGER);
retcode = SQLExecDirect(hstmt1,
                        "select au_lname from authors",
                        SQL_NTS);
```

## (2) 如何直接执行语句 (ODBC)

### A. 一次性直接执行语句

- 如果该语句有参数标记，则使用 **SQLBindParameter** 将每个参数绑定到程序变量。用数据值填充程序变量，然后设置所有执行中的数据参数。
- 调用 **SQLExecDirect** 执行语句。

- 如果使用了执行中的数据输入参数，**SQLExecDirect** 将返回 **SQL\_NEED\_DATA**。用 **SQLParamData** 和 **SQLPutData** 发送数据块。

## B. 使用专用于列的参数绑定多次执行语句

调用 **SQLSetStmtAttr** 设置以下特性：

将 **SQL\_ATTR\_PARAMSET\_SIZE** 设置为参数集的个数 (S)。

将 **SQL\_ATTR\_PARAM\_BIND\_TYPE** 设置为 **SQL\_PARAMETER\_BIND\_BY\_COLUMN**。

将 **SQL\_ATTR\_PARAMS\_PROCESSED\_PTR** 特性设置为指向 **SQLINTEGER** 变量以保存已处理参数的数目。

将 **SQL\_ATTR\_PARAMS\_STATUS\_PTR** 设置为指向 **SQLUSMALLINT** 变量的数组 [S]，以保存参数状态指示符。

对于每个参数标记：

分配 S 参数数组缓冲区以存储数据值。

分配 S 参数数组缓冲区以存储数据长度。

调用 **SQLBindParameter** 以将参数数据值数组和数据长度数组绑定到该语句参数。

设置所有执行中的数据 **text** 或 **image** 参数。

将 S 数据值和 S 数据长度放入绑定参数数组。

调用 **SQLExecDirect** 执行语句。驱动程序将有效执行该语句 S 次，对于每个参数集执行一次。

如果使用了执行中的数据输入参数，**SQLExecDirect** 将返回

SQL\_NEED\_DATA。用 **SQLParamData** 和 **SQLPutData** 发送数据块。

### C. 使用专用于行的参数绑定多次执行语句

1. 分配一个结构数组 [S]，其中 S 是参数集的个数。此结构中每个参数有一个元素，每个元素有两部分：
  - 第一部分是一个可保存参数数据的相应数据类型的变量。
  - 第二部分是一个可保存状态指示符的 **SQLINTEGER** 变量。
2. 调用 **SQLSetStmtAttr** 设置以下特性：
  - 将 **SQL\_ATTR\_PARAMSET\_SIZE** 设置为参数集的个数 (S)。
  - 将 **SQL\_ATTR\_PARAM\_BIND\_TYPE** 设置为在步骤 1 中分配的结构大小。
  - 将 **SQL\_ATTR\_PARAMS\_PROCESSED\_PTR** 特性设置为指向 **SQLINTEGER** 变量以保存已处理参数的数目。
  - 将 **SQL\_ATTR\_PARAMS\_STATUS\_PTR** 设置为指向 **SQLUSMALLINT** 变量的数组 [S]，以保存参数状态指示符。
3. 对于每个参数标记，调用 **SQLBindParameter** 以将参数的数据值和数据长度指针指向它们在步骤 1 中分配的结构数组第一个元素中的变量。如果该参数是执行中的数据参数，请对其进行设置。
4. 用数据值填充绑定参数缓冲区数组。
5. 调用 **SQLExecDirect** 执行语句。驱动程序将有效执行该语句 S 次，对于每个参数集执行一次。
6. 如果使用了执行中的数据输入参数，**SQLExecDirect** 将返回 **SQL\_NEED\_DATA**。用 **SQLParamData** 和 **SQLPutData** 发送数据块。

通常，专用于列的绑定和专用于行的绑定较多与 **SQLPrepare** 和 **SQLExecute** 一起使用，而不与 **SQLExecDirect** 一起使用。

### 示例

此示例显示使用 **SQLExecDirect** 执行 SELECT 语句的情况。已经通过删除所有的错误检查对其进行了简化。

```
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <odbcss.h>

#define MAXBUFLLEN    255

SQLHENV        henv = SQL_NULL_HENV;
SQLHDBC        hdbc1 = SQL_NULL_HDBC;
SQLHSTMT        hstmt1 = SQL_NULL_HSTMT;

int main()
{
    RETCODE retcode;

    // SQLBindCol variables
    SQLCHAR        szName[MAXNAME+1];
    SQLINTEGER        cbName;

    // Allocate the ODBC Environment and save handle.
    retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
    // Notify ODBC that this is an ODBC 3.0 application.
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                            (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_INTEGER);
    // Allocate an ODBC connection and connect.
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
    retcode = SQLConnect(hdbc1,
                          "MyDSN", SQL_NTS,
                          "sa", SQL_NTS,
```

```

        "MyPassWord", SQL_NTS);

// Allocate a statement handle.
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);

// Execute an SQL statement directly on the statement handle.
// Uses a default result set because no cursor attributes are set.
retcode = SQLExecDirect(hstmt1,
                        "SELECT au_lname FROM authors",
                        SQL_NTS);

// Simplified result set processing. Bind one column and
// then fetch until SQL_NO_DATA.
retcode = SQLBindCol(hstmt1, 1, SQL_C_CHAR,
                    szName, MAXNAME, &cbName);
while ( (retcode = SQLFetch(hstmt1)) != SQL_NO_DATA )
    printf("Name = %s\n", szName);

/* Clean up. */
SQLFreeHandle(SQL_HANDLE_STMT, hstmt1);
SQLDisconnect(hdbc1);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return(0);
}

```

## 7. 如何准备和执行语句 (ODBC)

### A. 准备一次语句，然后多次执行该语句

1. 调用 **SQLPrepare** 准备语句。
2. 调用 **SQLNumParams** 确定已准备语句中的参数数目（可选）。
3. 对于已准备语句中的每个参数（可选）：
  - 调用 **SQLDescribeParam** 获得参数信息。



- 使用 **SQLBindParam** 将每个参数绑定到程序变量。设置所有执行中的数据参数。
4. 对于每个已准备语句的执行：
- 如果该语句有参数标记，则将数据值放入绑定参数缓冲区。
  - 调用 **SQLExecute** 执行已准备的语句。
  - 如果使用了执行中的数据输入参数，**SQLExecute** 将返回 **SQL\_NEED\_DATA**。用 **SQLParamData** 和 **SQLPutData** 发送数据块。

## B. 用专用于列的参数绑定准备语句

1. 调用 **SQLSetStmtAttr** 设置以下特性：
- 将 **SQL\_ATTR\_PARAMSET\_SIZE** 设置为参数集的个数 (S)。
  - 将 **SQL\_ATTR\_PARAM\_BIND\_TYPE** 设置为 **SQL\_PARAMETER\_BIND\_BY\_COLUMN**。
  - 将 **SQL\_ATTR\_PARAMS\_PROCESSED\_PTR** 特性设置为指向 **SQLINTEGER** 变量以保存已处理参数的数目。
  - 将 **SQL\_ATTR\_PARAMS\_STATUS\_PTR** 设置为指向 **SQLUSMALLINT** 变量的数组 [S]，以保存参数状态指示符。
2. 调用 **SQLPrepare** 准备语句。
3. 调用 **SQLNumParams** 确定已准备语句中的参数数目（可选）。
4. 对于已准备语句中的每个参数，调用 **SQLDescribeParam** 获得参数信息（可选）。
5. 对于每个参数标记：
- 分配 S 参数数组缓冲区以存储数据值。
  - 分配 S 参数数组缓冲区以存储数据长度。

- 调用 **SQLBindParameter** 以将参数数据值数组和数据长度数组绑定到该语句参数。
  - 如果该参数为执行中的数据 **text** 或 **image** 参数，则对其进行设置。
  - 如果使用了任何执行中的数据参数，则对它们进行设置。
6. 对于每个已准备语句的执行：
- 将 S 数据值和 S 数据长度放入绑定参数数组。
  - 调用 **SQLExecute** 执行已准备的语句。
  - 如果使用了执行中的数据输入参数，**SQLExecute** 将返回 **SQL\_NEED\_DATA**。用 **SQLParamData** 和 **SQLPutData** 发送数据块。

### C. 用专用于行的绑定参数准备语句

1. 分配一个结构数组 [S]，其中 S 是参数集的个数。此结构中每个参数有一个元素，每个元素有两部分：
  - 第一部分是一个可保存参数数据的相应数据类型的变量。
  - 第二部分是一个可保存状态指示符的 **SQLINTEGER** 变量。
2. 调用 **SQLSetStmtAttr** 设置以下特性：
  - 将 **SQL\_ATTR\_PARAMSET\_SIZE** 设置为参数集的个数 (S)。
  - 将 **SQL\_ATTR\_PARAM\_BIND\_TYPE** 设置为在步骤 1 中分配的结构大小。
  - 将 **SQL\_ATTR\_PARAMS\_PROCESSED\_PTR** 特性设置为指向 **SQLINTEGER** 变量以保存已处理参数的数目。
  - 将 **SQL\_ATTR\_PARAMS\_STATUS\_PTR** 设置为指向 **SQLUSMALLINT** 变量的数组 [S]，以保存参数状态指示符。

3. 调用 **SQLPrepare** 准备语句。
4. 对于每个参数标记，调用 **SQLBindParameter** 以将参数的数据值和数据长度指针指向它们在步骤 1 中分配的结构数组第一个元素中的变量。如果该参数是执行中的数据参数，请对其进行设置。
5. 对于每个已准备语句的执行：
  - 用数据值填充绑定参数缓冲区数组。
  - 调用 **SQLExecute** 执行已准备的语句。驱动程序将有效执行 SQL 语句 S 次，对于每个参数集执行一次。
  - 如果使用了执行中的数据输入参数，**SQLExecute** 将返回 **SQL\_NEED\_DATA**。用 **SQLParamData** 和 **SQLPutData** 发送数据块。

## 示例

此示例显示使用 **SQLPrepare** 和 **SQLExecute** 执行 **SELECT** 语句的情况。已经通过删除所有的错误检查对其进行了简化。

```
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <odbcss.h>

#define MAXBUFLLEN 255

SQLHENV      henv = SQL_NULL_HENV;
SQLHDBC      hdbc1 = SQL_NULL_HDBC;
SQLHSTMT     hstmt1 = SQL_NULL_HSTMT;

int main()
```

```

{
    RETCODE retcode;

    // SQLBindCol variables
    SQLCHAR      szName[MAXNAME+1];
    SQLINTEGER    cbName;


    // Allocate the ODBC Environment and save handle.
    retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
    // Notify ODBC that this is an ODBC 3.0 application.
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                            (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_INTEGER);
    // Allocate an ODBC connection and connect.
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
    retcode = SQLConnect(hdbc1,
                          "MyDSN", SQL_NTS, "sa", SQL_NTS,
                          "MyPassWord", SQL_NTS);


    // Allocate a statement handle.
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);
    // Prepare and execute an SQL statement on the statement handle.
    // Uses a default result set because no cursor attributes are set.
    retcode = SQLPrepare(hstmt1,
                          "SELECT au_lname from authors", SQL_NTS);
    retcode = SQLExecute(hstmt1);
    // Simplified result set processing. Bind one column and
    // then fetch until SQL_NO_DATA.
    retcode = SQLBindCol(hstmt1, 1, SQL_C_CHAR,
                          szName, MAXNAME, &cbName);
    while ( ( retcode = SQLFetch(hstmt1) ) != SQL_NO_DATA )
        printf("Name = %s\n", szName);


    /* Clean up. */
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt1);
    SQLDisconnect(hdbc1);
}

```

```

SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return(0);
}

```

## 8. 处理结果 (ODBC)

在 ODBC 应用程序中处理结果包括要首先确定结果集特征，然后使用 **SQLBindCol** 或 **SQLGetData** 将数据检索到程序变量。

### (1) 如何检索结果集信息 (ODBC)

1. 调用 **SQLNumResultCols** 获得该结果集内的列数。
2. 对于该结果集内的每列：
  - 调用 **SQLDescribeCol** 获得有关该结果列的信息。 或
  - 调用 **SQLColAttribute** 获得有关该结果列特定描述符的信息。

### (2) 处理结果

1. 检索结果集信息。
2. 如果使用绑定列，则对于要绑定到的每一列，调用 **SQLBindCol** 将程序缓冲区绑定到该列。
3. 对于该结果集内的每一行：
  - 调用 **SQLFetch** 获得下一行。
  - 如果使用绑定列，则使用目前在绑定列缓冲区中可用的数据。
  - 如果使用未绑定列，则一次或多次调用 **SQLGetData** 获得最后一个绑定列之后的未绑定列的数据。对 **SQLGetData** 的调用应按照列号递增的顺序进行。
  - 多次调用 **SQLGetData** 获得 **text** 列或 **image** 列中的数据。
4. 通过返回 **SQL\_NO\_DATA**，**SQLFetch** 发出结果集结束的信号时，调用

**SQLMoreResults** 确定是否还有其它结果集可用。

- 如果返回 **SQL\_SUCCESS**，表明有另一个结果集可用。
- 如果返回 **SQL\_NO\_DATA**，表明没有其它结果集可用。
- 如果返回 **SQL\_SUCCESS\_WITH\_INFO** 或 **SQL\_ERROR**，则调用 **SQLGetDiagRec** 确定 **PRINT** 或 **RAISERROR** 语句的输出是否可用。

如果绑定语句参数用于输出参数或存储过程的返回值，则使用目前在绑定参数缓冲区中可用的数据。同样，在使用绑定参数时，对 **SQLExecute** 或 **SQLExecDirect** 的每次调用都可能执行该 SQL 语句 S 次，其中 S 是绑定参数数组中的元素数。这意味着将有 S 个结果集要处理，其中每个结果集都包含通常执行单个 SQL 语句所返回的所有结果集、输出参数和返回代码。

注意，当结果集包含计算行时，每个计算行都可作为单独的结果集使用。这些计算结果集分散在普通行内，并将普通行断开到多个结果集内。

5. 调用 **SQLFreeStmt** 时将 *fOption* 设置为 **SQL\_UNBIND** 以释放所有绑定列的缓冲区（可选）。
6. 如果有另一个可用的结果集，请转到步骤 1。

若要在 **SQLFetch** 返回 **SQL\_NO\_DATA** 前取消处理结果集，请调用 **SQLCloseCursor**。

## 示例

此示例显示如何使用 **SQLBindCol** 或 **SQLGetData**。已经通过删除所有的错误检查对其进行了简化。该程序可在注释掉 **SQLBindCol** 函数或 **SQLGetData** 函数的情况下进行编译，所得到的可执行文件将进行相同的操作。

```
#include <stdio.h>
```

```
#include <string.h>
```

```

#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <odbcss.h>

#define MAXBUFLEN    255

SQLHENV      henv = SQL_NULL_HENV;
SQLHDBC      hdbc1 = SQL_NULL_HDBC;
SQLHSTMT     hstmt1 = SQL_NULL_HSTMT;

int main() {
    RETCODE retcode;
    // SQLBindCol variables
    SQLCHAR      szName[MAXNAME+1];
    SQLINTEGER    cbName;

    // Allocate the ODBC Environment and save handle.
    retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
    // Notify ODBC that this is an ODBC 3.0 application.
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                            (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_INTEGER);
    // Allocate an ODBC connection and connect.
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
    retcode = SQLConnect(hdbc1,
                          "MyDSN", SQL_NTS,"sa", SQL_NTS, "MyPassWord", SQL_NTS);

    // Allocate a statement handle.
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);
    // Execute an SQL statement directly on the statement handle.
    // Uses a default result set because no cursor attributes are set.
    retcode = SQLExecDirect(hstmt1,
                             "SELECT au_lname FROM authors", SQL_NTS);

```

```

// Simplified result set processing. Fetch until SQL_NO_DATA.
// The application can be compiled with the SQLBindCol line
// commented out to illustrate SQLGetData, or compiled with the
// SQLGetData line commented out to illustrate SQLBindCol.
// This sample shows that SQLBindCol is called once for the
// result set, while SQLGetData must be called once for each
// row in the result set.

retcode = SQLBindCol(hstmt1, 1, SQL_C_CHAR,
                    szName, MAXNAME, &cbName);
while ( (retcode = SQLFetch(hstmt1)) != SQL_NO_DATA ) {
//    SQLGetData(hstmt1, 1, SQL_C_CHAR, szName, MAXNAME, &cbName);
    printf("Name = %s\n", szName);
}

/* Clean up.*/
SQLFreeHandle(SQL_HANDLE_STMT, hstmt1);
SQLDisconnect(hdbc1);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
SQLFreeHandle(SQL_HANDLE_ENV, henv);

return(0);
}

```

## 使用游标 (ODBC)

若要使用游标，则必须首先设置控制 ODBC 游标行为的连接特性和语句特性。游标允许应用程序在每次进行提取操作时检索多行，并在游标的当前位置执行 UPDATE、INSERT 或 DELETE 语句。

## 如何使用游标 (ODBC)

### 使用游标



1. 调用 **SQLSetStmtAttr** 设置所需游标特性：

设置 **SQL\_ATTR\_CURSOR\_TYPE** 和 **SQL\_ATTR\_CONCURRENCY** 特性（这是首选选项）。

或

设置 **SQL\_CURSOR\_SCROLLABLE** 和 **SQL\_CURSOR\_SENSITIVITY** 特性。

2. 使用 **SQL\_ATTR\_ROW\_ARRAY\_SIZE** 特性调用 **SQLSetStmtAttr** 以设置行集大小。
3. 如果要使用 **WHERE CURRENT OF** 子句进行定位更新，则可调用 **SQLSetCursorName** 设置游标名称（可选）。
4. 执行 SQL 语句。
5. 如果要使用 **WHERE CURRENT OF** 子句进行定位更新，同时游标名称未在步骤 3 中通过 **SQLSetCursorName** 提供，则可调用 **SQLGetCursorName** 获得游标名称（可选）。
6. 调用 **SQLNumResultCols** 以获得行集内的列数 (C)。
7. 使用专用于列的绑定。或使用专用于行的绑定。
8. 按照需要提取游标中的行集。
9. 调用 **SQLMoreResults** 以确定是否还有其它结果集可用。
  - 如果返回 **SQL\_SUCCESS**，表明有另一个结果集可用。
  - 如果返回 **SQL\_NO\_DATA**，表明没有其它结果集可用。
  - 如果返回 **SQL\_SUCCESS\_WITH\_INFO** 或 **SQL\_ERROR**，则调用 **SQLGetDiagRec** 确定 **PRINT** 或 **RAISERROR** 语句的输出是否可用。

如果绑定语句参数用于输出参数或存储过程的返回值，则使用目前在绑定参数缓冲区中可用的数据。

在使用绑定参数时，对 **SQLExecute** 或 **SQLExecDirect** 的每次调用都将执行该 SQL 语句 S 次，其中 S 是绑定参数数组中的元素数。这意味着将有 S 个结果集要处理，其中每个结果集都包含通常执行单个 SQL 语句所返回的所有结果集、输出参数和返回代码。

注意，当结果集包含计算行时，每个计算行都可作为单独的结果集使用。这些计算结果集分散在普通行内，并将普通行断开到多个结果集内。

10. 调用 **SQLFreeStmt** 时将 *fOption* 设置为 **SQL\_UNBIND** 以释放所有绑定列的缓冲区（可选）。

11. 如果有另一个可用的结果集，请转到步骤 6。

在步骤 9 中，在部分处理过的结果集上调用 **SQLMoreResults** 将清除该结果集的剩余部分。清除已部分处理的结果集的另一个方法是调用 **SQLCloseCursor**。通过设置 **SQL\_ATTR\_CURSOR\_TYPE** 和 **SQL\_ATTR\_CONCURRENCY**，或设置 **SQL\_ATTR\_CURSOR\_SENSITIVITY** 和 **SQL\_ATTR\_CURSOR\_SCROLLABLE**，可以控制所用游标的类型。不应将这两种指定游标行为的方法混合起来。

## 如何使用行集绑定 (ODBC)

### 使用专用于列的绑定

1. 对于每个绑定列

- 分配一个（或多个）列缓冲区数组 R 以存储数据值，其中 R 是该行集内的行数。
- 分配一个（或多个）列缓冲区数组 R 以存储数据长度（可选）。
- 调用 **SQLBindCol** 将列的数据值数组和数据长度数组绑定到该行

集的列中。

2. 调用 **SQLSetStmtAttr** 设置以下特性：

- 将 **SQL\_ATTR\_ROW\_ARRAY\_SIZE** 设置为该行集的行数 (R)。
- 将 **SQL\_ATTR\_ROW\_BIND\_TYPE** 设置为 **SQL\_BIND\_BY\_COLUMN**。
- 将 **SQL\_ATTR\_ROWS\_FETCHED\_PTR** 特性设置为指向 **SQLINTEGER** 变量以保存所提取的行数。
- 将 **SQL\_ATTR\_ROW\_STATUS\_PTR** 设置为指向 **SQLUSMALLINT** 变量的数组 [R]，以保存行状态指示符。

3. 执行语句。

4. 对 **SQLFetch** 或 **SQLFetchScroll** 的每次调用都要检索 R 行，并将数据传输到绑定列。

### 使用专用于行的绑定

1. 分配一个结构数组 [R]，其中 R 是该行集的行数。此结构中每列有一个元素，每个元素有两部分：

- 第一部分是一个可保存列数据的相应数据类型的变量。
- 第二部分是一个可保存列状态指示符的 **SQLINTEGER** 变量。

2. 调用 **SQLSetStmtAttr** 设置以下特性：

- 将 **SQL\_ATTR\_ROW\_ARRAY\_SIZE** 设置为该行集的行数 (R)。
- 将 **SQL\_ATTR\_ROW\_BIND\_TYPE** 设置为在步骤 1 中分配的结构大小。
- 将 **SQL\_ATTR\_ROWS\_FETCHED\_PTR** 特性设置为指向 **SQLINTEGER** 变量以保存已提取行的数目。
- 将 **SQL\_ATTR\_PARAMS\_STATUS\_PTR** 设置为指向

SQLUSMALLINT 变量的数组 [R]，以保存行状态指示符。

3. 对于该结果集内的每一列，调用 **SQLBindCol** 将列的数据值和数据长度指针指向其在步骤 1 中分配的结构数组的第一个元素中的变量。
4. 执行语句。
5. 对 **SQLFetch** 或 **SQLFetchScroll** 的每次调用都要检索 R 行，并将数据传输到绑定列。

## 如何提取和更新行集 (ODBC)

### 提取和更新行集

1. 调用 **SQLSetStmtAttr** 时将 *fOption* 设置为 **SQL\_ROW\_ARRAY\_SIZE** 以更改该行集内的行数 (R) (可选)。
2. 调用 **SQLFetch** 或 **SQLFetchScroll** 获得行集。
3. 如果使用绑定列，则使用目前在绑定列缓冲区中对该行集可用的数据值和数据长度。

如果使用未绑定列，则对于每一行可在调用 **SQLSetPos** 时将 *Operation* 设置为 **SQL\_POSITION** 以设置该游标的位置，然后对于每个未绑定列：

- 一次或多次调用 **SQLGetData** 获得该行集最后一个绑定列之后的未绑定列的数据。对 **SQLGetData** 的调用应按照列号递增的顺序进行。
  - 多次调用 **SQLGetData** 获得 **text** 列或 **image** 列中的数据。
4. 设置所有执行中的数据 **text** 列或 **image** 列。
  5. 调用 **SQLSetPos** 或 **SQLBulkOperations** 以设置游标位置，在行集内刷新、更新、删除或添加行。

如果执行中的数据 **text** 列或 **image** 列用于更新或添加操作，则对这些

列进行处理。

6. 执行定位 UPDATE 或 DELETE 语句，指定游标名称（可从 **SQLGetCursorName** 中得到），在同一连接中使用不同的语句句柄（可选）。

## 执行事务 (ODBC)

在 ODBC 中，事务无法跨越连接。ODBC 应用程序可使用标准 ODBC 事务管理功能处理单个连接中的事务。ODBC 应用程序还可以使用 Microsoft 分布式事务处理协调器 (MS DTC) 将多个 Microsoft® SQL Server™ 连接放在单个事务中，即使这些连接是连接到不同的服务器。

## 运行存储过程 (ODBC)

Microsoft® SQL Server™ ODBC 驱动程序支持将存储过程作为远程存储过程执行。将存储过程作为远程存储过程执行使驱动程序和服务器得以优化存储过程的执行性能。

## 如何调用存储过程 (ODBC)

当 SQL 语句使用 ODBC CALL 转义子句调用存储过程时，Microsoft® SQL Server™ 驱动程序会使用远程存储过程调用 (RPC) 机制将此过程发送给 SQL Server。RPC 请求回避 SQL Server 中的许多语句分析和参数处理，因此比使用 Transact-SQL EXECUTE 语句的速度要快。

## 将过程作为 RPC 运行

1. 构造一个使用 ODBC CALL 转义序列的 SQL 语句。该语句对每个输入、输入/输出和输出参数使用参数标记，对过程使用返回值（若有）：

{? = CALL procname (?,?)}

2. 对每个输入、输入/输出和输出参数调用 **SQLBindParameter**，对过程调用返回值（若有）。

3. 用 **SQLExecDirect** 执行该语句。

**说明** 如果应用程序使用 Transact-SQL EXECUTE 语法（相对于 ODBC CALL 转义序列）提交过程，SQL Server ODBC 驱动程序将此过程调用（作为 SQL Server 语句而不是 RPC）传递给 SQL Server。而且，如果使用 Transact-SQL EXECUTE 语句，则不返回输出参数。

### 如何处理返回代码和输出参数 (ODBC)

Microsoft® SQL Server™ 存储过程可包含整型返回代码及输出参数。返回代码和输出参数在服务器发送的最后一个数据包中，在 **SQLMoreResult** 返回 SQL\_NO\_DATA 之前无法由应用程序使用。

#### 处理返回代码和输出参数

1. 构造一个使用 ODBC CALL 转义序列的 SQL 语句。该语句应对每个输入、输入/输出和输出参数使用参数标记，对过程使用返回值（若有）。
2. 对每个输入、输入/输出和输出参数调用 **SQLBindParameter**，对过程调用返回值（若有）。
3. 用 **SQLExecDirect** 执行该语句。
4. 直到 **SQLFetch** 或 **SQLFetchScroll** 在处理最后的结果集时返回 SQL\_NO\_DATA 或直到 **SQLMoreResults** 返回 SQL\_NO\_DATA 时才处理结果集。此时，用返回的数据值填充与返回代码和输出参数绑定在一起的变量。

### 示例

本例说明对返回代码和输出参数的处理。为简化本示例，查错代码已删除。

```
// CREATE PROCEDURE TestParm @OutParm int OUTPUT AS
// SELECT au_lname FROM pubs.dbo.authors
// SELECT @OutParm = 88
// RETURN 99
```

```

#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <odbcss.h>

#define MAXBUFLEN    255

SQLHENV      henv = SQL_NULL_HENV;
SQLHDBC      hdbc1 = SQL_NULL_HDBC;
SQLHSTMT     hstmt1 = SQL_NULL_HSTMT;

int main() {
    RETCODE retcode;

    // SQLBindParameter variables.
    SWORD     sParm1=0, sParm2=1;
    SDWORD    cbParm1=SQL_NTS, cbParm2=SQL_NTS;

    // Allocate the ODBC environment and save handle.
    retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
    // Notify ODBC that this is an ODBC 3.0 app.
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                           (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_INTEGER);
    // Allocate ODBC connection handle and connect.
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
    retcode = SQLConnect(hdbc1, "MyDSN", SQL_NTS,
                        "sa", SQL_NTS, "MyPassWord", SQL_NTS);

    // Allocate statement handle.
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);
    // Bind the return code to variable sParm1.
    retcode =

```

```

SQLBindParameter(hstmt1,1,SQL_PARAM_OUTPUT,SQL_C_SSHORT,
                  SQL_INTEGER,0,0,&sParm1,0,&cbParm1);
// Bind the output parameter to variable sParm2.
retcode =
SQLBindParameter(hstmt1,2,SQL_PARAM_OUTPUT,SQL_C_SSHORT,
                  SQL_INTEGER,0,0,&sParm2,0,&cbParm2);
// Execute the command.
retcode = SQLExecDirect(hstmt1, "{? = call TestParm(?)}", SQL_NTS);

// Show parameters are not filled.
printf("Before result sets cleared: RetCode = %d, OutParm = %d.\n",
       sParm1, sParm2);

// Clear any result sets generated.
while ( ( retcode = SQLMoreResults(hstmt1) ) != SQL_NO_DATA )
    ;

// Show parameters are now filled.
printf("After result sets drained: RetCode = %d, OutParm = %d.\n",
       sParm1, sParm2);

/* Clean up. */
SQLFreeHandle(SQL_HANDLE_STMT, hstmt1);
SQLDisconnect(hdbc1);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return(0);
}

```

## 管理 text 和 image 列 (ODBC)

Microsoft® SQL Server™ ODBC 驱动程序支持使用 **text** 和 **image** 参数以及从结果集的 **text**、**ntext** 和 **image** 列中检索数据。



## 如何使用执行中的数据参数 (ODBC)

### 使用执行中的数据 `text`、`ntext` 或 `image` 参数

1. 调用 **SQLBindParameter** 以将某个程序缓冲区与语句参数绑定在一起时：
  - 使用 `SQL_LEN_DATA_AT_EXEC(length)` 的 *pcbValue*，其中 *length* 是以字节为单位的 `text`、`ntext` 或 `image` 参数数据的总长度。
  - 使用程序定义参数标识符的 *rgbValue*。
2. 调用 **SQLExecDirect** 或 **SQLExecute** 将返回 `SQL_NEED_DATA`，这表明执行中的数据参数已准备好进行处理。
3. 对每个执行中的数据参数：
  - 调用 **SQLParamData** 以获得程序定义参数 ID。如果还有其它执行中的数据参数，将返回 `SQL_NEED_DATA`。
  - 一次或多次调用 **SQLPutData** 以发送参数数据，直到 *length* 全部送出。
4. 调用 **SQLParamData** 以表明最后执行中的数据参数的所有数据已全部送出。不返回 `SQL_NEED_DATA`。

### 示例

本例说明使用 **SQLPutData** 在执行中的数据文本参数中填充数据的方法。为简化本示例，查错代码已删除。

```
// Sample ODBC3 console application to write SQL_LONGVARCHAR data
// using SQLPutData.
// Assumes DSN has table:
//   SQLSrvr: CREATE TABLE emp3 (NAME char(30), AGE int,
//                               BIRTHDAY datetime, Memo1 text)

#include <stdio.h>
```

```

#include <string.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <odbcss.h>

#define TEXTSIZE    12000

SQLHENV    henv = SQL_NULL_HENV;
SQLHDBC    hdbc1 = SQL_NULL_HDBC;
SQLHSTMT    hstmt1 = SQL_NULL_HSTMT;

int main() {
    RETCODE retcode;

    // SQLBindParameter variables.
    SDWORD    cbTextSize, lbytes;
    //SQLParamData variable.
    PTR        pParmID;
    //SQLPutData variables.
    UCHAR    Data[] =
        "abcdefghijklmnpqrstuvwxyzabcdefghijklmnpqrstuvwxyz"
        "abcdefghijklmnpqrstuvwxyzabcdefghijklmnpqrstuvwxyz"
        "abcdefghijklmnpqrstuvwxyzabcdefghijklmnpqrstuvwxyz"
        "abcdefghijklmnpqrstuvwxyzabcdefghijklmnpqrstuvwxyz"
        "abcdefghijklmnpqrstuvwxyzabcdefghijklmnpqrstuvwxyz"
        "abcdefghijklmnpqrstuvwxyzabcdefghijklmnpqrstuvwxyz"
        "abcdefghijklmnpqrstuvwxyzabcdefghijklmnpqrstuvwxyz"
        "abcdefghijklmnpqrstuvwxyzabcdefghijklmnpqrstuvwxyz"
        "abcdefghijklmnpqrstuvwxyz";
    SDWORD    cbBatch = (SDWORD)sizeof(Data)-1;

    // Allocate the ODBC environment and save handle.

```

```

retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
// Notify ODBC that this is an ODBC 3.0 app.
retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_INTEGER);
// Allocate ODBC connection handle and connect.
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
retcode = SQLConnect(hdbc1, "MyDSN", SQL_NTS,
                    "sa", SQL_NTS, "MyPassWord", SQL_NTS);

// Allocate statement handle.
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);

// Set parameters based on total data to send.
lbytes = (SDWORD)TEXTSIZE;
cbTextSize = SQL_LEN_DATA_AT_EXEC(lbytes);
// Bind the parameter marker.
retcode = SQLBindParameter(hstmt1, // hstmt
                          1, // ipar
                          SQL_PARAM_INPUT, // fParamType
                          SQL_C_CHAR, // fCType
                          SQL_LONGVARCHAR, // FSqlType
                          lbytes, // cbColDef
                          0, // ibScale
                          (VOID *)1, // rgbValue
                          0, // cbValueMax
                          &cbTextSize); // pcbValue

// Execute the command.
retcode = SQLExecDirect(hstmt1,
                        "INSERT INTO emp3 VALUES('Paul Borm', 46,'1950-11-24 00:00:00', ?)",
                        SQL_NTS);
// Check to see if NEED_DATA; if yes, use SQLPutData.
retcode = SQLParamData(hstmt1, &pParmID);
if (retcode == SQL_NEED_DATA)

```

```

{
    while (lbytes > cbBatch)
    {
        SQLPutData(hstmt1, Data, cbBatch);
        lbytes -= cbBatch;
    }
    // Put final batch.
    SQLPutData(hstmt1, Data, lbytes);
}
else
{
    ProcessErrorMessages(SQL_HANDLE_STMT, hstmt1,
        "SQLPutData Failed\n\n");
    return(9);
}

// Make final SQLParamData call.
retcode = SQLParamData(hstmt1, &pParmID);

/* Clean up. */
SQLFreeHandle(SQL_HANDLE_STMT, hstmt1);
SQLDisconnect(hdbc1);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return(0);
}

```

## 如何使用执行中的数据列 (ODBC)

### 使用执行中的数据 text、ntext 或 image 列

1. 对每个执行中的数据列，在以前由 **SQLBindCol** 绑定的缓冲区中放入特殊值：

- 在 *pcbValue* 数据值缓冲区中放入

`SQL_LEN_DATA_AT_EXEC(length)`, 其中 *length* 是以字节为单位的 **text**、**ntext** 或 **image** 列数据的总长度。

- 在 *rgbValue* 数据长度缓冲区中放入由程序定义的列标识符。
2. 调用 **SQLSetPos** 返回 `SQL_NEED_DATA`, 这表明执行中的数据列已准备好进行处理。
  3. 对每个执行中的数据列:
    - 调用 **SQLParamData** 以获得列数组指针。如果还有其它执行中的数据列, 将返回 `SQL_NEED_DATA`。
    - 一次或多次调用 **SQLPutData** 以发送列数据, 直到 *length* 全部送出。
  4. 调用 **SQLParamData** 以表明最后执行中的数据列的所有数据已全部发送。不返回 `SQL_NEED_DATA`。

## 示例

本例说明使用 **SQLGetData** 从执行中的数据 **text** 列中检索数据的方法。删除了查错代码以简化本示例。

```
// Sample ODBC3 console application to read SQL_LONGVARCHAR
// data using SQLGetData.
// Assumes DSN has table:
//   SQLSrvr: CREATE TABLE emp3 (NAME char(30), AGE int,
//                               BIRTHDAY datetime, Memo1 text)
```

```
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <odbc.h>
```

```
#define TEXTSIZE      12000
```

```
SQLHENV      henv = SQL_NULL_HENV;
SQLHDBC      hdbc1 = SQL_NULL_HDBC;
SQLHSTMT     hstmt1 = SQL_NULL_HSTMT;

int main() {
    RETCODE retcode;
    SWORD    cntr;

    //SQLGetData variables.
    UCHAR    Data[BUFFERSIZE];
    SDWORD    cbBatch = (SDWORD)sizeof(Data)-1;
    SDWORD    cbTxtSize;

    // Clear data array.
    for(cntr = 0; cntr < BUFFERSIZE; cntr++)
        Data[cntr] = 0x00;

    // Allocate the ODBC environment and save handle.
    retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
    // Notify ODBC that this is an ODBC 3.0 app.
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                            (SQLPOINTER) SQL_OV_ODBC3,
                            SQL_IS_INTEGER);

    // Allocate ODBC connection handle and connect.
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
    retcode = SQLConnect(hdbc1, "MyDSN", SQL_NTS,
                          "sa", SQL_NTS, "MyPassWord", SQL_NTS);

    // Allocate statement handle; prepare, then execute command.
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);
    retcode = SQLExecDirect(hstmt1,
                            "SELECT Memo1 FROM emp3",
```

```

        SQL_NTS);

// Get first row.
retcode = SQLFetch(hstmt1);
// Get the SQL_LONG column.
cntr = 1;
do {
    retcode = SQLGetData(hstmt1,    // hstmt
        1,                          // ipar
        SQL_C_CHAR,                // fCType
        Data,                       // rgbValue
        cbBatch,                   // cbValueMax
        &cbTxtSize);               // pcbValue
    if ( retcode != SQL_NO_DATA ) {
        printf("GetData iteration %d, pcbValue = %d,\n",
            cntr++, cbTxtSize);
        printf("Data = %s\n\n", Data);
    }
} while (retcode != SQL_NO_DATA);

/* Clean up. */
SQLFreeHandle(SQL_HANDLE_STMT, hstmt1);
SQLDisconnect(hdbc1);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return(0);
} // End Main.

```

## 如何处理 ODBC 错误 (ODBC)

可以使用两个 ODBC 函数调用来检索 ODBC 信息: **SQLGetDiagRec** 和 **SQLGetDiagField**。若要获得 *SQLState*、*pfNative* 和 *ErrorMessage* 诊断字段中与 ODBC 有关的主要信息, 请调用 **SQLGetDiagRec**, 直到它返回 **SQL\_NO\_DATA** 为止。可对每个诊断记录调用 **SQLGetDiagField** 以检索个别

字段。所有驱动程序专用字段都必须用 **SQLGetDiagField** 检索。

**SQLGetDiagRec** 和 **SQLGetDiagField** 由 ODBC 驱动程序管理器而不是个别的驱动程序来处理。ODBC 驱动程序管理器在连接成功建立之前，不高速缓存驱动程序专用诊断字段。在连接成功建立之前无法对驱动程序专用诊断字段调用 **SQLGetDiagField**。这还包括 ODBC 连接命令，即便它们返回 **SQL\_SUCCESS\_WITH\_INFO**。驱动程序专用诊断字段在下一个 ODBC 函数调用之前不可用。

## 示例

本例说明一个简单的错误处理程序，此程序对标准的 ODBC 信息调用 **SQLGetDiagRec**，然后进行有效连接测试，如果有，则为 Microsoft® SQL Server™ ODBC 驱动程序专用诊断字段调用 **SQLGetDiagField**。

```
// Example of SQL Server ODBC driver-specific options
// on SQLGetDiagField.
//
// This application assumes the existence of the following
// stored procedure:
//
// CREATE PROCEDURE BadOne AS SELECT * FROM NotThere
//
// where no object named NotThere exists.
```

```
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <odbc.h>
```

```
#define MAXBUFLen 256
```

```
SQLHENV      henv = SQL_NULL_HENV;
```



```

SQLHDBC          hdbc1 = SQL_NULL_HDBC;
SQLHSTMT          hstmt1 = SQL_NULL_HSTMT;
char              logstring[MAXBUFLen] = "";

void              ProcessLogMessages(SQLSMALLINT plm_handle_type,
                                     SQLHANDLE plm_handle, char *logstring,
                                     int ConnInd);

int main() {
    RETCODE retcode;

    // Allocate the ODBC environment and save handle.
    retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);
    if( (retcode != SQL_SUCCESS_WITH_INFO) &&
        (retcode != SQL_SUCCESS)) {
        printf("SQLAllocHandle(Env) Failed\n\n");
        return(9);
    }

    // Notify ODBC that this is an ODBC 3.0 app.
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                           (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_INTEGER);
    if( (retcode != SQL_SUCCESS_WITH_INFO) &&
        (retcode != SQL_SUCCESS)) {
        printf("SQLSetEnvAttr(ODBC version) Failed\n\n");
        return(9);
    }

    // Allocate ODBC connection handle and connect.
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);
    if( (retcode != SQL_SUCCESS_WITH_INFO) &&
        (retcode != SQL_SUCCESS)) {
        printf("SQLAllocHandle(hdbc1) Failed\n\n");
        return(9);
    }

```

```

}
retcode = SQLConnect(hdbc1, "MyDSN", SQL_NTS,
    "sa", SQL_NTS, "MyPassWord", SQL_NTS);
if ( (retcode != SQL_SUCCESS) &&
    (retcode != SQL_SUCCESS_WITH_INFO) ) {
    ProcessLogMessages(SQL_HANDLE_DBC, hdbc1,
        "SQLConnect() Failed\n\n", FALSE);
    return(9);
}
else {
    ProcessLogMessages(SQL_HANDLE_DBC, hdbc1,
        "\nConnect Successful\n\n", FALSE);
}

// Allocate statement handle, and then execute command.
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);
if ( (retcode != SQL_SUCCESS) &&
    (retcode != SQL_SUCCESS_WITH_INFO) ) {
    ProcessLogMessages(SQL_HANDLE_DBC, hdbc1,
        "SQLAllocHandle(hstmt1) Failed\n\n",
        TRUE);
    return(9);
}
retcode = SQLExecDirect(hstmt1, "exec BadOne", SQL_NTS);
if ( (retcode != SQL_SUCCESS) &&
    (retcode != SQL_SUCCESS_WITH_INFO) ) {
    ProcessLogMessages(SQL_HANDLE_STMT, hstmt1,
        "SQLExecute() Failed\n\n", TRUE);
    return(9);
}
// Clear any result sets generated.
while ( ( retcode = SQLMoreResults(hstmt1) ) != SQL_NO_DATA )
    ;

```

```

/* Clean up. */
SQLFreeHandle(SQL_HANDLE_STMT, hstmt1);
SQLDisconnect(hdbc1);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return(0);
}

void ProcessLogMessages(SQLSMALLINT plm_handle_type,
                        SQLHANDLE plm_handle,
                        char *logstring, int ConnInd)
{
    RETCODE      plm_retcode = SQL_SUCCESS;
    UCHAR        plm_szSqlState[MAXBUFLen] = "",
                plm_szErrorMsg[MAXBUFLen] = "";
    SDWORD       plm_pfNativeError = 0L;
    SWORD        plm_pcbErrorMsg = 0;
    SQLSMALLINT   plm_cRecNbr = 1;
    SDWORD        plm_SS_MsgState = 0, plm_SS_Severity = 0;
    SQLINTEGER    plm_Rownumber = 0;
    USHORT       plm_SS_Line;
    SQLSMALLINT   plm_cbSS_Procname, plm_cbSS_Srvname;
    SQLCHAR       plm_SS_Procname[MAXNAME],
plm_SS_Srvname[MAXNAME];

    printf(logstring);

    while (plm_retcode != SQL_NO_DATA_FOUND) {
        plm_retcode = SQLGetDiagRec(plm_handle_type, plm_handle,
            plm_cRecNbr, plm_szSqlState, &plm_pfNativeError,
            plm_szErrorMsg, MAXBUFLen - 1, &plm_pcbErrorMsg);

        // Note that if the application has not yet made a
        // successful connection, the SQLGetDiagField

```

```

// information has not yet been cached by ODBC
// Driver Manager and these calls to SQLGetDiagField
// will fail.
if (plm_retcode != SQL_NO_DATA_FOUND) {
    if (ConnInd) {
        plm_retcode = SQLGetDiagField(
            plm_handle_type, plm_handle, plm_cRecNmbr,
            SQL_DIAG_ROW_NUMBER, &plm_Rownumber,
            SQL_IS_INTEGER,
            NULL);
        plm_retcode = SQLGetDiagField(
            plm_handle_type, plm_handle, plm_cRecNmbr,
            SQL_DIAG_SS_LINE, &plm_SS_Line,
            SQL_IS_INTEGER,
            NULL);
        plm_retcode = SQLGetDiagField(
            plm_handle_type, plm_handle, plm_cRecNmbr,
            SQL_DIAG_SS_MSGSTATE, &plm_SS_MsgState,
            SQL_IS_INTEGER,
            NULL);
        plm_retcode = SQLGetDiagField(
            plm_handle_type, plm_handle, plm_cRecNmbr,
            SQL_DIAG_SS_SEVERITY, &plm_SS_Severity,
            SQL_IS_INTEGER,
            NULL);
        plm_retcode = SQLGetDiagField(
            plm_handle_type, plm_handle, plm_cRecNmbr,
            SQL_DIAG_SS_PROCNAME, &plm_SS_Procname,
            sizeof(plm_SS_Procname),
            &plm_cbSS_Procname);
        plm_retcode = SQLGetDiagField(
            plm_handle_type, plm_handle, plm_cRecNmbr,
            SQL_DIAG_SS_SRVNAME, &plm_SS_Srvname,
            sizeof(plm_SS_Srvname),

```

```

        &plm_cbSS_Srvname);
    }
    printf("szSqlState = %s\n",plm_szSqlState);
    printf("pfNativeError = %d\n",plm_pfNativeError);
    printf("szErrorMsg = %s\n",plm_szErrorMsg);
    printf("pcbErrorMsg = %d\n\n",plm_pcbErrorMsg);
    if (ConnInd) {
        printf("ODBCRowNumber = %d\n", plm_Rownumber);
        printf("SSrvrLine = %d\n", plm_Rownumber);
        printf("SSrvrMsgState = %d\n",plm_SS_MsgState);
        printf("SSrvrSeverity = %d\n",plm_SS_Severity);
        printf("SSrvrProcname = %s\n",plm_SS_Procname);
        printf("SSrvrSrvname = %s\n\n",plm_SS_Srvname);
    }
}
plm_cRecNmbr++; //Increment to next diagnostic record.
} // End while.
}

```

## Using Catalog Functions

All databases have a structure containing the data stored in the database. A definition of this structure, along with other information such as permissions, is stored in a catalog (implemented as a set of system tables), also known as a data dictionary.

The Microsoft® SQL Server™ ODBC driver enables an application to determine the database structure through calls to ODBC catalog functions. Catalog functions return information in result sets and are implemented using catalog stored procedures to query the system tables in the catalog. For example, an application might request a result set containing information about all the tables on the system or all the columns

in a particular table. The standard ODBC catalog functions are used to get catalog information from the SQL Server to which the application connected.

SQL Server supports distributed queries in which data from multiple, heterogeneous OLE DB data sources is accessed in a single query. One of the methods of accessing a remote OLE DB data source is to define the data source as a linked server. This can be done by using **sp\_addlinkserver**. After the linked server has been defined, objects in that server can be referenced in Transact-SQL statements by using a four part name:

*linked\_server\_name.catalog.schema.object\_name*

The SQL Server ODBC driver supports two driver-specific functions that help get catalog information from linked servers:

- **SQLLinkedServers**

Returns a list of the linked servers defined to the local server.

- **SQLLinkedCatalogs**

Returns a list of the catalogs contained in a linked server.

After you have a linked server name and a catalog name, the SQL Server ODBC driver supports getting information from the catalog by using a two part name of *linked\_server\_name.catalog* for *CatalogName* on the following ODBC catalog functions:

<b>SQLColumnPrivileges</b>	<b>SQLColumns</b>	<b>SQLPrimaryKeys</b>
<b>SQLStatistics</b>	<b>SQLTablePrivileges</b>	<b>SQLTables</b>

The two part *linked\_server\_name.catalog* is also supported for *FKCatalogName* and *PKCatalogName* on **SQLForeignKeys**.

Using **SQLLinkedServers** and **SQLLinkedCatalogs** requires the following files:

- Odbcss.h

Includes function prototypes and constant definitions for the linked server catalog functions. Odbcss.h must be included in the ODBC application and must be in the include path when the application is compiled.

- Odbcbcpl.lib

Must be in the library path of the linker and specified as a file to be linked. Odbcbcpl.lib is distributed with the SQL Server ODBC driver.

- Odbcbcpl.dll

Must be present at execution time. Odbcbcpl.dll is distributed with the SQL Server ODBC driver.

## SQLTables

When restricted to the current database, **SQLTables** executes the Transact-SQL procedure **sp\_tables** to report table catalog data for Microsoft® SQL Server™.

The following table shows **SQLTables** parameter mapping for **sp\_tables** stored procedure execution.

SQLTables parameter name	sp_tables parameter name
<i>CatalogName</i>	<i>table_qualifier</i>
<i>SchemaName</i>	<i>table_owner</i>
<i>TableName</i>	<i>table_name</i>
<i>TableType</i>	<i>table_type</i>

**SQLTables** can be executed on a static server cursor. An attempt to execute

**SQLTables** on an updatable (dynamic or keyset) cursor will return

SQL\_SUCCESS\_WITH\_INFO indicating that the cursor type has been changed.

**SQLTables** reports tables from all databases when the *CatalogName* parameter is SQL\_ALL\_CATALOGS and all other parameters contain default values (NULL pointers). **SQLTables** does not make use of **sp\_tables** in this special case.

To report available catalogs, schemas, and table types, **SQLTables** makes special use of empty strings (zero-length byte pointers). Empty strings are not default values (NULL pointers).

The SQL Server ODBC driver supports reporting information for tables on linked servers by accepting a two-part name for the *CatalogName* parameter:

*Linked\_Server\_Name.Catalog\_Name.*

**SQLTables** returns information about any tables whose names match *TableName* and are owned by the current user.

## 结果集

列名	数据类型	描述
TABLE_QUALIFIER	sysname	表限定符名称。在 SQL Server 中，该列表示数据库名。该字段可以为 NULL。
TABLE_OWNER	sysname	表所有者名称。在 SQL Server 中，该列表示创建表的数据库用户的姓名。该字段始终返回值。
TABLE_NAME	sysname	表名。该字段始终返回值。
TABLE_TYPE	varchar(32)	表、系统表或视图。
REMARKS	varchar(254)	SQL Server 不为该列返回值。

## Example

```
// Get a list of all tables in the current database.
```



```

SQLTables(hstmt, NULL, 0, NULL, 0, NULL, 0, NULL,0);
// Get a list of all tables in all databases.
SQLTables(hstmt, (SQLCHAR*) "%", SQL_NTS, NULL, 0, NULL, 0, NULL,0);
// Get a list of databases on the current connection's server.
SQLTables(hstmt, (SQLCHAR*) "%", SQL_NTS, (SQLCHAR*)"", 0,
(SQLCHAR*)"",
0, NULL, 0);

```

## SQLColumns

**SQLColumns** executes the Transact-SQL procedure **sp\_columns** to report catalog data for database columns.

The following table shows **SQLColumns** parameter mapping for **sp\_columns** stored procedure execution.

SQLColumns parameter name	sp_columns parameter name
<i>CatalogName</i>	<i>object_qualifier</i>
<i>SchemaName</i>	<i>object_owner</i>
<i>TableName</i>	<i>object_name</i>
<i>ColumnName</i>	<i>column_name</i>

**SQLColumns** returns SQL\_SUCCESS whether or not values exist for the *CatalogName*, *TableName*, or *ColumnName* parameters. **SQLFetch** returns SQL\_NO\_DATA when invalid values are used in these parameters.

**SQLColumns** can be executed on a static server cursor. An attempt to execute **SQLColumns** on an updatable (dynamic or keyset) cursor will return SQL\_SUCCESS\_WITH\_INFO indicating that the cursor type has been changed.

The Microsoft® SQL Server™ ODBC driver supports reporting information for tables on linked servers by accepting a two-part name for the *CatalogName* parameter: *Linked\_Server\_Name.Catalog\_Name*.

For ODBC 2.x applications not using wildcards in *TableName*, **SQLColumns** returns information about any tables whose names match *TableName* and are owned by the current user. If the current user owns no table whose name matches the *TableName* parameter, **SQLColumns** returns information about any tables owned by other users where the table name matches the *TableName* parameter. For ODBC 2.x applications using wildcards, **SQLColumns** returns all tables whose names match *TableName*. For ODBC 3.x applications **SQLColumns** returns all tables whose names match *TableName*

结果集

**sp\_columns** 目录存储过程与 ODBC 中的 **SQLColumns** 等价。返回的结果按 **TABLE\_QUALIFIER**、**TABLE\_OWNER** 和 **TABLE\_NAME** 排序。

列名	数据类型	描述
<b>TABLE_QUALIFIER</b>	<b>sysname</b>	表或视图限定符的名称。该字段可以为 NULL。
<b>TABLE_OWNER</b>	<b>sysname</b>	表或视图所有者的名称。该字段始终返回值。
<b>TABLE_NAME</b>	<b>sysname</b>	表或视图的名称。该字段始终返回值。
<b>COLUMN_NAME</b>	<b>sysname</b>	所返回的 <b>TABLE_NAME</b> 每列的列名。该字段始终返回值。
<b>DATA_TYPE</b>	<b>smallint</b>	ODBC 数据类型的整型代码。如果数据类型无法映射到 ODBC 类型，则为 NULL。本机数据类型名称在 <b>TYPE_NAME</b> 列中返回。
<b>TYPE_NAME</b>	<b>varchar(13)</b>	表示数据类型的字符串。基础 DBMS 提供此数据类型的名称。
<b>PRECISION</b>	<b>int</b>	有效数字个数。 <b>PRECISION</b> 列的返回

		值以 10 为基数。
<b>LENGTH</b>	<b>int</b>	数据的传输大小。 <sup>1</sup>
<b>SCALE</b>	<b>smallint</b>	小数点右边的数字个数。
<b>RADIX</b>	<b>smallint</b>	数字数据类型的基数。
<b>NULLABLE</b>	<b>smallint</b>	指定是否可以为空。  1 = 可以为 NULL。 0 = NOT NULL。
<b>REMARKS</b>	<b>varchar(254)</b>	该字段总是返回 NULL。
<b>COLUMN_DEF</b>	<b>nvarchar(4000)</b>	列的默认值。
<b>SQL_DATA_TYPE</b>	<b>smallint</b>	SQL 数据类型出现在描述符的 TYPE 字段时的值。该列与 <b>DATA_TYPE</b> 列相同， <b>datetime</b> 和 SQL-92 <b>interval</b> 数据类型除外。该列始终返回值。
<b>SQL_DATETIME_SUB</b>	<b>smallint</b>	<b>datetime</b> 及 SQL-92 <b>interval</b> 数据类型的子类型代码。对于其它数据类型，该列返回 NULL。
<b>CHAR_OCTET_LENGTH</b>	<b>int</b>	字符或整型数据类型的列的最大字节长度。对于所有其它数据类型，该列返回 NULL。
<b>ORDINAL_POSITION</b>	<b>int</b>	列在表中的顺序位置。表中的第一列为 1。该列始终返回值。
<b>IS_NULLABLE</b>	<b>varchar(254)</b>	表中的列是否可以为空。根据 ISO 规则决定是否可以为空。遵从 ISO SQL 标准的 DBMS 无法返回空字符串。  YES = 列可以包含 NULL。

		<p>NO = 列不能包含 NULL。</p> <p>如果不知道是否可以为空，该列则返回零长度字符串。</p> <p>该列返回的值与 <b>NULLABLE</b> 列返回的值不同。</p>
<b>SS_DATA_TYPE</b>	<b>tinyint</b>	<p>SQL Server 数据类型，由开放式数据服务扩展存储过程使用。有关更多信息，请参见<a href="#">数据类型</a>。</p>

## 实验5、实用数据库应用系统的设计与开发

**实验目的：**掌握数据库应用系统的开发过程，熟悉数据库设计、界面设计、开发语言的使用，以及开发语言与数据库的接口。

**实验背景：**必须学完 JDBC 或 ODBC 接口，并且学习前端开发语言 JSP。

**实验设备：**一台 PC 机

**实验条件：**PC 机上必须安装 SQL SERVER 2015 或 MYSQL 8.0 或 ORACLE Database 12c 等数据库，

**实验学时：**20 学时

**实验要求：**(1) 数据库表结构设计、以及脚本文件的编写

(2) 应用界面的设计、

(3) 利用 JDBC、ODBC 或专用的数据接口创建数据库连接。

(4) 最后完成应用程序包，建立安装程序。

**实验内容：**饭店前台登记系统（简化版）（仅为例子，可换其它应用背景）

### （一）需求分析

- 1.方便的登记客人信息(尽量减少日常录入工作量)包括:查询,登记信息相互参考.能选择输入的不录入输入,如性别,日期.
- 2.按照公安部要求记录国籍,证件标准代码.自设标准代码,保证参照完整性.
- 3.通用系统,不受具体房间号限制.

#### 4.方便,灵活查询,修改

包括:房间状态查询,更改,客人情况组合查询,房价统一更改(及特例更改)

### (二) 数据库表结构

#### 1.客人信息表 guests

(roomcode(房间编号),name,sex,age,  
certclassno(证件种类编号),  
certificate(证件号),countryno(国籍编码),  
arrdate( 抵达日期),leftdate(拟走日期),  
remarks 备注,(price 房价))

#### 2.房间状态表 roomstate

(roomcode,roomstste(房间状态))

- 房间状态      r:可租 z:整理 o:故障 l:住人
- 暂表达一天房态

#### 3.房间类型表 roomclass

(classno(分类号),class(分类名),  
beds(床位数),price (定价),  
remarks (备注))

分类: (豪华套,总统套,标准间,三人间,单人间)

#### 4.房间配置表 room

(roomcode(房间编号),roomno(房间号码),  
classno(分类号))

#### 5.国籍编码表 countrycodes

(countryno(国籍编码), country (国籍))

其中:

countryno(国籍编码) char(2) 大写字母

country (国籍) char(20)

#### 6.证件编码表 certclasses

( certno (证件编码),  
certclass (证件名称))

其中:

certno (证件编码): char(2)

certclass (证件名称): char(16)

### (三) 功能要求

1.初始化: 将房态全部置为 R,客人表清空,若已有客人可提示,选择,(注意数据一致性)

2.客人登记

(1) 首先选择房类,了解房价,该类房空房数,空床数.

(2) 分配房号(是否修改房态?)

(3) 登记客人信息(注意数据完整性,一致性)

(4) 房号,证件,国籍要选取(显示汉字或房号,保存代码)

3.代码维护

对代码表能录入,查询,修改,删除

4.退房及换房

(1) 新房有空床或为空房?

(2) 客人信息与房态一致

换房或退房后,房间状态为"Z"

5.查询打印

(1) 查询各类房态(按类,按房,按层)并有统计信息

(2) 查客人情况(各种情况),并有统计信息

6.修改房态

允许两种修改:

'Z'⇄'R'

'O'⇄'R'

### (四) 实验报告

1.数据字典

表名(中,英),属性名,类型,大小, 含义, 取值域,约束,输入方式(来自其他表,手工输入),关键字

2.功能设计及使用说明

3.以 Form 为单位描述其名称,重要属性及设置,与数据关系,与功能流程关系.

4.写出两个以上查询的 SQL 语句

5.安装程序,使用说明书

程序+报告---◇ 软盘

盘上注明：班级、记分册上序号、姓名

### (五) 应用程序界面示例说明（参考，不要求）

#### 1. 初始状态界面

说明：

1) 上面类型列表数据来自表 roomclass

2) 房间详细说明中前 5 项数据来自表 roomclass，最后一项数据来自计算和统计；

#### 2. 房间详细情况



房间详细情况

房间类型： Combo1

房间号：

房间状态

☐ 可租 ☐ 住人

☐ 故障 ☐ 整理

房间配置

客人情况

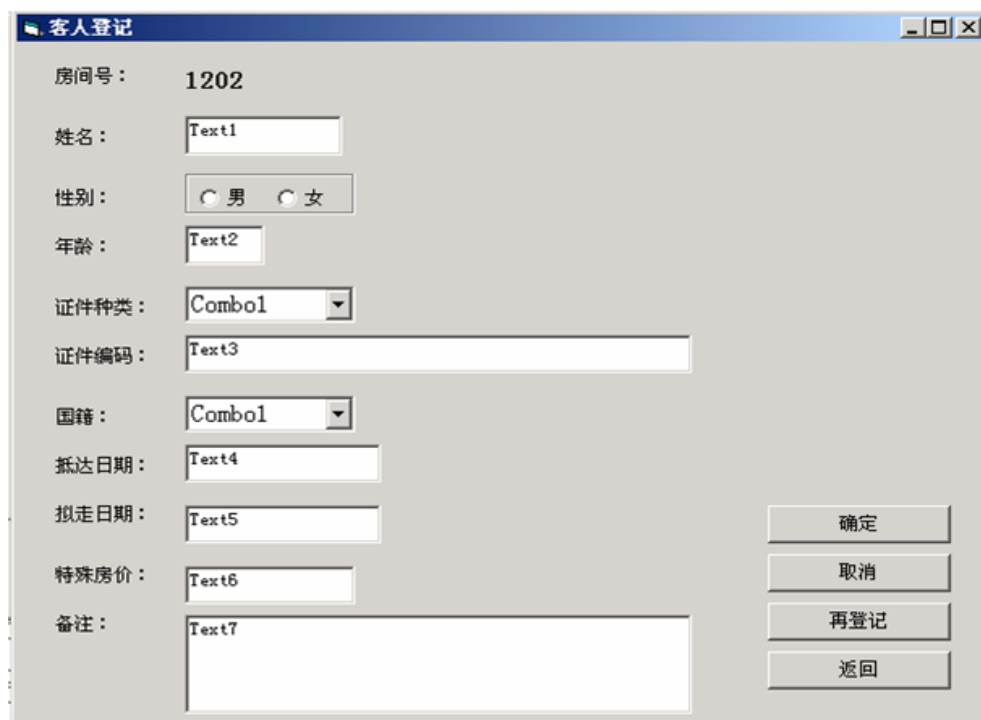
入住登记

返回

说明：

- 1) 房间类型数据来自表 roomclass + ‘全部’，可单选房类；
- 2) 右上框“房间状态”可单选，也可多选，选择后左下侧的房间号列表显示
- 3) 根据查看的房类及房态的选择显示房间号，数据来自表 room 及 guests；
- 4) 当选中一个房间号时可进行查看其房间配置、查看其客人情况、入住该房间的客人登记

### 3. 客人登记



客人登记

房间号： 1202

姓名： Text1

性别： ☐ 男 ☐ 女

年龄： Text2

证件种类： Combo1

证件编码： Text3

国籍： Combo1

抵达日期： Text4

拟走日期： Text5

特殊房价： Text6

备注： Text7

确定

取消

再登记

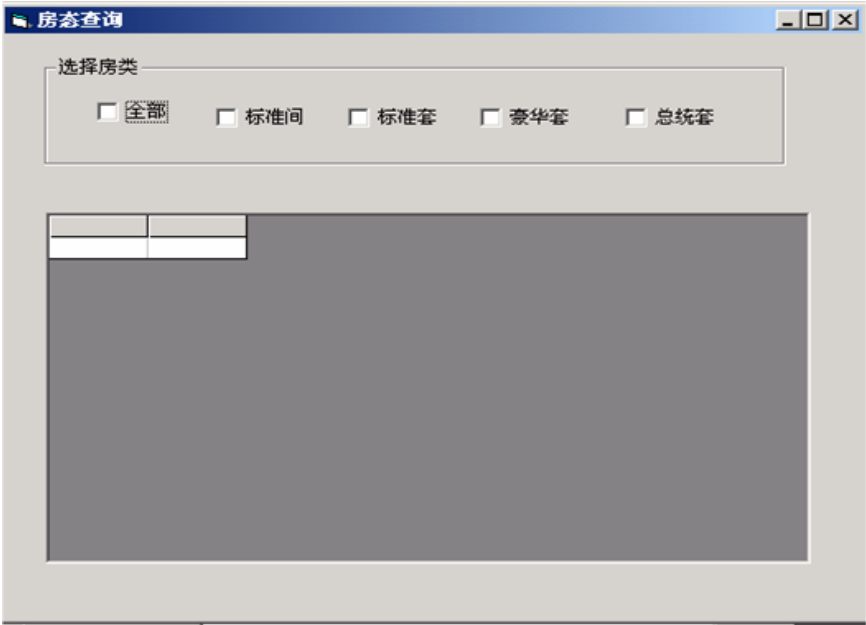
返回



说明：

- 1) 房间号不能在此输入，是在 2 中界面选定后带入的；
- 2) 证件种类：界面显示汉字，存入表 `guests` 中为证件种类代码；国籍与此相同；
- 3) 抵达日期与拟走日期要限制正确日期，并且拟走日期要大于抵达日期；
- 4) 确定钮：把客人信息存入表 `guests`，并且检查数据填写是否完整正确，如有错误出提示，并不存库；
- 5) 取消钮：清空界面；
- 6) 再登记：同一房间下一客人信息，房号不变；

4. 房态查询



说明：

- 1) 下面的表格显示上面所选房类的房间信息，如房号，状态信息（如果是住人房，显示客人人数），数据来自 `roomstat`

5. 复杂的客人查询（参考）

客人组合查询

请组合查询条件：

查询

组合查询条件

(抵达日期='2002/10/10' OR 抵达日期='2002/10/11')  
AND 姓名 Like '王--'

( )  
OR AND  
NOT  
清空 查询

姓名	性别	抵达日期	拟走日期	国籍	房号
王新日	女	002/10/10	002/10/14	中国	1222
王立群	男	002/10/11	002/10/23	中国	1225

说明：

- 1) 第一行第一项为表 guests 的各域汉字名称，只选；
  - 2) 第一行第二项为 Like、=、>=、<、>、!=，只选；
  - 3) 第一行第三项为对应第一项名称的值，输入；
  - 4) 确定钮：将第一行内容组合写入查询条件框；
  - 5) 取消钮：将第一行初始化；
  - 6) 组合查询条件框：查询条件由第一行写入；组合条件可编辑，在光标设定处，点击右侧所需连接符；
  - 7) 清空钮：清空组合查询条件框；
- 查询钮：将查询条件转化为 SQL 语句，即将汉字域转换成对应的表的域名，程序查询 guests 表，得到结果写入下面表格中；

## 6. 简单客人查询

姓名	性别	抵达日期	拟走日期	国籍	房号
王新日	女	002/10/10	002/10/14	中国	1222
王立群	男	002/10/11	002/10/23	中国	1225
王花蕊	女	002/10/13	002/10/23	中国	1001

说明：

- 1) 查询条件中四项可输入一致四项；
- 2) 当点击查找时用输入的内容构成查询的 SQL 语句，程序查询 guests 表，得到结果写入下面表格中；

#### (六) 最低要求

1. 完成客人登记
2. 完成客人简单查询；
3. 完成房间简单查询；
4. 数据表必须有 guests（可做简化）、room、roomstat 三表；

#### 设计要求：

在实验报告中要给出具体的操作要求和过程，并针对各种情况做出具体的分析和讨论。

#### 提交文档：

- 1、设计文档(提交书面文档)

实验总结：编程时间、调试情况（主要错误，解决办法）、实验过程中遇到的主要问题、如何克服、对你的程序进行评价、实验的收获。

## 2、程序源代码（含实验结果）

（提交电子文档（文件夹名为“学号-姓名-项目 x”））

### 三、实验考核

- 1、采取实验室演示提问与提交文档相结合的方式进行考核。
- 2、该实验的各实验项目要求每位同学独立完成。
- 3、成绩为：优、良、中、及格、不及格。

说明：1、书面提交的设计文档面格式见实验报告格式。

- 2、电子文档形式提交的程序源码等最后形成一个文件夹（名称为“学号-姓名”）。

# 实验报告格式

详见《数据库系统原理实践报告》。