# ODBC 开发手册

# 目录

# 一， 前言

本手册作为安谷云数据库的接口开发系列手册之一，主要介绍安谷云数据库 ODBC 接口的主要功能和其简单的外围应用，旨在帮助使用安谷云数据库服务的应用开发人员快速开发有关于数据库交互的接口编程。为使用 ODBC 访问安谷云数据库提供技术支持。

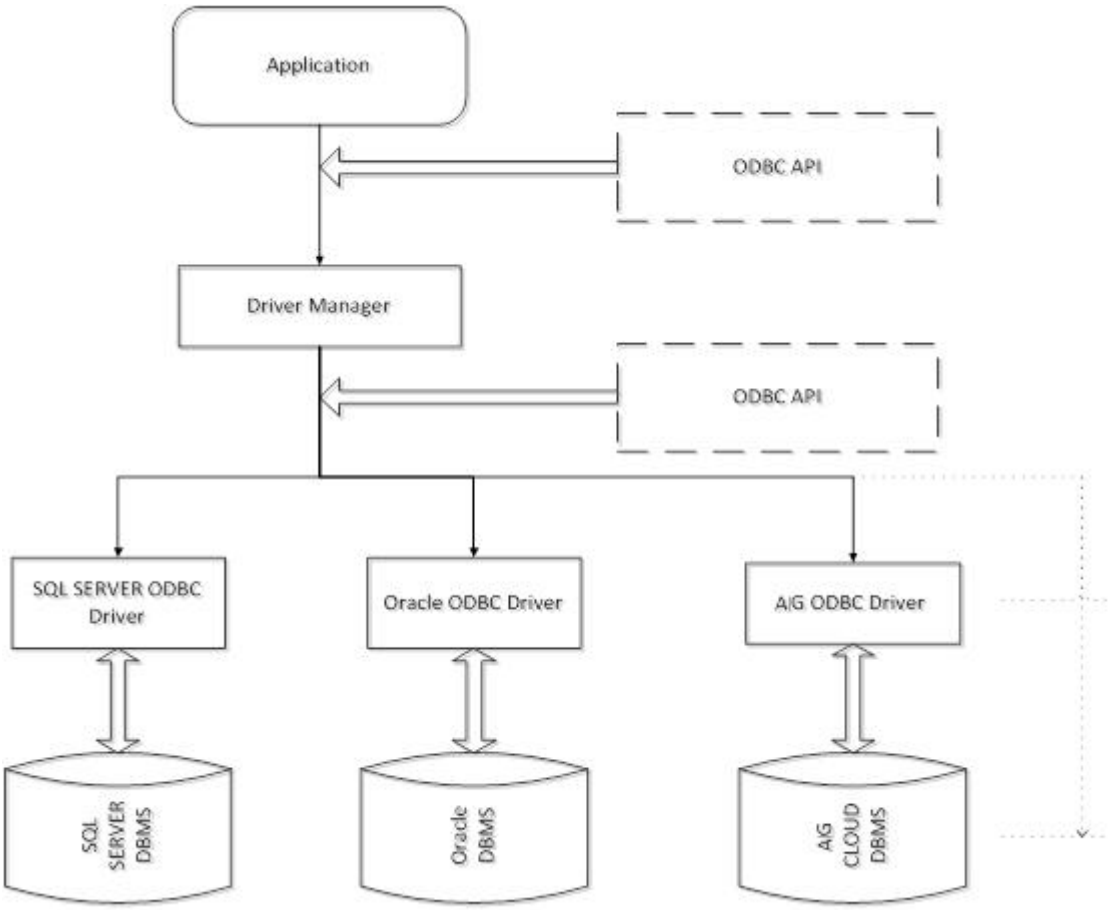
安谷云数据库系统是大型通用数据库系统，可对外提供的编程服务接口主要有：ODBC，JDBC，OLEDB 等。

其中 JDBC 主要为 JAVA 的应用提供编程接口。OLEDB 是为 COM 类使用提供接口。而 ODBC (Open Database Connectivity，开放数据库互连)是基于微软公司开放服务结构之上提供的一套编程接口。可以对 WIN 平台下的 C，C++ ，C#，VB 等编程语言编写的应用程序提供安谷云数据库的访问接口。本手册重点介绍 ANGOO CLOUD ODBC 的使用。

# 二，ODBC 概述

在 WINDOWS 平台下的 ODBC 使用流程框架如下图所示：

图 2-1




ODBC（Open Database Connectivity,开放式数据库连接），建立了一组规范，提供了一组对数据库访问的标准 API（Application Programming Interface,应用程序编程接口）,是 WOSA（Windows Open Services Architecture，微软公司开放服务结构）中有关数据库的一个组成部分。它提供了一种应用程序与数据库之间的交流的通道与标准。

DSN (Data Source Name，数据源简称)是指任一种可以通过 ODBC 连接的数据库管理系统，它包括要访问的数据库和数据库的运行平台。数据源名掩盖了数据库服务器或数据库文件间的差别，通过定义多个数据源，每个数据源指向一个服务器名，就可在应用程序中实现同时访问多个 DBMS 的目的。

数据源是驱动程序与 DBMS 连接的桥梁，数据源不是 DBMS，而是用于表达一个 ODBC 驱动程序和 DBMS 特殊连接的命名。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，可以将数据源名看成是与一个具体数据库建立的连接。

数据源分为以下三类：

用户数据源 用户创建的数据源，称为“用户数据源”。此时只有创建者才能使用，并且只能在所定义的机器上运行。任何用户都不能使用其他用户创建的用户数据源。

系统数据源 所有用户和在 Windows NT 下以服务方式运行的应用程序均可使用系统数据源。

文件数据源 文件数据源是 ODBC 3.0 以上版本增加的一种数据源，可用于企业用户，ODBC 驱动程序也安装在用户的计算机上。

ODBC 提供了在不同数据库环境中为客户机/服务器（简称 C / S）结构的客户机访问异构数据库的接口，也就是在由异构数据库服务器构成的 C / S 结构中，要实现对不同数据库进行的数据访问，就需要一个能连接不同的客户机平台到不同服务器的桥梁，ODBC 就是起这种连接作用的桥梁。ODBC 提供了一个开放的、标准的能访问从 PC 机、小型机到大型机数据库数据的接口。使用 ODBC 标准接口的应用程序，开发者可以不必深入了解要访问的数据库系统，比如其支持的操作和数据类型等信息，而只需掌握通用的 ODBC API 编程方法即可。使用 ODBC 的另一个好处是当作为数据库源的数据库服务器上的数据库管理系统升级或转换到不同的数据库管理系统时，客户机端应用程序不需作任何改变，因此利用 ODBC 开发的数据库应用程序具有很好的移植性。

应用程序要通过 ODBC 接口访问一个数据库，首先必须用 ODBC 管理器注册一个数据源，管理器根据数据源提供的数据库位置、数据库类型及 ODBC 驱动程序等信息，建立起 ODBC 与具体数据库的联系。

这样，只要应用程序将数据源名提供给 ODBC，ODBC 就能建立起与相应数据库的连接。

在 ODBC 中，ODBC API 不能直接访问数据库，必须通过驱动程序管理器与数据库交换信息。

驱动程序管理器负责将应用程序对 ODBC API 的调用传递给正确的驱动程序，而驱动程序在执行完相应的操作后，将结果通过驱动程序管理器返回给应用程序。

根据以上论述，要通过 ODBC 访问 ANGOO CLOUD 数据库系统， 则需要用 ODBC 管理器（DriverManager）注册一个数据源(DSN)，然后应用程序通过指定数据源 DSN 来访问 ANGOO CLOUD 数据库系统。

ODBC 总体结构有四个组件：

1，应用程序。 负责执行处理并调用 ODBC 函数，以提交 SQL 语句并检索结果。

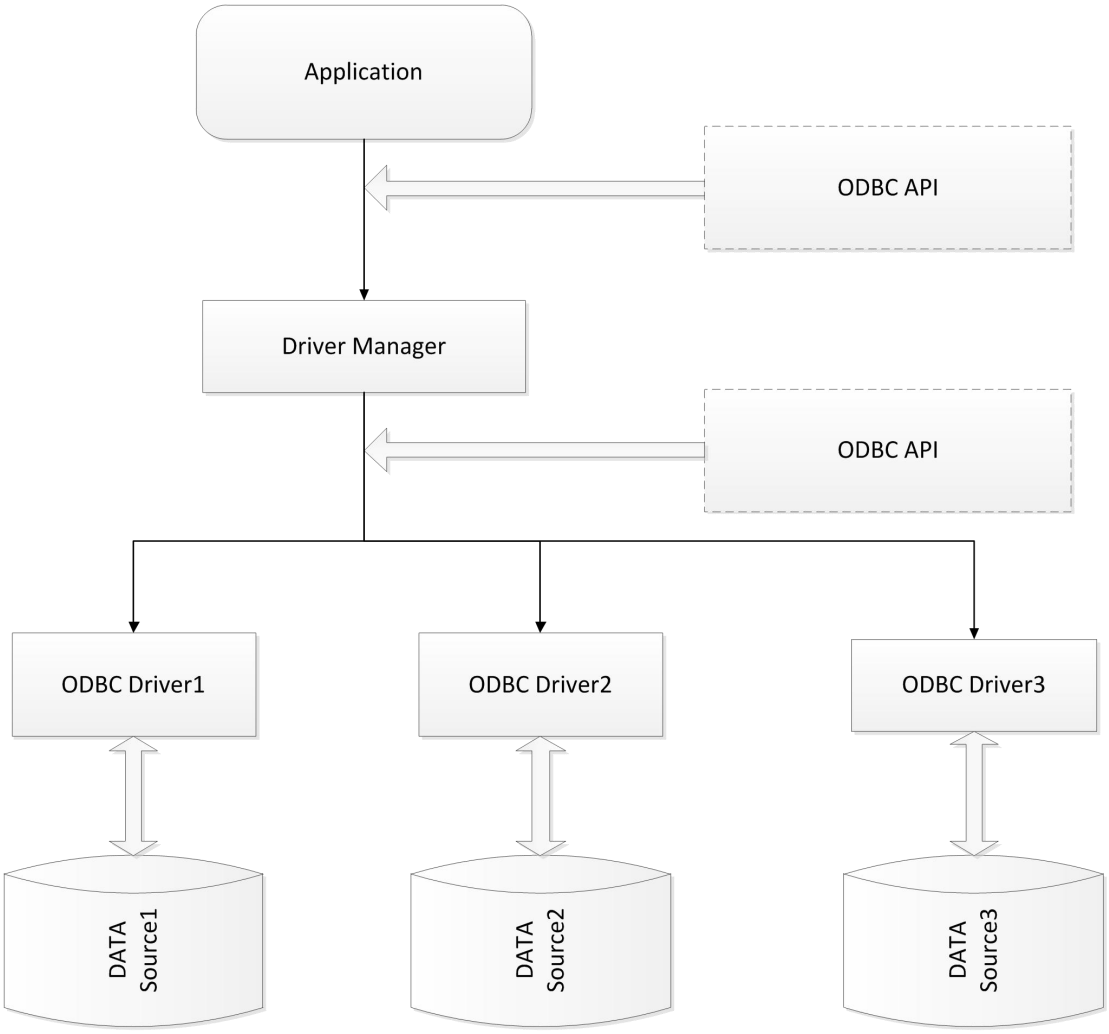
2，Driver Manager。负责根据应用程序加载并卸载驱动程序。处理 ODBC 函数调用，或把他们传送到驱动程序。

3，驱动程序。负责处理 ODBC 函数调用，提交 SQL 请求到一个指定的数据源，并把结果返回到应用程序。如果有必要，驱动程序修改一个应用程序请求，以使请求与相关的 DBMS 支持的语法一致。

4，数据源。包括用户要访问的数据及其相关的操作系统，DBMS 及用于访问 DBMS 的网络平台（如果有的话）。

下图显示这四个组件之间的关系。

图 2-2


Application
ODBC API
Driver Manager
ODBC API
ODBC Driver1
ODBC Driver2
ODBC Driver3
DATA Source1
DATA Source2
DATA Source3

# 三，安谷云 ODBC 的安装及数据源配置

## 3.1 安装 ODBC

一般情况下，在 WIN 平台下 安装安谷云客户端系统，会集成安装 ANGOO CLOUD ODBC。

单独安装的情况： ODBC 目录下 会有 AngooOdbc.dll 和 AngooODBC_Install.exe 文件

执行 AngooODBC_Install.exe 可以将 ANGOO CLOUD ODBC 注册到 ODBC 管理器 的驱动程序列表中

此过程需注意使用的 ODBC 是 WIN32 的 还是 X64 的，win32 的 ODBC 为 32 位的应用程序提供服务 X64 的 ODBC 为 64 位的应用程序

特别的在 64 位操作系统下 ：

X64 的 ODBC 管理器 位于系统安装盘的 C:\Windows\System32 目录下的 odbcad32.exe

32 位的 ODBC 管理器 位于系统安装盘的 C:\Windows\SysWOW64 目录下的 odbcad32.exe

注意区分他们，以免出现注册了 ODBC 数据源，但是在应用程序访问时出现数据源不存在的问题。

## 3.2 ODBC 数据源的注册

根据应用程序是 32 位还是 64 位，选择相应的 ODBC 管理器注册数据源

1 运行 odbcad32.exe 或者 选择 控制面板->管理工具->ODBC 数据源

2 选择 “系统 DSN” ->“添加” -> 选择数据源 “Angoo SQL 7.01” ->完成

3 配置 连接参数

4 测试连接完成

**数据源名称(DSN)**：我们通过数据源名称来调用数据源信息，达到连接数据库与数据库交互的目的。必填 DSN 名称是应用程序调用的依据

**数据源描述(Describle)**：描述此数据源的备注信息，通过此属性可以填写一些简短的信息供 ODBC 数据源使用者查阅。选填 描述此数据源的版本信息

**服务器网址(url)**：本机可填写（127.0.0.1 或者 localhost），其他机器（异机）可填写 IP 地址。因为数据库服务器是通过网络与应用程序交互的，所以这个网址也是必填的重要属性之一。

**侦听端口(Port)** ：这个需与连接的服务器节点的侦听端口一致。通常服务器使用 5138 为默认端口，有时使用 5139 或者其他端口（通过查阅 ANGOO.INI 文件可以看到使用的端口号），客户端或者应用程序通过此端口与服务器交互，数据库服务器端有专用的线程侦听用过此端口的信息。在服务器为默认侦听端口时，默认填入 5138。

**数据库名称(Databse 、DB)**：此属性决定了应用程序调用该数据源时连接的数据库名。因为安谷 DBMS 是多库并存的数据库管理系统，所以为了保证连接的

准确定，该属性必填。

**启用安全连接方式（Use SSL）**：此属性决定了应用程序与数据库服务器之间的数据传输是否加密。（众所周知，网络本身是不安全的，谁都可以截获网络中的数据，如果我们通过把数据加密再通过网络传输，他得到的就是密文，没有解密的密钥就得不到有用的明文，从而达到保护数据的目的。）选择此选项时，网络传输数据加密，安全性更高，但在频繁建立连接的模型中不推荐使用，较不启用安全连接方式时，建立连接速度较慢。

**用户名（User 、UID）**：此属性描述了连接数据源时的用户名信息。必填

**口令（password 、PWD）**：与用户名相匹配的口令。必填

**字符集(char_set) ：**连接的字符集属性。取值可以为：GBK，UTF8，GB2312等

**时区(timezone)**：连接的时区属性。取值可以为：GMT+01:00~ GMT+11:00，GMT-01:00~ GMT-11:00，中国大陆默认取值为（GMT+08:00）。

**自动提交(auto_commit)**： 取值可以为 true（自动提交），false （非自动提交）两种。当取值为自动提交时，应用程序每发一个 SQL 语句到服务器，驱动程序都会自动提交它；而取值为非自动提交时，则需要应用程序自己发送 commit 到服务器去，之前的 SQL 语句才会被提交。当然有特殊的情况，参见“常见问题以及解决方案” ，可根据用户需求选择是否自动提交。

**默认隔离级别(iso_level):**可选填：读已提交(READ COMMITED)，可重复读（REPEATABLE READ），串行化（SERIALIZABLE），读未提交（READ UNCOMMITTED）。一般选择读已提交。

一个可用的参数配置如下图所示：


图 3-1

# 四，安谷云 ODBC 数据类型介绍

## 4.1 数据库端数据类型

表 4-1 列举了安谷云数据库数据类型以及描述信息

表 4-1

| 安谷云数据类型 | 数据长度（所占字节） | 描述信息 |
|---|---|---|
| Char(n) | n 字节，最大不超过 64K | 固定串长度为 n 的字符串 |
| Varchar(n) | n 字节，最大不超过 64K | 最大字符串长度为 n 的可变长度字符串 |
| Binary(n) | n 字节，最大不超过 64K | 固定长度为 n 的二进制数据 |
| Image | 最大不超过 2G | 影像图片数据类型，可变长度的二进制数据 |
| Tinyint | 1 字节 | 精度为 3，标度为 0 的有符号精确数字，范围-128-- 127 |
| Smallint | 2 字节 | 精度为 5，标度为 0 的有符号精确数字 范围 -32768 -- 32767 |
| Integer | 4 字节 | 精度为 10，标度为 0 的有符号精确数字 C int 的取值 |
| Bigint | 8 字节 | 精度为 19，标度为 0 的有符号精确数字值 int64 的取值范围 |
| Float | 4 字节 | 浮点数类型 |
| Double | 8 字节 | 双精度浮点数字 |
| Bool | 1 字节 | 布尔型，取值 true，false 或者 ‘T’，‘F’ |
| Numeric(p,s) | 20 字节 | 精度为 p，标度为 s 的有符号精确数字值 |
| Time | 4 字节 | 时间数据类型，时分秒字段 |
| Datetime | 8 字节 | 时间戳数据类型，年月日时分秒字段 |
| Date | 4 字节 | 日期数据类型，年月日字段 |
| Time with time zone | 6 字节 | 时间数据类型，时分秒，时区字段 |
| Datetime with time zone | 10 字节 | 时间戳数据类型，年月日时分秒,时区字段 |

| | | |
|---|---|---|
| Blob | 最大不超过 4G | 二进制大对象类型字段 |
| Clob | 最大不超过 4G | 字符大对象的存储字段 |
| Interval year | 4 字节 | 年间隔，即两个日期之间的年数字 |
| Interval month | 4 字节 | 月间隔，即两个日期之间的月数字 |
| Interval day | 4 字节 | 日间隔，即两个日期之间的日数字 |
| Interval hour | 4 字节 | 时间隔，即为两个日期/时间之间的时数字 |
| Interval minute | TYPE_INTERVAL_MI | 分间隔，即为两个日期/时间之间的分数字 |
| Interval second | 8 字节 | 秒间隔，即为两个日期/时间之间的秒数字 |
| Interval day to hour | 4 字节 | 日时间隔，即为两个日期/时间之间的日时数字 |
| Interval day to minute | 4 字节 | 日时分间隔，即为两个日期/时间之间的日时分数字 |
| Interval day to second | 8 字节 | 日时分秒间隔，即为两个日期/时间之间的日时分秒数字 |
| Interval hour to minute | TYPE_INTERVAL_H2M | 时分间隔，即为两个日期/时间之间的时分数字 |
| Interval hour to second | 8 字节 | 时分秒间隔，即为两个日期/时间之间的时分秒数字 |
| Interval minute to second | 8 字节 | 分秒间隔，即为两个日期/时间之间的分秒间隔 |
| Interval year to month | 4 字节 | 年月间隔，即两个日期之间的年月数字 |
| Point | 16 字节 | 元素点类型 |
| Box | 32 字节 | 边框类型 |
| Polyline | 变长 最大 32K | 折线类型 |
| Polygon | 变长 最大 32K | 多边形类型 |
| Geometry | 变长 最大 32K | 通用空间类型 |

## 4.2 数据库端数据类型与 ODBC 标准的数据类型的映射

表 4-2 列举了安谷云数据库数据类型以及 ODBC 的通用 SQL 数据类型的映射关系

表 4-2

| **安谷云数据类型** | **ODBC SQL 公共数据类型** | **数据类型描述** |
|---|---|---|

| | | |
|---|---|---|
| Char(n) | SQL_CHAR | 固定串长度为 n 的字符串 |
| Varchar(n) | SQL_CHAR | 最大字符串长度为 n 的可变长度字符串 |
| Binary(n) | SQL_BINARY | 固定长度为 n 的二进制数据 |
| Image | SQL_LONGVARBINARY | 影像图片数据类型，可变长度的二进制数据 |
| Tinyint | SQL_TINYINT | 精度为 3，标度为 0 的有符号精确数字，范围 -128-- 127 |
| Smallint | SQL_SMALLINT | 精度为 5，标度为 0 的有符号精确数字 范围 -32768 -- 32767 |
| Integer | SQL_INTEGER | 精度为 10，标度为 0 的有符号精确数字 C int 的取值 |
| Bigint | SQL_BIGINT | 精度为 19，标度为 0 的有符号精确数字值 int64 的取值范围 |
| Float | SQL_FLOAT | 浮点数类型 |
| Double | SQL_DOUBLE | 双精度浮点数字 |
| Bool | SQL_TINYINT、SQLCHAR | 布尔型，取值 true，false 或者 ‘T’，‘F’ |
| Numeric(p,s) | SQL_NUMERIC | 精度为 p，标度为 s 的有符号精确数字值 |
| Time | SQL_TIME | 时间数据类型，时分秒字段 |
| Datetime | SQL_DATETIME | 时间戳数据类型，年月日时分秒字段 |
| Date | SQL_DATE | 日期数据类型，年月日字段 |
| Time with time zone | XG_C_CHAR | 时间数据类型，时分秒，时区字段 |
| Datetime with time zone | XG_C_CHAR | 时间戳数据类型，年月日时分秒,时区字段 |
| Blob | SQL_LONGVARBINARY | 二进制大对象类型字段 |
| Clob | SQL_LONGVARCHAR | 字符大对象的存储字段 |
| Interval year | SQL_INTERVAL_YEAR | 年间隔，即两个日期之间的年数字 |
| Interval month | SQL_INTERVAL_MONTH | 月间隔，即两个日期之间的月数字 |
| Interval day | SQL_INTERVAL_DAY | 日间隔，即两个日期之 |

| | | |
|---|---|---|
| | | 间的日数字 |
| Interval hour | SQL_INTERVAL_HOUR | 时间隔，即为两个日期/时间之间的时数字 |
| Interval minute | SQL_INTERVAL_MINUTE | 分间隔，即为两个日期/时间之间的分数字 |
| Interval second | SQL_INTERVAL_SECOND | 秒间隔，即为两个日期/时间之间的秒数字 |
| Interval day to hour | SQL_INTERVAL_DAY_TO_HOUR | 日时间隔，即为两个日期/时间之间的日时数字 |
| Interval day to minute | SQL_INTERVAL_DAY_TO_MINUTE | 日时分间隔，即为两个日期/时间之间的日时分数字 |
| Interval day to second | SQL_INTERVAL_DAY_TO_SECOND | 日时分秒间隔，即为两个日期/时间之间的日时分秒数字 |
| Interval hour to minute | SQL_INTERVAL_HOUR_TO_MINUTE | 时分间隔，即为两个日期/时间之间的时分数字 |
| Interval hour to second | SQL_INTERVAL_HOUR_TO_SECOND | 时分秒间隔，即为两个日期/时间之间的时分秒数字 |
| Interval minute to second | SQL_INTERVAL_MINUTE_TO_SECOND | 分秒间隔，即为两个日期/时间之间的分秒间隔 |
| Interval year to month | SQL_INTERVAL_YEAR_TO_MONTH | 年月间隔，即两个日期之间的年月数字 |
| Point | SQL_BINARY | 元素点类型 |
| Box | SQL_BINARY | 边框类型 |
| Polyline | SQL_BINARY | 折线类型 |
| Polygon | SQL_BINARY | 多边形类型 |
| Geometry | SQL_BINARY | 通用空间类型 |

## 4.3 ODBC 标准的数据类型与 C 数据类型的绑定

表 4-3 列举了数据库数据类型，ODBC SQL 通用数据类型，以及他们与 C 语言数据类型绑定时的映射关系信息。

表 4-3

| **ANGOO DBMS 数据库的数据类型** | **ODBC SQL 公共数据类型** | **ODBC 绑定的 C 映射的数据类型（有的可以以字符串方式绑定）（C datatype to SQL** | **备注** |
|---|---|---|---|

| | | **datatype mapping）** | |
|---|---|---|---|
| Char(n) | SQL_CHAR | SQL_C_CHAR | |
| Varchar(n) | SQL_CHAR | SQL_C_CHAR | |
| Binary(n) | SQL_BINARY | SQL_C_BINARY | |
| Image | SQL_LONGVARBINARY | SQL_C_BINARY | |
| Tinyint | SQL_TINYINT | SQL_C_TINYINT/SQL_C_CHAR | |
| Smallint | SQL_SMALLINT | SQL_C_SHORT | |
| Integer | SQL_INTEGER | SQL_C_LONG | |
| Bigint | SQL_BIGINT | SQL_C_SBIGINT | |
| Float | SQL_FLOAT | SQL_C_FLOAT | |
| Double | SQL_DOUBLE | SQL_C_DOUBLE | |
| Bool | SQL_C_TINYINT | SQL_C_TINYINT/SQL_C_CHAR | 没有 SQL_BOOL 类型，可用单字节的 TINYINT 和 CHAR 来存放 |
| Numeric(p,s) | SQL_NUMERIC | SQL_C_NUMERIC/SQL_C_CHAR | |
| Time | SQL_TIME/SQL_TYPE_TIME | SQL_C_TIME/SQL_C_TYPE_TIME/SQL_C_CHAR | 根据 ODBC 版本不同可以映射至两种 SQL 编号，与 C 类型映射时，可以映射至 C 的数据结构，也可以映射至 字符串形式存放 |
| Datetime | SQL_DATETIME/SQL_TYPE_TIMESTAMP | SQL_C_TIMESTAMP/SQL_C_TYPE_TIMESTAMP/SQL_C_CHAR | 根据 ODBC 版本不同可以映射至两种 SQL 编号，与 C 类型映射时，可映射至 C 的结构体数据也可映射成字符串 |
| Date | SQL_DATE/SQL_TYPE_DATE | SQL_C_DATE/SQL_C_TYPE_DATE/SQL_C_CHAR | 根据 ODBC 版本不同可以映射至两种 SQL 编号，与 C 类型映射时，可映射至 C 的结构体数据也可映射成字符串 |
| Time with time zone | SQL_CHAR | SQL_C_CHAR | 结果较为特殊，故适合映射成 CHAR 字符串存放 |
| Datetime with time zone | SQL_CHAR | SQL_C_CHAR | 结果较为特殊，故适合映射成 CHAR 字符串存放 |

| | | | |
|---|---|---|---|
| Blob | SQL_LONGVARBINARY | SQL_C_BINARY | 大对象类型，以二进制的方式存放 |
| Clob | SQL_LONGVARCHAR | SQL_C_CHAR | |
| Interval year | SQL_INTERVAL_YEAR | SQL_C_INTERVAL_YEAR/SQL_C_CHAR | |
| Interval month | SQL_INTERVAL_MONTH | SQL_C_INTERVAL_MONTH/SQL_C_CHAR | |
| Interval day | SQL_INTERVAL_DAY | SQL_C_INTERVAL_DAY/SQL_C_CHAR | |
| Interval hour | SQL_INTERVAL_HOUR | SQL_C_INTERVAL_HOUR/SQL_C_CHAR | |
| Interval minute | SQL_INTERVAL_MINUTE | SQL_C_INTERVAL_MINUTE/SQL_C_CHAR | |
| Interval second | SQL_INTERVAL_SECOND | SQL_C_INTERVAL_SECOND/SQL_C_CHAR | |
| Interval day to hour | SQL_INTERVAL_DAY_TO_HOUR | SQL_C_INTERVAL_DAY_TO_HOUR/SQL_C_CHAR | |
| Interval day to minute | SQL_INTERVAL_DAY_TO_MINUTE | SQL_C_INTERVAL_DAY_TO_MINUTE/SQL_C_CHAR | |
| Interval day to second | SQL_INTERVAL_DAY_TO_SECOND | SQL_C_INTERVAL_DAY_TO_SECOND/SQL_C_CHAR | |
| Interval hour to minute | SQL_INTERVAL_HOUR_TO_MINUTE | SQL_C_INTERVAL_HOUR_TO_MINUTE/SQL_C_CHAR | |
| Interval hour to second | SQL_INTERVAL_HOUR_TO_SECOND | SQL_C_INTERVAL_HOUR_TO_SECOND /SQL_C_CHAR | |
| Interval minute to | SQL_INTERVAL_MINU | SQL_C_INTERVAL_MINUTE_TO_S | |

| | | | |
|---|---|---|---|
| second | TE_TO_SECOND | ECOND/SQL_C_CHAR | |
| Interval year to month | SQL_INTERVAL_YEAR_TO_MONTH | SQL_C_INTERVAL_YEAR_TO_MONTH/SQL_C_CHAR | |
| Point | SQL_BINARY | SQL_C_BINARY | |
| Box | SQL_BINARY | SQL_C_BINARY | |
| Polyline | SQL_BINARY | SQL_C_BINARY | |
| Polygon | SQL_BINARY | SQL_C_BINARY | |
| Geometry | SQL_BINARY | SQL_C_BINARY | |

## 4.4 常用的数据类型转换

在数据库的数据信息与 C 语言中的变量绑定时，有时会做一些数据转化使得应用程序更方便的操作数据。他们主要分为两大类：

### 4.4.1 数据库中的数值型数据类型

使用 C 中的 CHAR 类型变量进行绑定，用于输入和输出。
具体的有：
Double ， float, bigint,
Numeric, short , int,tinyint 等
下面是一些此类数据类型的参数按字符串方式绑定的示例：

```
char c1[]="123";
    char c2[]="34";
    char c3[]="777";
    char c4[]="23.42";
    char c5[]="3322.4564";
    char c6[]="121212";
    char c7[]="863977";

                    SQLBindParameter(hstmt, 1,
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_TINYINT,10,0,c1,10,&cbLen1);
                    SQLBindParameter(hstmt, 2,
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_SMALLINT,10,0,c2,10,&cbLen2);
                    SQLBindParameter(hstmt, 3,
```

```
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_INTEGER,11,0,c3,11,&cbLen3);
                                    SQLBindParameter(hstmt,                    4,
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_FLOAT,21,0,c4,21,&cbLen4);
                                    SQLBindParameter(hstmt,                    5,
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_DOUBLE,21,0,c5,21,&cbLen5);
                                    SQLBindParameter(hstmt,                    6,
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_BIGINT,21,0,c6,21,&cbLen6);
                                    SQLBindParameter(hstmt,                    7,
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_NUMERIC,21,0,c7,21,&cbLen7);
```

### 4.4.2 日期型等在 C 语言中有结构体存放的数据类型

与数据库中的数据类型绑定时，可根据用户使用的方便可以选择 以 C 的结构体变量绑定 或者采用某种约定格式的字符串方式绑定

具体的情况有：

Date,time , datetime3

Time with timezone

Datetime with timezone 等

示例：

```
DATE_STRUCT * c1=(DATE_STRUCT *)malloc(sizeof(DATE_STRUCT ));
TIME_STRUCT * c2 =(TIME_STRUCT * )malloc(sizeof(TIME_STRUCT ));
TIMESTAMP_STRUCT* c3= (TIMESTAMP_STRUCT*)malloc(sizeof(TIMESTAMP_STRUCT));
    c1->year =2015;
    c1->month =1;
    c1->day= 21;
    c2->hour =11;
    c2->minute =56;
    c2->second =34;
    c3->year =2015;
    c3->month=1;
    c3->day =23;
    c3->hour =12;
    c3->minute =03;
    c3->second =45;
    char c4[]="09:45:16.0 +08:00";
    char c5[]="1990-02-17 06:35:47 +08:00";
    cbLen1=sizeof(DATE_STRUCT );
    cbLen2=sizeof(TIME_STRUCT );
    cbLen3=sizeof(TIMESTAMP_STRUCT );
    cbLen4 =strlen(c4);
    cbLen5 =strlen(c5);
    SQLBindParameter(hstmt, 1,
SQL_PARAM_INPUT, SQL_C_DATE, SQL_DATE, 10, 0, c1, 10, &cbLen1);
```

```
    SQLBindParameter(hstmt, 2,
SQL_PARAM_INPUT,SQL_C_TIME,SQL_TIME,10,0,c2,10,&cbLen2);
    SQLBindParameter(hstmt, 3,
SQL_PARAM_INPUT,SQL_C_TIMESTAMP,SQL_TIMESTAMP,11,0,c3,11,&cbLen3);
    SQLBindParameter(hstmt, 4,
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_CHAR,21,0,c4,21,&cbLen4);
    SQLBindParameter(hstmt, 5,
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_CHAR,30,0,c5,30,&cbLen5);
```

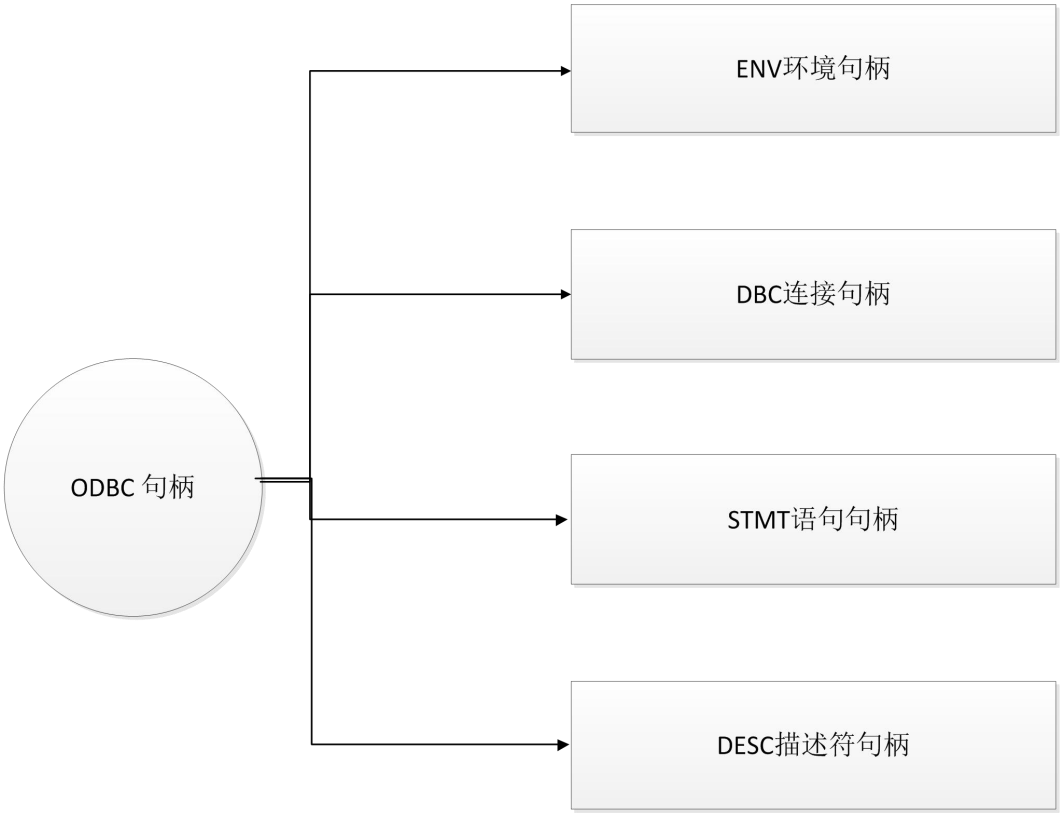
# 五，安谷云 ODBC 建立连接

在应用程序调用 ODBC 时，主要的内容分为分配句柄和通过句柄调用相应的 API 函数。

## 5.1 句柄

句柄：是 Windows ODBC API 封装的指针地址。通过它可以分配下级句柄，以及对相应层的句柄，调用对应的 API 函数。句柄下的数据结构对用户是不透明的。用户只能通过相应的功能 API 函数来得到用户需要的信息和完成 SQL 的执行。
句柄类型有：环境句柄，连接句柄，语句句柄，描述句柄等。
如图 5-1 所示：

图 5-1



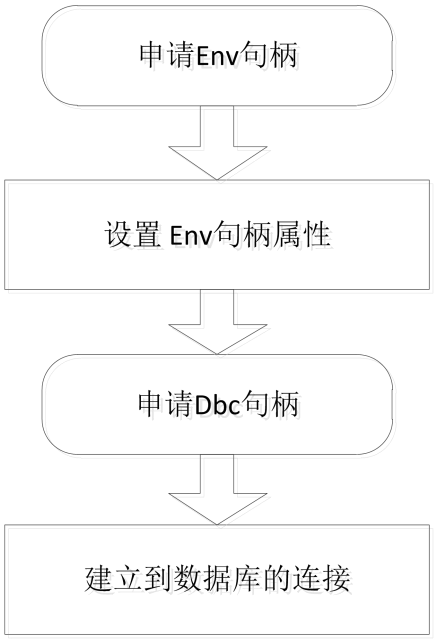

## 5.2 建立连接的过程

step1 申请分配环境句柄
step2 设置环境句柄属性
step3 申请连接句柄
step4 连接指定的用户到服务器

此时连接的部分完结，之后便是申请语句句柄，通过语句句柄绑定 sql 语句执行了。

以下是建立连接的流程图：

图 5-2




此段程序的 C 代码示例如下：

```
sr = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
sr = SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_IS_INTEGER);
    sr = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
sr = SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAuthStr,SQL_NTS);
```

此为连接建立的主要流程,当函数 SQLConnect 的返回值为 SQL_SUCCESS 时表示连接建立成功。

连接的建立除了 SQLConnect 函数之外，还有 SQLDriverConnect 函数以及 SQLBrowseConnec 函数。他们提供了更灵活的连接方式，对于他们，在下一章会有详细的介绍。

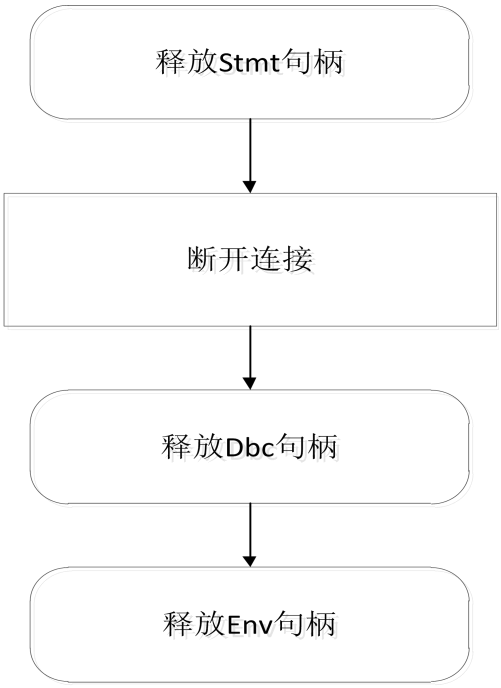
## 5.3 连接的释放

在应用完成对数据库的操作之后，需要完成断开连接释放资源的操作，如果连接不从服务器端断开连接，则会影响其他的新建连接服务。如果客户端的资源不得到释放则会影响应用的后续资源申请，特别是频繁的建连接的操作，如果旧资源得不到释放，会极大的影响后续应用的新建连接。所以，连接的释放对于

应用的长期稳定运行有极重要的作用。
连接的释放主要分为以下步骤：

s1 在有语句句柄的情况下先释放语句句柄。
s2 断开连接。
s3 释放连接句柄。
s4 释放环境句柄。
流程图如图 5-3 所示：

图 5-3




此段程序的 C 代码示例如下：

```
rs=SQLFreeHandle(SQL_HANDLE_STMT,hstmt);
rs=SQLDisconnect(hdbc);
rs=SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
rs=SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

## 5.4 各编程语句的连接串。

C# 连接串 “DSN=ANGOO; UID=SYSDBA; PWD= SYSDBA;DB=SYSTEM ;URL=192.168.0.18”;

“Dsn=TEST;desc=AngooSQL Server ODBC 3.01 Driver DSN;db=SYSTEM;server=192.168.0.88;user=SYSDBA;port=5138;autocommit=TRUE;usessl=TRUE;charset=GBK;timezone=GMT+08:00;isolevel=2”

C++连接串 “DSN= ANGOO; UID= SYSDBA; PWD= SYSDBA;”

VB 连接串 “DSN= ANGOO ; UID= SYSDBA; PWD= SYSDBA;”

备注：主要 ODBC 的参数配置都是在 ODBC 数据源的配置中完成的，所以 ODBC 连接串仅需填写 ODBC 的 DSN 名称、用户名、密码即可。

# 六，安谷云 ODBC 常用 API 介绍

## 6.1 SQLAllocHandle()函数

用途：句柄申请函数。
函数原型：

SQLRETURN SQL_API SQLAllocHandle(SQLSMALLINT HandleType,
SQLHANDLE InputHandle, SQLHANDLE *OutputHandle);

参数：

HandleType：句柄类型

InputHandle：输入句柄类型，通常是要申请的句柄类型的父句柄。

OutputHandle：申请的目标句柄（输出句柄）。申请句柄成功后，此参数返回成功的目标句柄地址。

返回值：

通常在成功时返回 SQL_SUCCESS 。

有状况时 返回 SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, 或 SQL_ERROR。

可申请的指定句柄类型有：

SQL_HANDLE_ENV 环境句柄

SQL_HANDLE_DBC 连接句柄

SQL_HANDLE_STMT 语句句柄

SQL_HANDLE_DESC 描述符句柄

除了环境句柄不需要父句柄外，其他句柄的申请都需要有父句柄的存在。

### 6.1.1 环境句柄的申请，是使用 ODBC 的最初的环境准备。

示例：

```
sr = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
```

附带的 SQLSetEnvAttr() 环境属性设置：

```
sr = SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_IS_INTEGER);
```

设置 ODBC 版本, 以便确认之后 ODBC 的 API 函数版本调用情况。

因不同的 ODBC 版本，某些功能接口需要调用不同的 API 函数

### 6.1.2 连接句柄的申请：

连接句柄的申请是在环境句柄作为父句柄的情况下进行的,例如

```
sr = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
```

此外，因为应用可能会多个线程并发的访问数据库服务器 Server 端。故一个环境句柄可以申请多个连接句柄

他们彼此独立的工作，可并行访问数据库。提高访问效率。

### 6.1.3 语句句柄的申请

语句句柄的申请，需要连接句柄已完成数据库连接之后进行。他的父句柄是连接句柄。

在句柄句柄执行某些 API 函数的过程中，语句句柄和连接之间存在一对一的绑定关系，这种绑定时隐式的。

```
sr = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);
```

## 6.2 连接函数：

### 6.2.1 SQLConnect()函数：

作用：连接客户端和服务器端，生成会话

函数原型：

```
SQLRETURN    SQL_API SQLConnect(SQLHDBC ConnectionHandle,
            SQLCHAR *ServerName, SQLSMALLINT NameLength1,
            SQLCHAR *UserName, SQLSMALLINT NameLength2,
            SQLCHAR *Authentication, SQLSMALLINT NameLength3);
```

参数：

ConnectionHandle： 连接句柄

ServerName： DSN 名 即为 ODBC 配置的 DSN 的名字。在第三章有配置 DSN 的介绍。

NameLength1: DSN 名的长度 以字节为单位，可以选填默认值 SQL_NTS ，它可以 根据 DSN 名长度自动匹配长度

UserName: 用户名，字符串格式。连接数据库的身份权限标识

NameLength2: 用户名长度 以字节为单位，可以选填默认值 SQL_NTS ，它可以 根据用户名长度自动匹配长度

Authentication：用户口令 字符串格式 ，连接数据库的用户匹配的口令。

NameLength3：用户口令长度 以字节为单位，可以选填默认值 SQL_NTS ，它可以 根据用户口令长度自动匹配长度

返回值：

通常在成功时返回 SQL_SUCCESS 。

此函数为连接的核心函数，是连接 ODBC 数据源的最基本的方法，在所有的一致性级别上都支持。参数如上面的介绍，使用 SQLConnect 函数的连接数据源的示例代码如下：

```
ret= SQLConnect(hdbc,(SQLCHAR*)"ANGOO",SQL_NTS,
    (SQLCHAR*)"SYSDBA",SQL_NTS,(SQLCHAR*)"SYSDBA",SQL_NTS);
```

```
if(SUCCESS!=ret)
{
    /*连接数据源失败！*/
    ...进行相应的错误处理...
    exit(0);
}
```

### 6.2.2 SQLDriverConnect()函数：

用途：根据用户的需求，设置连接属性，新建连接。

函数原型：

```
SQLRETURN SQL_API SQLDriverConnect(
  SQLHDBC              hdbc,
  SQLHWND               hwnd,
  SQLCHAR              *szConnStrIn,
  SQLSMALLINT           cbConnStrIn,
  SQLCHAR              *szConnStrOut,
  SQLSMALLINT           cbConnStrOutMax,
  SQLSMALLINT          *pcbConnStrOut,
  SQLUSMALLINT          fDriverCompletion);
```

参数：

hdbc: 连接句柄

hwnd: 窗口句柄

szConnStrIn:连接属性串。用户可以选择需要的 DSN，并在此设置更改需要的连接参数属性

cbConnStrIn：连接串属性长度，以字节为单位 可以选填默认值 SQL_NTS

szConnStrOut：连接输出信息串，本字符串包含完全连接的信息。注意申请相应的内存

cbConnStrOutMax:输出连接串的最大长度。

pcbConnStrOut：连接输出信息串长度 ，以字节为单位 可以选填默认值 SQL_NTS

fDriverCompletion：指示完成标识，指示是否驱动需要提供更多的连接参数信息

返回值：

通常在成功时返回 SQL_SUCCESS 。

备注：此函数比 SQLConnect 更为灵活 。它支持以下几种连接：要求更多连接参数的数据源，对话框提示用户输入所有的连接信息以及没有在系统信息表中定义的数据源。

SQLDriverConnect 提供以下的连接方法：

（1） 用一个连接字符串建立一个连接，这个字符串包括建立连接的所有数据，如 DSN，一个或多个用户 ID 及其口令，以及其他的数据库所需要的连接信息。

（2） 用一个并不完整的连接字符串来建立连接，使得 ODBC 驱动程序管理器来提示用户输入所需要的连接信息。

（3） 用一个没有在系统信息表中登记的数据源建立连接，驱动程序自动提示用户输入连接信息。
（4） 用一个连接字符串建立连接，这个字符串在 DSN 配置文件中是确定的。
SQLDriverConnect 函数 fDriverCompletion
示例：

```
SQLCHAR szConnStrIn[256] = "DSN=ANGOO;DRIVER=Angoo ODBC DRIVER;
DATABASE=SYSTEM;UID=SYSDBA;PWD=SYSDBA; PORT=5138";
SQLCHAR szConnStrOut[256];
SQLSMALLINT cbConnStrOut;
ret = SQLDriverConnect(hdbc, hwnd, szConnStrIn, SQL_NTS, szConnStrOut, 256,
&cbConnStrOut, SQL_DRIVER_PROMT);
if(SUCCESS!=ret)
{
/* 连接数据源失败! */
…进行相应的错误处理…
exit(0);
}
```

### 6.2.3 SQLBrowseConnect() 函数

用途：根据用户的需求，设置连接属性，新建连接。
函数原型：

```
SQLRETURN SQL_API SQLBrowseConnect(
    SQLHDBC              hdbc,
    SQLCHAR             *szConnStrIn,
    SQLSMALLINT          cbConnStrIn,
    SQLCHAR             *szConnStrOut,
    SQLSMALLINT          cbConnStrOutMax,
    SQLSMALLINT         *pcbConnStrOut);
```

参数：
hdbc：连接句柄
szConnStrIn：连接属性串
cbConnStrIn：连接串属性长度，以字节为单位 可以选填默认值 SQL_NTS
szConnStrOut:连接输出信息串，本字符串包含完全连接的信息。注意申请相应的内存
cbConnStrOutMax: 输出连接串的最大长度。
pcbConnStrOut:连接输出信息串长度 ，以字节为单位 可以选填默认值 SQL_NTS
返回值：
通常在成功时返回 SQL_SUCCESS 。
SQLBrowseConnec 函数与 SQLDriverConnect 函数相似， 但是调用 SQLBrowseConnect 函数时，程序在运行时可以再形成一个连接字符串，使用这个函数可以用一个交互的方式来决定连接到数据源时所需要的一些信息。
使用 SQLBrowseConnect 函数连接数据源的示例如下：

```
SQLCHAR szConnStrIn[256] = "";
SQLCHAR szConnStrOut[256];
SQLSMALLINT cbConnStrOut;
strcpy(constr, "DRIVER=Angoo ODBC DRIVER");
ret = SQLBrowseConnect(hdbc,szConnStrIn,SQL_NTS,szConnStrOut, 256,
&cbConnStrOut);
if (ret != SQL_NEED_DATA)
{
/* 连接数据源失败! */
…进行相应的错误处理…
exit(0);
}
strcpy(constr, "SERVER=127.0.0.1; PORT=5138");
ret=SQLBrowseConnect(hdbc,szConnStrIn, SQL_NTS, szConnStrOut, 256,
&cbConnStrOut);
if (ret != SQL_NEED_DATA)
{
/* 连接数据源失败! */
…进行相应的错误处理…
exit(0);
}
strcpy(constr, "UID=SYSDBA;PWD=SYSDBA;");
sret=SQLBrowseConnect(hdbc,szConnStrIn, SQL_NTS, szConnStrOut, 256,
&cbConnStrOut);
if (ret != SQL_SUCCESS)
{
/* 连接数据源失败! */
…进行相应的错误处理…
exit(0);
}
/*连接成功*/
```

## 6.3 SQLPrepare()函数

作用：准备一个 SQL 语句，生成执行执行计划，但不执行。
函数原型：

```
SQLRETURN  SQL_API SQLPrepare
(
  SQLHSTMT StatementHandle,
  __in_ecount(TextLength) SQLCHAR* StatementText,
  SQLINTEGER TextLength
);
```

参数：

StatementHandle： 语句句柄
StatementText： sql 语句字串
TextLength：语句长度，以字节为单位
返回值：
在成功时返回 SQL_SUCCESS 。
备注：
此为 SQL 语句准备执行函数，他在服务器端申请游标，规划执行路径等；注意 Create,Truncate 语句等 DDL 语句不可以使用 SQLPrepare 的方式来执行，严格的说，DDL 语句不能像 DML 一样享受 Prepare 的方式带来的好处。

## 6.4 SQLBindParameter()函数

作用：绑定执行 SQL 时需要的参数。
函数原型：

```
SQLRETURN SQL_API SQLBindParameter(
    SQLHSTMT              hstmt,
    SQLUSMALLINT          ipar,
    SQLSMALLINT           fParamType,
    SQLSMALLINT           fCType,
    SQLSMALLINT           fSqlType,
    SQLULEN               cbColDef,
    SQLSMALLINT           ibScale,
    SQLPOINTER            rgbValue,
    SQLLEN                cbValueMax,
    SQLLEN                *pcbValue);
```

参数：
hstmt：语句句柄
ipar：参数序列号
fParamType:参数 io 类型
fCType：参数 C 类型
fSqlType：参数对应的 sqltype 类型
cbColDef:参数精度（string 类型时参数的长度大小）
ibScale：数值 numeric 时参数的标度
rgbValue：参数值指针（引用地址）
cbValueMax：参数字节宽度
pcbValue：参数的实际长度，指针值
返回值：
在成功时返回 SQL_SUCCESS 。

### 6.4.2 不同版本的参数绑定函数

作用：绑定执行 SQL 时需要的参数。在 ODBC Driver Manager 1.0 的版本时使

用 (ODBCVER = 0x0100) ，后续的版本均使用 SQLBindParameter 函数来完成相同的操作。该函数可完全被 SQLBindParameter 替换，但考虑到应用历史原因，我们保留了此功能接口 API。

函数原型:

```
SQLRETURN   SQL_API SQLBindParam(SQLHSTMT StatementHandle,
            SQLUSMALLINT ParameterNumber,
            SQLSMALLINT ValueType,
            SQLSMALLINT ParameterType,
            SQLULEN LengthPrecision,
            SQLSMALLINT ParameterScale,
            SQLPOINTER ParameterValue,
            SQLLEN *StrLen_or_Ind);
```

参数：

StatementHandle：语句句柄参数

ParameterNumber：参数序列号（从 1 开始）

ValueType：参数 C 类型

ParameterType：参数 IO 类型

LengthPrecision：参数精度

ParameterScale：参数标度

ParameterValue：参数值指针

StrLen_or_Ind：参数实际长度指针

## 6.5 SQLBindCol()函数

作用：将 C 数据类型变量，申请内存空间按类型绑定到输出列。

函数原型：

```
SQLRETURN   SQL_API SQLBindCol(
            SQLHSTMT StatementHandle,
            SQLUSMALLINT ColumnNumber,
            SQLSMALLINT TargetType,
            SQLPOINTER TargetValue,
            SQLLEN BufferLength,
            SQLLEN *StrLen_or_Ind);
```

参数：

StatementHandle：语句句柄参数

ColumnNumber：绑定输出列序号（从 1 开始）

TargetType：绑定列类型

TargetValue:绑定输出列指针

BufferLength：列宽度，string 类型的长度，以字节为单位

StrLen_or_Ind：实际输出列大小。

备注：输出列值的内存空间由用户自己申请，如果申请的空间不够，可能导致内存访问越界。参数的大小与超界问题 有 API 函数 SQLColAttribute 可以解决。

### 6.5.2 SQLColAttribute()函数

作用：本函数用于在输出列绑定之前，获知所需绑定的输出列的大小，绑定的列名等等用户需要的列属性，以便于用户进行绑定时的参数设置。

函数原型：

```
SQLRETURN SQLColAttribute (
        SQLHSTMT        StatementHandle,
        SQLUSMALLINT        ColumnNumber,
        SQLUSMALLINT        FieldIdentifier,
        SQLPOINTER        CharacterAttributePtr,
        SQLSMALLINT        BufferLength,
        SQLSMALLINT *        StringLengthPtr,
        SQLPOINTER        NumericAttributePtr);
```

参数：

StatementHandle：语句句柄参数

ColumnNumber：输出列编号参数（从 1 开始）

FieldIdentifier：用户需要得到的列的属性标识。

CharacterAttributePtr：该参数在序列号为 ColumnNumber 的输出列的 FieldIdentifier 属性所指定的输出值为 string 时，指定一块 buffer 以供输出使用。当 FieldIdentifier 所指定的属性值不是 string 类型时，该参数无效。

BufferLength：当 FieldIdentifier 指定的属性值由 CharacterAttributePtr 提供输出时，BufferLength 指定输出字串的 buffer 长度大小，单位为字节。

StringLengthPtr：指向 buff 的返回值的字节大小数。

NumericAttributePtr：该参数在序列号为 ColumnNumber 的输出列的 FieldIdentifier 属性所指定的输出值为数值型的描述值时，由该属性来完成属性值的输出功能。

## 6.6 SQLExecute()函数

作用：此函数执行 SQLPrepare 准备的语句

函数原型：

```
SQLRETURN    SQL_API SQLExecute(SQLHSTMT StatementHandle);
```

参数：

StatementHandle：语句句柄参数

备注：在 stmt 句柄绑定 prepare 的情况下，该函数相较于 SQLExecDirect 函数有更好的执行效率。且在有参数进行绑定时，可变换绑定的参数，多次执行 SQLExecute 函数，在执行过程中，只传递参数值，提高执行效率。

## 6.7 SQLExecuteDirect()函数

作用：直接执行一个 SQL 语句到服务器端，不需要 prepare 的准备过程。

函数原型：

```
SQLRETURN    SQL_API SQLExecDirect
(
   SQLHSTMT StatementHandle,
   __in_ecount_opt(TextLength) SQLCHAR* StatementText,
   SQLINTEGER TextLength
 );
```

参数：

StatementHandle：语句句柄参数

StatementText： sql 语句，字符串类型

TextLength：sql 字符串的长度 以字节为单位 可以选填默认值 SQL_NTS

备注：该函数的执行时如果携带参数，需要将参数放在此函数调用的前面。在某语句只执行一次时，推荐此函数，在同一语句多次执行时，推荐 SQLExecute，因其具有更高的执行效率。

## 6.8 SQLFetch()函数

作用：对于 SQLExecute，SQLExecDirect，以及其他带返回结果集的功能 API 函数执行后，产生的结果集进行获取行操作。

函数原型：

```
SQLRETURN    SQL_API SQLFetch(SQLHSTMT StatementHandle);
```

参数：

StatementHandle：语句句柄参数。

备注：对于 Select 语句产生的结果集进行取数操作。每执行一次，结果集的当前记录向下移一条，SQLBindCol 函数绑定的输出列值更新到新值。直到移动至结果集的末尾,此时返回值为 SQL_NO_DATA。注意在每次执行 SQLFetch 函数后，SQLBindCol 绑定的 C 变量 buffer 需要重置，以免上一条的记录值残留在输出值 buffer 里面影响输出结果的获取。

例如：当一个结果集的某列 col2 为字符串列输出，当前值为”XIAOLIU”,实际返回长度为 7，执行 SQLFetch 取下一个行时，col2 的输出值为”LUCY”,实际返回长度为 4，如果该 buffer 在 SQLFetch 执行之前未重置，则此时 buffer 的内部的实际内容为”LUCYLIU”,实际返回长度指针值是 4 是正确的，但 buffer 内未产生正常的截断’\0’。引用 buffer 时，其值为”LUCYLIU”,在进行某些操作时会发生错误。

对于一个结果集行序列，该函数的取值是单向往后取值的，不提供取了某条记录之后又取之前的记录的功能。

## 6.9 常用功能 API 接口集

### 6.9.1 SQLTables()函数

作用：根据输入参数条件，检索相关的信息形成结果集，供用户查阅。
函数原型：

```
SQLRETURN SQLTables(
        SQLHSTMT        StatementHandle,
        SQLCHAR *       CatalogName,
        SQLSMALLINT         NameLength1,
        SQLCHAR *       SchemaName,
        SQLSMALLINT         NameLength2,
        SQLCHAR *       TableName,
        SQLSMALLINT         NameLength3,
        SQLCHAR *       TableType,
        SQLSMALLINT         NameLength4);
```

参数：
StatementHandle：语句句柄参数
CatalogName：数据库名，char* 类型的字符串。当值为 NULL 时表示所有
NameLength1：参数*CatalogName 字符串的长度，以字节为单位。
SchemaName：模式名，char*类型的字符串。当值为 NULL 时表示所有
NameLength2：参数*SchemaName 字符串的长度，以字节为单位。
TableName：表名，char*类型的字符串。当值为 NULL 时表示所有
NameLength3：参数*TableName 字符串的长度，以字节为单位。
TableType：要匹配的表类型。比如：”table”,”view”等
NameLength4：参数*TableType 字符串的长度，以字节为单位。
备注：当 CatalogName 和 SchemaName 非空，而 TableName 为 NULL 时，该函数可查询出某个指定的库的指定模式下的所有的表的信息，以某种指定的格式组成行最终形成结果集返回。结果集的获取通过 SQLFetch()函数进行。

### 6.9.2 SQLColumns()

作用：
函数原型：

```
SQLRETURN SQLColumns(
        SQLHSTMT        StatementHandle,
        SQLCHAR *       CatalogName,
        SQLSMALLINT         NameLength1,
        SQLCHAR *       SchemaName,
        SQLSMALLINT         NameLength2,
        SQLCHAR *       TableName,
        SQLSMALLINT         NameLength3,
```

SQLCHAR * ColumnName,
SQLSMALLINT NameLength4);

参数：

StatementHandle：语句句柄参数

CatalogName：数据库名，char* 类型的字符串。当值为 NULL 时表示所有

NameLength1：参数*CatalogName 字符串的长度，以字节为单位。

SchemaName：模式名，char*类型的字符串。当值为 NULL 时表示所有

NameLength2：参数*SchemaName 字符串的长度，以字节为单位。

TableName：表名，char*类型的字符串。当值为 NULL 时表示所有

NameLength3：参数*TableName 字符串的长度，以字节为单位。

ColumnName：列名，char*类型的字符串。

NameLength4：参数* ColumnName 字符串的长度，以字节为单位。

备注：当指定了库名，模式名，表名，且 ColumnName 值为 NULL 时，函数罗列该表的所有列的信息，当然也可以将 ColumnName 指定为需要检索的某列的列名，则只返回数据库中，该表该列的信息。类似于 SQLTables 函数，该函数的查询结果也是以某种方式形成记录，多条记录形成结果集的方式返回给用户。

### 6.9.3 SQLPrimaryKeys()

作用：获取数据库中某个表的主键信息

函数原型：

```
SQLRETURN SQLPrimaryKeys(
        SQLHSTMT        StatementHandle,
        SQLCHAR *       CatalogName,
        SQLSMALLINT         NameLength1,
        SQLCHAR *       SchemaName,
        SQLSMALLINT         NameLength2,
        SQLCHAR *       TableName,
        SQLSMALLINT         NameLength3);
```

参数：

StatementHandle：语句句柄参数

CatalogName：数据库名，char* 类型的字符串。

NameLength1：参数*CatalogName 字符串的长度，以字节为单位。

SchemaName：模式名，char*类型的字符串。

NameLength2：参数*SchemaName 字符串的长度，以字节为单位。

TableName：表名，char*类型的字符串。

NameLength3：参数*TableName 字符串的长度，以字节为单位。

备注：当指明了数据库名，模式名，和表名后，可获取该表的主键信息。

### 6.9.4 SQLForeignKeys()

作用：获取某表与外键相关的信息。

函数原型：

```
SQLRETURN SQLForeignKeys(
        SQLHSTMT        StatementHandle,
        SQLCHAR *       PKCatalogName,
        SQLSMALLINT         NameLength1,
        SQLCHAR *       PKSchemaName,
        SQLSMALLINT         NameLength2,
        SQLCHAR *       PKTableName,
        SQLSMALLINT         NameLength3,
        SQLCHAR *       FKCatalogName,
        SQLSMALLINT         NameLength4,
        SQLCHAR *       FKSchemaName,
        SQLSMALLINT         NameLength5,
        SQLCHAR *       FKTableName,
        SQLSMALLINT         NameLength6);
```

参数：

StatementHandle：语句句柄参数
PKCatalogName：外键所引用的主键列所在表的数据库名。
NameLength1：参数*PKCatalogName 字符串的长度，以字节为单位。
PKSchemaName：外键所引用的主键列所在表的模式名。
NameLength2 ：参数* PKSchemaName 字符串的长度，以字节为单位。
PKTableName：外键所引用的主键列所在表的表名。
NameLength3：参数* PKTableName 字符串的长度，以字节为单位。
FKCatalogName：外键列所在表的数据库名。
NameLength4：参数* FKCatalogName 字符串的长度，以字节为单位。
FKSchemaName：外键列所在表的模式名。
NameLength5：参数* FKSchemaName 字符串的长度，以字节为单位。
FKTableName：外键列所在表的表名。
NameLength6：参数* FKTableName 字符串的长度，以字节为单位。
备注：除了外键自身，还可以获知外键所引用的主键列的信息。从输入参数来看，我们可以选择外键所引用表的信息来检索外键信息，也可以从外键所在表自身的信息来检索外键的信息。参数限定越多，目标定位越精准。如一个外键表，他有来自不同的引用表的外键信息，此时同时定位外键表和他引用的主表的信息，则可以避免另一个外键信息在此被检索出来。

## 6.9.5 SQLProcedures()

作用：用户可以指定数据库名，模式名，存储过程名来查询存储过程的创建信息。
函数原型：

```
SQLRETURN SQLProcedures(
        SQLHSTMT        StatementHandle,
        SQLCHAR *       CatalogName,
        SQLSMALLINT         NameLength1,
        SQLCHAR *       SchemaName,
```

SQLSMALLINT NameLength2,
SQLCHAR * ProcName,
SQLSMALLINT NameLength3);

参数：

StatementHandle：语句句柄参数

CatalogName：数据库名，char* 类型的字符串。

NameLength1：参数*CatalogName 字符串的长度，以字节为单位。

SchemaName：模式名，char*类型的字符串。

NameLength2：参数*SchemaName 字符串的长度，以字节为单位。

ProcName：存储过程名，char*类型的字符串。

NameLength3：参数* ProcName 字符串的长度，以字节为单位。

备注：用户可以指定数据库名，模式名，存储过程名来查询存储过程的创建信息。主要是过程体的内容定义。

## 6.9.6 SQLStatistics()

作用：可检索表的索引相关信息

函数原型：

```
SQLRETURN SQLStatistics(
    SQLHSTMT StatementHandle,
    SQLCHAR * CatalogName,
    SQLSMALLINT NameLength1,
    SQLCHAR * SchemaName,
    SQLSMALLINT NameLength2,
    SQLCHAR * TableName,
    SQLSMALLINT NameLength3,
    SQLUSMALLINT Unique,
    SQLUSMALLINT Reserved);
```

参数：

StatementHandle：语句句柄参数

CatalogName：数据库名，char* 类型的字符串。

NameLength1：参数*CatalogName 字符串的长度，以字节为单位。

SchemaName：模式名，char*类型的字符串。

NameLength2：参数*SchemaName 字符串的长度，以字节为单位。

TableName：表名，char*类型的字符串。

NameLength3：参数*TableName 字符串的长度，以字节为单位。

Unique：是否指定唯一值索引。可填值为：SQL_INDEX_UNIQUE 或 SQL_INDEX_ALL

Reserved：标识索引信息是即时更新还是延迟更新，当为延迟更新时，驱动不保证某些信息是当前即时的。

## 6.10 大对象操作函数

### 6.10.1 SQLPutData()函数

作用：该函数主要用于 CLOB 和 BLOB 的大对象数据录入操作。
函数原型：

```
SQLRETURN SQLPutData(
        SQLHSTMT        StatementHandle,
        SQLPOINTER        DataPtr,
        SQLINTEGER        StrLen_or_Ind);
```

参数：
StatementHandle：语句句柄参数。
DataPtr：一个指向实际数据值的指针，该值与要求输入的参数列值相映射。
StrLen_or_Ind：参数*DataPtr 数据值的长度，以字节为单位。
备注：该函数的主要用途在于以参数的形式录入大对象的值，包括 char 类型的 CLOB 的大对象和 binary 类型的 BLOB 的大对象。因有的大对象较大需要占用较大的空间故可以反复多次调用此函数进行录入操作。该函数的使用在第 8 章的大对象的插入会有使用示例。

### 6.10.2 SQLGetData()函数

作用：用于返回一个结果行的单独列的数据，当要返回的数据较大时，可分部分多次调用返回。
函数原型：

```
SQLRETURN SQLGetData(
        SQLHSTMT        StatementHandle,
        SQLUSMALLINT        ColumnNumber,
        SQLSMALLINT        TargetType,
        SQLPOINTER        TargetValuePtr,
        SQLINTEGER        BufferLength,
        SQLINTEGER *        StrLen_or_IndPtr);
```

参数：
StatementHandle：语句句柄参数。
ColumnNumber：输出列号参数。
TargetType：C 数据类型标识。
TargetValuePtr：指向一块数据返回的 buffer。
BufferLength：数据返回的 buffer 的长度大小，以字节为单位。当返回数据的大小比这个长度大时，将发生截断。
StrLen_or_IndPtr：输出参数，该参数将返回数据的实际填充到 buffer 的长度。
备注：当实际返回数据长度等于 buffer 的大小时，可能数据未完结，需要继续取数，当实际返回长度小于 buffer 的大小，或等于 0 时，可函数根据返回值判断数据是否已经获取完毕。该函数的使用在第 8 章的大对象的查询会有使用示例。

## 6.11 SQLMoreResults()函数

作用：判断是否还有多的结果集，如果有，提供初始化操作。
函数原型：
SQLRETURN SQLMoreResults(
SQLHSTMT StatementHandle);
参数：
StatementHandle：语句句柄参数。
备注：该函数主要对于一次发送多个 SELECT，UPDATE，DELETE，INSERT 等语句的情况进行判别。如果发送了多个语句一起发送，则返回的结果集也会有多个与之对应，本函数可检测到是否还有其他的结果集缓存于网络上未收取完毕。

# 七，安谷云 ODBC 应用编程的基本步骤

应用程序使用 ODBC 访问数据源，可以按照以下几个基本步骤进行：

s1.调用函数 SQLAllocHandle 申请环境、连接句柄，调用函数 SQLSetEnvAttr 设置环境句柄属性，调用函数 SQLSetConnectAttr 设置连接句柄属性，调用连接函数 SQLConnect、SQLDriverConnect 或 SQLBrowseConnect 连接相关的数据源。
s2．调用函数 SQLAllocHandle 申请语句句柄，通过语句句柄应用程序可以执行 SQL 语句进行相关的 SQL 操作。调用函数 SQLPrepare 对 SQL 语句和操作进行准备，调用 SQLDescribeCol、SQLDescribeParam 等函数取得相关的描述信息，依据描述信息调用 SQLBindCol、SQLBindParam 等函数绑定相关的列和参数，然后调用 SQLExecute 执行 SQL 语句，实现相关的 SQL 操作。应用程序也可以调用函数 SQLExecDirect 直接执行 SQL 语句进行相关的 SQL 操作。
s3．应用程序可以通过调用 ODBC 编目函数 SQLTables、SQLColumns、SQLStatistics 等取得数据源相关的字典信息。

或者调用 SQLExecuteDirect、SQLPrepare+SQLExecute 执行用户需要的 SQL 语句
s4．如果连接属性自动提交选项设置为手动提交状态，应用程序可以调用函数 SQLEndTran 来提交或回滚事务，进行相关的事务处理。
s5．调用函数 SQLFreeHandle 来释放申请的语句句柄。
s6．调用函数 SQLDisconnect 来断开应用程序与数据源之间的连接。
s7．调用函数 SQLFreeHandle 来释放申请的连接、环境句

在 VB，C#等环境下，对 ODBC 的使用提供的封装，用户不用执行全部的步骤，而是交由封装的程序去完成。具体的见 8 章 ODBC 的使用示例。

# 八，安谷云 ODBC 的常用简单应用编程示例

以下示例，均在用户在 Windows 平台配置好 ODBC DSN 的前提下进行。

## 8.1 c,c++ 部分

### 8.1.1 建立连接和释放连接的示例：

代码如下：

```
char szDSN[] ="ANGOO" ;
  char szUID[] ="SYSDBA";
  char szAuthStr[] = "SYSDBA";
  SQLHENV    henv = NULL;
  SQLHDBC    hdbc = NULL;
  SQLHSTMT hstmt = NULL;
  SQLRETURN rs = 0;
  rs = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
  rs                                                                    =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_I
S_INTEGER);

  rs = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
  rs                                                                    =
SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAu
thStr,SQL_NTS);
  if(sr != SQL_SUCCESS )
  {
        char pstr_errstate[128];
        SQLINTEGER ierror;
        char pstr_messtext[256];
        short itextlen;

SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messtext,2
56,&itextlen);
        int tt = 0;
  }else
  {
    printf("connect OK！ \n");
  }
  rs=SQLDisconnect(hdbc);
  rs=SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
```

```
 rs=SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

### 8.1.2 Select 查询 SQL 语句的执行的示例:

代码如下：

```
char szDSN[] ="ANGOO" ;
    char szUID[] ="SYSDBA";
    char szAuthStr[] = "SYSDBA";
    SQLHENV    henv = NULL;
    SQLHDBC    hdbc = NULL;
    SQLHSTMT hstmt = NULL;
    SQLRETURN rs = 0;
    rs = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
    rs =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_I
S_INTEGER);

    rs = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
    rs =
SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAu
thStr,SQL_NTS);
    if(rs != SQL_SUCCESS )
    {
        char pstr_errstate[128];
        SQLINTEGER ierror;
        char pstr_messtext[256];
        short itextlen;

    SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messt
ext,256,&itextlen);
        int tt = 0;
    }else
    {
        printf("connect OK！ \n");
    }
    rs = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);
    char sq2[]="select * from t8_odbc";
    FILE* fp;
    fp=fopen("D:\\tout_t8_odbc.txt","wb+");
    rs =SQLExecDirect(hstmt,(UCHAR*)sq2,strlen(sq2));

    if(SQL_SUCCESS!=rs)
    {
        char pstr_errstate[64];
```

```
        char pstr_messtext[256];
        SQLINTEGER ierror;
        short itextlen;
        memset(pstr_errstate,0x0, 64);
        memset(pstr_messtext,0x0, 256);

    SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messt
ext,256,&itextlen);
        int a=9;
    }
    SQLLEN cbLen1,cbLen2,cbLen3,cbLen4,cbLen5;
    char ci1[30]={0};
    char ci2[30]={0};
    char ci3[30]={0};
    char ci4[30]={0};
    char ci5[30]={0};
    SQLBindCol(hstmt, 1, SQL_C_CHAR, ci1, 30, &cbLen1);
    SQLBindCol(hstmt, 2, SQL_C_CHAR, ci2, 30, &cbLen2);
    SQLBindCol(hstmt, 3, SQL_C_CHAR, ci3, 30, &cbLen3);
    SQLBindCol(hstmt, 4, SQL_C_CHAR, ci4, 30, &cbLen4);
    SQLBindCol(hstmt, 5, SQL_C_CHAR, ci5, 30, &cbLen5);

    char buff[1024]={0};
    while(rs!=-1 &&rs !=100)
    {
        rs=SQLFetch(hstmt);
        memset(buff,0x0,1024);
        if(rs != 100)
        {
            sprintf(buff,"%s, %s, %s, %s, %s    \n" ,ci1,ci2,ci3,ci4,ci5);
            memset(ci1 ,0x0,30);
            memset(ci2 ,0x0,30);
            memset(ci3 ,0x0,30);
            memset(ci4 ,0x0,30);
            memset(ci5 ,0x0,30);
        }else {
            break;
        }
      fwrite(buff,1,strlen(buff),fp);
    }
    fclose(fp);
    rs=SQLFreeHandle(SQL_HANDLE_STMT,hstmt);

    rs=SQLDisconnect(hdbc);
```

```
    rs=SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
    rs=SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

### 8.1.3 Insert 插入的 SQL 语句的执行示例：

代码如下：

```
 char szDSN[] ="ANGOO" ;
 char szUID[] ="SYSDBA";
 char szAuthStr[] = "SYSDBA";
 SQLHENV    henv = NULL;
 SQLHDBC    hdbc = NULL;
 SQLHSTMT hstmt = NULL;
 SQLRETURN rs = 0;
 rs = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
 rs                                                                    =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_I
S_INTEGER);

 rs = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
 rs                                                                    =
SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAu
thStr,SQL_NTS);
 if(rs != SQL_SUCCESS )
 {
      char pstr_errstate[128];
      SQLINTEGER ierror;
      char pstr_messtext[256];
      short itextlen;

SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messtext,2
56,&itextlen);
      int tt = 0;
 }else
 {
  printf("connect OK！ \n");
 }
 rs = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);
 char sql[]= "insert into t8_odbc(c1,c2,c3,c4,c5) values(?,?,?,?,?)";
 rs =SQLPrepare(hstmt,(UCHAR*)sql,strlen(sql));    // SQLPrepare 准备要执行的 SQL
语句。
    if(rs==-1)
    {
```

```
            char pstr_errstate[64];
            char pstr_messtext[256];
            SQLINTEGER ierror;
            short itextlen;
            memset(pstr_errstate,0x0, 64);
            memset(pstr_messtext,0x0, 256);

        SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messt
ext,256,&itextlen);
            int a =8;
        }
    SQLLEN cbLen1,cbLen2,cbLen3,cbLen4,cbLen5;
    DATE_STRUCT * c1=(DATE_STRUCT *)malloc(sizeof(DATE_STRUCT ));
        TIME_STRUCT * c2 =(TIME_STRUCT * )malloc(sizeof(TIME_STRUCT ));
        TIMESTAMP_STRUCT                    *                    c3=
(TIMESTAMP_STRUCT*)malloc(sizeof(TIMESTAMP_STRUCT));
        c1->year =2015;
        c1->month =1;
        c1->day= 21;
        c2->hour =11;
        c2->minute =56;
        c2->second =34;
        c3->year =2015;
        c3->month=1;
        c3->day =23;
        c3->hour =12;
        c3->minute =03;
        c3->second =45;
        char c4[]="09:45:16.0 +08:00";
        char c5[]="1990-02-17 06:35:47 +08:00";
        cbLen1=sizeof(DATE_STRUCT );
        cbLen2=sizeof(TIME_STRUCT );
        cbLen3=sizeof(TIMESTAMP_STRUCT );
        cbLen4 =strlen(c4);
        cbLen5 =strlen(c5);
        SQLBindParameter(hstmt,                                    1,
SQL_PARAM_INPUT,SQL_C_DATE,SQL_DATE,10,0,c1,10,&cbLen1);
        SQLBindParameter(hstmt,                                    2,
SQL_PARAM_INPUT,SQL_C_TIME,SQL_TIME,10,0,c2,10,&cbLen2);
        SQLBindParameter(hstmt,                                    3,
SQL_PARAM_INPUT,SQL_C_TIMESTAMP,SQL_TIMESTAMP,11,0,c3,11,&cbLen3);
        SQLBindParameter(hstmt,                                    4,
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_CHAR,21,0,c4,21,&cbLen4);
        SQLBindParameter(hstmt,                                    5,
```

```
SQL_PARAM_INPUT,SQL_C_CHAR,SQL_CHAR,30,0,c5,30,&cbLen5);

    rs =SQLExecute(hstmt);
    if(SQL_SUCCESS!= rs)
    {
        char pstr_errstate[64];
        char pstr_messtext[256];
        SQLINTEGER ierror;
        short itextlen;
        memset(pstr_errstate,0x0, 64);
        memset(pstr_messtext,0x0, 256);
    SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messt
ext,256,&itextlen);
        int a=9;
    }

 rs=SQLFreeHandle(SQL_HANDLE_STMT,hstmt);

 rs=SQLDisconnect(hdbc);
 rs=SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
 rs=SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

## 8.1.4 Update 更改记录的 SQL 语句的执行示例：

代码如下：

```
char szDSN[] ="ANGOO" ;
 char szUID[] ="SYSDBA";
 char szAuthStr[] = "SYSDBA";
 SQLHENV    henv = NULL;
 SQLHDBC    hdbc = NULL;
 SQLHSTMT hstmt = NULL;
 SQLRETURN rs = 0;
 rs = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
 rs                                                                    =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_I
S_INTEGER);

 rs = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
 rs                                                                    =
SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAu
thStr,SQL_NTS);
 if(rs != SQL_SUCCESS )
 {
```

```
        char pstr_errstate[128];
        SQLINTEGER ierror;
        char pstr_messtext[256];
        short itextlen;

SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messtext,2
56,&itextlen);
        int tt = 0;
  }
  rs = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);
  char sq2[]="update    t2_odbc set name ='guofeng' where id =34 ";
  rs =SQLExecDirect(hstmt,(UCHAR*)sq2,strlen(sq2));
  if(SQL_SUCCESS!=rs)
     {
          char pstr_errstate[64];
          char pstr_messtext[256];
          SQLINTEGER ierror;
          short itextlen;
          memset(pstr_errstate,0x0, 64);
          memset(pstr_messtext,0x0, 256);

     SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messt
ext,256,&itextlen);
          int a=9;
     }
  rs=SQLFreeHandle(SQL_HANDLE_STMT,hstmt);

  rs=SQLDisconnect(hdbc);
  rs=SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
  rs=SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

## 8.1.5 Delete 删除 SQL 语句的执行示例：

代码如下：

```
char szDSN[] ="ANGOO" ;
  char szUID[] ="SYSDBA";
  char szAuthStr[] = "SYSDBA";
  SQLHENV    henv = NULL;
  SQLHDBC    hdbc = NULL;
  SQLHSTMT hstmt = NULL;
  SQLRETURN rs = 0;
  rs = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
  rs                                                                    =
```

```
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_I
S_INTEGER);

  rs = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
  rs                                                                    =
SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAu
thStr,SQL_NTS);
  if(rs != SQL_SUCCESS )
  {
        char pstr_errstate[128];
        SQLINTEGER ierror;
        char pstr_messtext[256];
        short itextlen;

SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messtext,2
56,&itextlen);
        int tt = 0;
  }
  rs = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);
  char sq3[]="delete from    t2_odbc      where id =? ";
    rs =SQLPrepare(hstmt,(UCHAR*)sq3,strlen(sq3));
      SQLLEN cbLen1=4;
      int id =34;
      SQLBindParameter(hstmt,                                                1,
SQL_PARAM_INPUT,SQL_C_LONG,SQL_INTEGER,4,0,&id,4,&cbLen1);

  rs =SQLExecute(hstmt);
  if(SQL_SUCCESS!=rs)
      {
            char pstr_errstate[64];
            char pstr_messtext[256];
            SQLINTEGER ierror;
            short itextlen;
            memset(pstr_errstate,0x0, 64);
            memset(pstr_messtext,0x0, 256);

      SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messt
ext,256,&itextlen);
            int a=9;
      }
  rs=SQLFreeHandle(SQL_HANDLE_STMT,hstmt);

  rs=SQLDisconnect(hdbc);
  rs=SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
```

```
 rs=SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

## 8.1.6 CLOB 大对象的插入及查询示例：

代码如下：

```
char szDSN[] ="ANGOO" ;
 char szUID[] ="SYSDBA";
 char szAuthStr[] = "SYSDBA";
 SQLHENV    henv = NULL;
 SQLHDBC    hdbc = NULL;
 SQLHSTMT hstmt = NULL;
 SQLRETURN rs = 0;
 rs = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
 rs =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_I
S_INTEGER);

 rs = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
 rs =
SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAu
thStr,SQL_NTS);
 if(rs != SQL_SUCCESS )
 {
       char pstr_errstate[128];
       SQLINTEGER ierror;
       char pstr_messtext[256];
       short itextlen;

SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messtext,2
56,&itextlen);
       int tt = 0;
 }else
 {
   printf("connect OK！ \n");
 }
 rs = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);

 int      h_file      =      open("d:\\cl59.txt",O_BINARY|O_CREAT|O_RDWR,_S_IREAD
|_S_IWRITE);   //输入地址

     int cbRead = 0;
     char Data[4096];
     SQLPOINTER    pToken=NULL;
```

```
    SQLLEN cbLen,cbPartID,PartID =6;// 输入 ID

 char sql[]= "insert into t_BIG (id,big) values(?,?)";

 rs =SQLPrepare(hstmt,(UCHAR*)sql,strlen(sql));   // SQLPrepare 准备要执行的 SQL
语句。

 if(rs==-1)
 {
          char pstr_errstate[64];
          char pstr_messtext[256];
          SQLINTEGER ierror;
          short itextlen;
          memset(pstr_errstate,0x0, 64);
          memset(pstr_messtext,0x0, 256);

    SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messt
ext,256,&itextlen);
          int a =8;
 }
  rs = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
                        SQL_INTEGER, 0, 0, &PartID, 4, &cbPartID);
  rs     =     SQLBindParameter(hstmt,     2,     SQL_PARAM_INPUT,SQL_C_CHAR,
SQL_LONGVARCHAR,
                                    123456789, 0, (SQLPOINTER)2, 0, &cbLen);
  cbPartID = 4;
    cbLen = SQL_LEN_DATA_AT_EXEC(0);

    rs = SQLExecute(hstmt);

    while (rs == SQL_NEED_DATA)
    {
        rs = SQLParamData(hstmt, &pToken);

        if (rs == SQL_NEED_DATA)
       {
          do{
               cbRead = _read(h_file,Data,4096);
               if( cbRead <= 0 )break;
               SQLPutData(hstmt, Data, cbRead);
          }while(1);
       }
    }
```

```
    close(h_file);

  rs=SQLFreeHandle(SQL_HANDLE_STMT,hstmt);

  hstmt=NULL;
  rs=SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);
  int
h_file_out=open("D:\\CLOBdemo.txt",O_BINARY|O_CREAT|O_RDWR,_S_IREAD
|_S_IWRITE);
  char qrystr2[] ="select big from t_big where id=6";
    rs = SQLExecDirect(hstmt,(UCHAR*)qrystr2,SQL_NTS);
    rs=SQLFetch(hstmt);
    SQLLEN retlen;
    char pstraddr[102400];
    int i=0;

    while(SQLGetData(hstmt,1,SQL_C_CHAR,pstraddr,102400,&retlen)          !=
SQL_NO_DATA )
    {
        if(retlen>=102400)
            write(h_file_out,pstraddr,102400);
        else
             write(h_file_out,pstraddr,retlen);
    }
    close(h_file_out);

 rs=SQLFreeHandle(SQL_HANDLE_STMT,hstmt);

 rs=SQLDisconnect(hdbc);
 rs=SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
 rs=SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

## 8.1.7 BLOB 大对象的插入及查询示例：

代码如下：

```
char szDSN[] ="ANGOO" ;
char szUID[] ="SYSDBA";
char szAuthStr[] = "SYSDBA";
SQLHENV    henv = NULL;
SQLHDBC    hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN sr = 0;
sr = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);   //申请一个环
```

境句柄
```
sr = SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_IS_INTEGER); // 设置环境句柄的 ODBC 版本

sr = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc); // 申请一个连接句柄

sr = SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAuthStr,SQL_NTS); //

if(sr != SQL_SUCCESS )
{
	char pstr_errstate[128];
	SQLINTEGER ierror;
	char pstr_messtext[256];
	short itextlen;
	SQLError(NULL,NULL,hstmt,(UCHAR*)pstr_errstate,&ierror,(UCHAR*)pstr_messtext,256,&itextlen);
	int tt = 0;
}

sr = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);

const char	add[50]= "D:\\chibifu.jpg" ;//准备测试文件

int h_file = open(add,O_BINARY|O_CREAT|O_RDWR,_S_IREAD |_S_IWRITE);	// 输入地址

int cbRead = 0;
char Data[4096];

SQLRETURN retcode;
SQLPOINTER	pToken=NULL;

SQLLEN cbLen,cbPartID,PartID =2;// 输入 ID

cbPartID = 4;
cbLen = SQL_LEN_DATA_AT_EXEC(0);

char qrystr1[] = "insert into	t6_blob_2(id,big) values(?,?)";

sr = SQLPrepare(hstmt,(UCHAR*)qrystr1, SQL_NTS);// 准备执行的 SQL 语句
```

```
sr = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,SQL_INTEGER, 0, 0, &PartID, 4, &cbPartID);   //绑定参数

sr = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT,SQL_C_BINARY, SQL_LONGVARBINARY,123456789, 0, (SQLPOINTER)2, 0, &cbLen);



sr = SQLExecute(hstmt);   // 执行 SQL 语句

while (sr == SQL_NEED_DATA)
{
    sr = SQLParamData(hstmt, &pToken);

    if (sr == SQL_NEED_DATA)
    {
        do{
            cbRead = _read(h_file,Data,4096);
            if( cbRead <= 0 )break;
            SQLPutData(hstmt, Data, cbRead);
        }while(1);
    }
}
close(h_file);

SQLFreeHandle(SQL_HANDLE_STMT,hstmt);//释放连接句柄
hstmt=NULL;
sr = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);

int h_file_out = open("D:\\output_pic.jpg",O_BINARY|O_CREAT|O_RDWR,_S_IREAD|_S_IWRITE);
char qrystr2[] ="select big from t6_blob_2";
sr = SQLExecDirect(hstmt,(UCHAR*)qrystr2,SQL_NTS);
SQLFetch(hstmt);
SQLLEN retlen;
char pstraddr[102400];
int i=0;
while(SQLGetData(hstmt,1,SQL_C_BINARY,pstraddr,102400,&retlen)                    != SQL_NO_DATA )
{
    if(retlen>=102400)
        write(h_file_out,pstraddr,102400);
    else
        write(h_file_out,pstraddr,retlen);
```

```
}
close(h_file_out);

SQLFreeHandle(SQL_HANDLE_STMT,hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
SQLFreeHandle(SQL_HANDLE_ENV,henv); //  释放环境句柄
```

## 8.1.8 SQLPrimaryKeys 主键信息查询示例：

代码如下：

```
//primarykeys

  struct PK_struct{
            char * tab_cat;
            char* PkSchema;
            char* PkTable;
            char* PkColumnName;
            int seq;
            char* Pkname;
      } ;

char szDSN[] ="ANGOO";
char szUID[] ="SYSDBA";
char szAuthStr[] ="SYSDBA";
SQLHENV     henv = NULL;
SQLHDBC     hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN sr = 0;
sr = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
sr                                                                                    =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_I
S_INTEGER);

      sr = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);

      sr                                                                              =
SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAu
thStr,SQL_NTS);

      sr = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);

      char * pStrSchemaName="SYSDBA";
      char * tabname="TT1";
```

```
// SQLPrimary key
sr=SQLPrimaryKeys(hstmt,
      (UCHAR*)NULL,0,
      (UCHAR*)pStrSchemaName,SQL_NTS,      //SQL_ALL_SCHEMAS
      (UCHAR*)tabname,SQL_NTS);

SQLLEN      iLen1,iLen2,iLen3,iLen4,iLen5,iLen6;
iLen1=iLen2=iLen3=iLen4=iLen5=iLen6=0;

SQLColAttribute(hstmt,1,SQL_DESC_LENGTH,NULL,0,NULL,&iLen1);
SQLColAttribute(hstmt,2,SQL_DESC_LENGTH,NULL,0,NULL,&iLen2);

SQLColAttribute(hstmt,3,SQL_DESC_LENGTH,NULL,0,NULL,&iLen3);
SQLColAttribute(hstmt,4,SQL_DESC_LENGTH,NULL,0,NULL,&iLen4);
SQLColAttribute(hstmt,6,SQL_DESC_LENGTH,NULL,0,NULL,&iLen6);

if(iLen1<=0) iLen1=100;
if(iLen2<=0) iLen2=100;
if(iLen3<=0) iLen3=100;
if(iLen4<=0) iLen4=100;
if(iLen6<=0) iLen6=100;

char * szPkcatolog;
char * szPkSchema;      /* Primary key schema name */
char * szPkTable;      /* Primary key table name */
char * szPkCol;         /* Primary key column */
char * szPkName;
SQLSMALLINT iKeySeq=0;
SQLLEN       cbLen1,cbLen2,cbPkTable, cbPkCol, cbKeySeq,cbLen6;

szPkcatolog=(char*)malloc(iLen1+1);
szPkSchema=(char*)malloc(iLen2+1);
szPkTable=(char*)malloc(iLen3+1);
szPkCol=(char*)malloc(iLen4+1);
szPkName=(char*)malloc(iLen6+1);

memset(szPkcatolog,0x0,iLen1+1);
memset(szPkSchema,0x0,iLen2+1);
memset(szPkTable,0x0,iLen3+1);
memset(szPkCol,0x0,iLen4+1);
memset(szPkName,0x0,iLen6+1);

SQLBindCol(hstmt, 1, SQL_C_CHAR, szPkcatolog, iLen1+1, &cbLen1);
SQLBindCol(hstmt, 2, SQL_C_CHAR, szPkSchema, iLen2+1, &cbLen2);
```

```
    SQLBindCol(hstmt, 3, SQL_C_CHAR, szPkTable, iLen3+1, &cbPkTable);
    SQLBindCol(hstmt, 4, SQL_C_CHAR, szPkCol, iLen4+1, &cbPkCol);
    SQLBindCol(hstmt, 5, SQL_C_SSHORT, &iKeySeq, 2, &cbKeySeq);
    SQLBindCol(hstmt, 6, SQL_C_CHAR, szPkName, iLen6+1, &cbLen6);

    int i=0;
    PK_struct* pk_Info[5];
    while ((sr == SQL_SUCCESS) || (sr == SQL_SUCCESS_WITH_INFO))
    {
        sr=SQLFetch(hstmt);
        if (sr == SQL_SUCCESS || sr == SQL_SUCCESS_WITH_INFO)
        {
            pk_Info[i]=(PK_struct*)malloc(sizeof(PK_struct));

            pk_Info[i]->tab_cat =strdup(szPkcatolog);
            pk_Info[i]->PkSchema =strdup(szPkSchema);
            pk_Info[i]->PkTable =strdup(szPkTable);
            pk_Info[i]->PkColumnName =strdup(szPkCol);
            pk_Info[i]->seq =iKeySeq ;
            pk_Info[i]->Pkname =strdup(szPkName);
        }
        i=i+1;
    }
  SQLFreeHandle(SQL_HANDLE_STMT,hstmt);

  SQLDisconnect(hdbc);

  SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
  SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

## 8.1.9 SQLForeignKeys 外键信息查询示例：

代码如下：
Foreignkeys

```
struct FK_struct{
        char * Pk_tab_cat;
        char* PkSchema;
        char* PkTable;
        char* PkColumnName;

        char * Fk_tab_cat;
        char* FkSchema;
        char* FkTable;
```

```
        char* FkColumnName;

        int key_seq;
        int upRule;
        int edlRule;
        char * FkName;
        char * PkName;
          int    referRule;
    };

char szDSN[] ="ANGOO";
char szUID[] = "SYSDBA";
char szAuthStr[] ="SYSDBA";
SQLHENV    henv = NULL;
SQLHDBC    hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN sr = 0;
sr = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
sr                                                                    =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(SQLPOINTER)SQL_OV_ODBC3,SQL_I
S_INTEGER);

    sr = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);

    sr                                                                    =
SQLConnect(hdbc,(UCHAR*)szDSN,SQL_NTS,(UCHAR*)szUID,SQL_NTS,(UCHAR*)szAu
thStr,SQL_NTS);

    sr = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);


    char * pStrSchemaName="SYSDBA";
    char * pStrTableName="TT2";

    sr=SQLForeignKeys(hstmt,
                          (UCHAR*)NULL,0,
                      (UCHAR*)NULL,0,
                      (UCHAR*)NULL,0,
                      (UCHAR*)NULL,0,
                          (UCHAR*)pStrSchemaName,    SQL_NTS,              //
(UCHAR*)NULL,0,//                Schema name
                      (UCHAR*)pStrTableName,SQL_NTS);

    SQLLEN iLen1, iLen2,iLen3,iLen4,iLen5,iLen6,iLen7,iLen8,iLen12,iLen13;
```

```
 iLen1=iLen2=iLen3=iLen4=iLen6=iLen7=iLen8=iLen12=iLen13=0;

SQLColAttribute(hstmt,1,SQL_DESC_LENGTH,NULL,0,NULL,&iLen1);
SQLColAttribute(hstmt,2,SQL_DESC_LENGTH,NULL,0,NULL,&iLen2);
SQLColAttribute(hstmt,3,SQL_DESC_LENGTH,NULL,0,NULL,&iLen3);
SQLColAttribute(hstmt,4,SQL_DESC_LENGTH,NULL,0,NULL,&iLen4);

SQLColAttribute(hstmt,5,SQL_DESC_LENGTH,NULL,0,NULL,&iLen5);
SQLColAttribute(hstmt,6,SQL_DESC_LENGTH,NULL,0,NULL,&iLen6);
SQLColAttribute(hstmt,7,SQL_DESC_LENGTH,NULL,0,NULL,&iLen7);
SQLColAttribute(hstmt,8,SQL_DESC_LENGTH,NULL,0,NULL,&iLen8);

SQLColAttribute(hstmt,12,SQL_DESC_LENGTH,NULL,0,NULL,&iLen12);
SQLColAttribute(hstmt,13,SQL_DESC_LENGTH,NULL,0,NULL,&iLen13);

char *szPkcatolog;
char *szPkSchema;
char *szPkTable;
char *szPkCol;

char *szFkcatolog;
char *szFkSchema;
char *szFkTable;
char *szFkCol;

char * szPkName;
char * szFkName;
SQLSMALLINT iKeySeq,FK_UpdateRule,FK_DeleteRule;
  int FK_CheckYanchi;

int Flen=100;
if(iLen1<=0) iLen1=Flen;
if(iLen2<=0) iLen2=Flen;
if(iLen3<=0) iLen3=Flen;
if(iLen4<=0) iLen4=Flen;

if(iLen5<=0) iLen5=Flen;
if(iLen6<=0) iLen6=Flen;
if(iLen7<=0) iLen7=Flen;
if(iLen8<=0) iLen8=Flen;

if(iLen12<=0) iLen12=Flen;
if(iLen13<=0) iLen13=Flen;
```

```
szPkcatolog=(char*)malloc(iLen1+1);
szPkSchema=(char*)malloc(iLen2+1); //两个模式的
szPkTable=(char*)malloc(iLen3+1);
szPkCol=(char*)malloc(iLen4+1);

szFkcatolog=(char*)malloc(iLen5+1);
szFkSchema=(char*)malloc(iLen6+1);
szFkTable=(char*)malloc(iLen7+1);
szFkCol=(char*)malloc(iLen8+1);

szPkName=(char*)malloc(iLen12+1);
szFkName=(char*)malloc(iLen13+1);

memset(szPkcatolog,0x0,iLen1+1);
memset(szPkSchema,0x0,iLen2+1);
memset(szPkTable,0x0,iLen3+1);
memset(szPkCol,0x0,iLen4+1);

memset(szFkcatolog,0x0,iLen5+1);
memset(szFkSchema,0x0,iLen6+1);
memset(szFkTable,0x0,iLen7+1);
memset(szFkCol,0x0,iLen8+1);

memset(szFkName,0x0,iLen12+1);
memset(szPkName,0x0,iLen13+1);

SQLLEN
cbLen1,cbPkSchema,cbPkTable,cbPkCol,cbLen5,cbFkSchema,cbFkTable,cbFkCol ;
SQLLEN cbKeySeq,cbUpdateRule,cbDeleteRule,cbCheckYanchi,cbLen12,cbLen13;

SQLBindCol(hstmt, 1, SQL_C_CHAR, szPkcatolog, iLen1+1, &cbLen1);
SQLBindCol(hstmt, 2, SQL_C_CHAR, szPkSchema, iLen2+1, &cbPkSchema);
SQLBindCol(hstmt, 3, SQL_C_CHAR, szPkTable, iLen3+1, &cbPkTable);
SQLBindCol(hstmt, 4, SQL_C_CHAR, szPkCol, iLen4+1, &cbPkCol);

SQLBindCol(hstmt, 5, SQL_C_CHAR, szFkcatolog, iLen5+1, &cbLen5);
 SQLBindCol(hstmt, 6, SQL_C_CHAR, szFkSchema, iLen6+1, &cbFkSchema);
SQLBindCol(hstmt, 7, SQL_C_CHAR, szFkTable, iLen7+1, &cbFkTable);
SQLBindCol(hstmt, 8, SQL_C_CHAR, szFkCol, iLen8+1, &cbFkCol);

SQLBindCol(hstmt,    9, SQL_C_SSHORT, &iKeySeq, 2, &cbKeySeq);
SQLBindCol(hstmt, 10, SQL_C_SSHORT, &FK_UpdateRule, 2, &cbUpdateRule);
SQLBindCol(hstmt, 11, SQL_C_SSHORT, &FK_DeleteRule, 2, &cbDeleteRule);
```

```
SQLBindCol(hstmt, 12, SQL_C_CHAR, szFkName, iLen12+1, &cbLen12);
SQLBindCol(hstmt, 13, SQL_C_CHAR, szPkName, iLen13+1, &cbLen13);
SQLBindCol(hstmt, 14, SQL_C_LONG, &FK_CheckYanchi, 4, &cbCheckYanchi);

int i=0;
FK_struct* fk_Info[10];
while ((sr == SQL_SUCCESS) || (sr == SQL_SUCCESS_WITH_INFO))
{
    sr=SQLFetch(hstmt);
    if (sr == SQL_SUCCESS || sr == SQL_SUCCESS_WITH_INFO)
    {
        fk_Info[i]=(FK_struct*)malloc(sizeof(FK_struct));

        fk_Info[i]->Pk_tab_cat= strdup(szPkcatolog);
        fk_Info[i]->PkSchema= strdup(szPkSchema);
        fk_Info[i]->PkTable= strdup(szPkTable);
        fk_Info[i]->PkColumnName= strdup(szPkCol);

        fk_Info[i]->Fk_tab_cat = strdup(szFkcatolog);
        fk_Info[i]->FkSchema= strdup(szFkSchema);
        fk_Info[i]->FkTable= strdup(szFkTable);
        fk_Info[i]->FkColumnName= strdup(szFkCol);

        fk_Info[i]->key_seq= iKeySeq ;
        fk_Info[i]->upRule=FK_UpdateRule    ;
        fk_Info[i]->edlRule= FK_DeleteRule ;

        fk_Info[i]->FkName= strdup(szFkName);
        fk_Info[i]->PkName= strdup(szPkName);
        fk_Info[i]->referRule= FK_CheckYanchi;

    }
    i++;
}

SQLFreeHandle(SQL_HANDLE_STMT,hstmt);

SQLDisconnect(hdbc);

SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

## 8.2 C#部分

C#的示例代码：

### 8.2.1 关于 C#常见数据类型参数方式插入的示例源代码

1 配置 Dsn=ANGOO 的 ODBC 数据源
2 code1：数据类型 bigint ,smallint ,tinyint

```
string odbcconn_str = "DSN=ANGOO;USER=SYSDBA;PWD=SYSDBA";
OdbcConnection conn3 = new OdbcConnection(odbcconn_str);

try
  {
  OdbcParameter[] parameters ={new OdbcParameter("c1",OdbcType.BigInt,8),
                               new OdbcParameter("c2",OdbcType.SmallInt,2),
                               new OdbcParameter("c3",OdbcType.TinyInt,1)
                               };

  parameters[0].Value = 64134256475367;
  parameters[1].Value = (short)31456;
  parameters[2].Value = 114;

  parameters[0].Direction = ParameterDirection.Input;
  parameters[1].Direction = ParameterDirection.Input;
  parameters[2].Direction = ParameterDirection.Input;

  OdbcCommand cmd = new OdbcCommand();
  cmd.CommandType = CommandType.Text;
  cmd.CommandText = "insert into t_bint values(?,?,?)";
  cmd.Connection = conn3;
  conn3.Open();
  cmd.Parameters.Add(parameters[0]);
  cmd.Parameters.Add(parameters[1]);
  cmd.Parameters.Add(parameters[2]);
  int i = cmd.ExecuteNonQuery();

  conn3.Close();
  }
  catch (Exception ex)
   {
      MessageBox.Show(ex.Message.ToString());
      return;
```

```
    }
//=====================================================================
Code2: 数据类型 char,date,time,datetime
  string odbcconn_str = "DSN=ANGOO;USER=SYSDBA;PWD=SYSDBA";
 OdbcConnection conn1 = new OdbcConnection(odbcconn_str);

 try
 {
 OdbcParameter[] parameters ={new OdbcParameter("c1",OdbcType.Char,38),
                              new OdbcParameter("c2",OdbcType.Date,12),
                              new OdbcParameter("c3",OdbcType.Time,10),
                              new OdbcParameter("c4",OdbcType.DateTime,21)
                              };
 System.DateTime c2 = new DateTime(2015,2,4);
 System.TimeSpan c3 = new TimeSpan(15,45,59);

 System.DateTime c4 = new DateTime(2015,2,3,14,42,36);
 parameters[0].Value = "leiaq112";
 parameters[1].Value = c2 ;

 parameters[2].Value =c3;
 parameters[3].Value= c4 ;

 parameters[0].Direction = ParameterDirection.Input;
 parameters[1].Direction = ParameterDirection.Input;
 parameters[2].Direction = ParameterDirection.Input;
 parameters[3].Direction = ParameterDirection.Input;

 OdbcCommand cmd = new OdbcCommand();
 cmd.CommandType = CommandType.Text;
 cmd.CommandText = "insert into t_cdate values(?,?,?,?)";
 cmd.Connection = conn1;
 conn1.Open();
 cmd.Parameters.Add(parameters[0]);
 cmd.Parameters.Add(parameters[1]);
 cmd.Parameters.Add(parameters[2]);
 cmd.Parameters.Add(parameters[3]);
 int i = cmd.ExecuteNonQuery();

 conn1.Close();
 conn1.Dispose();

 }
 catch(Exception ex )
```

```
    {
        MessageBox.Show(ex.Message.ToString());
        return;
    }
Code3: 数据类型 int ,numeric
string odbcconn_str = "DSN=ANGOO;USER=SYSDBA;PWD=SYSDBA";
    OdbcConnection conn1 = new OdbcConnection(odbcconn_str);

    try
    {
    OdbcParameter[] parameters ={new OdbcParameter("c1",OdbcType.Int,4),
                                 new OdbcParameter("c2",OdbcType.Numeric,38)
                                };

    parameters[0].Value = 641;
    System.Decimal a = new Decimal(4598987.97655);
    parameters[1].Value = a;

    parameters[0].Direction = ParameterDirection.Input;
    parameters[1].Direction = ParameterDirection.Input;

    OdbcCommand cmd = new OdbcCommand();
    cmd.CommandType = CommandType.Text;
    cmd.CommandText = "insert into t_numeric values(?,?)";

    cmd.Connection = conn1;
    conn1.Open();
    cmd.Parameters.Add(parameters[0]);
    cmd.Parameters.Add(parameters[1]);
    int i = cmd.ExecuteNonQuery();

    conn1.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message.ToString());
        return;
    }
//=====================================================================
```

## 8.2.2 关于 C#调用 ODBC 存储过程的源代码

1 配置 Dsn=angoo415 的 ODBC 数据源

2 code：

==========

```
  string sConn = "Dsn=angoo415;user=SYSDBA;pwd=SYSDBA;";
  OdbcConnection conn = new OdbcConnection(sConn);
  OdbcParameter[]parameters={new
OdbcParameter("c1",OdbcType.Char,30),
   new OdbcParameter("c2", OdbcType.VarChar,50)};
//对参数赋值
   parameters[0].Value = "2009-7-15 16:24:11 dfsg";
   parameters[1].Value ="2009-7-15";
//指明参数类型 输入型，输出型，输入输出型
   parameters[0].Direction = ParameterDirection.InputOutput;
   parameters[1].Direction = ParameterDirection.InputOutput;
   conn.Open();
   OdbcCommand cmd = new OdbcCommand();
   cmd.CommandType = CommandType.StoredProcedure;
   cmd.CommandText = "execute TEST_num9(?,?)";
   cmd.Connection = conn;
   cmd.Parameters.Add(parameters[0]);
   cmd.Parameters.Add(parameters[1]);
   int i = cmd.ExecuteNonQuery();
   conn.Close();
```

---

```
procedure:
create procedure "SYSDBA"."TEST_num9" (
c1 in out char(30),c2 in out varchar(50)
)
AS
BEGIN
c1:='1999-5-12 12:45:32';
c2:='1999-5-12 ghgsfadasfadgadghdf' ;
END;
```

## 8.3 VB 部分

下面是 ODBC 的使用案例之一　　VB 使用

```
Global DBcn As New ADODB.Connection
Global DBmd As New ADODB.Command
Global DBrs As New ADODB.Recordset
```

```
Dim DBtmprs As New ADODB.Recordset
DBtmprs.CursorLocation = adUseClient
DBcn.Open "DSN=Agodbc;User ID=SYSDBA;Password=SYSDBA"
DBtmprs.CursorLocation = adUseClient
DBtmprs.Open "SELECT * FROM MM WHERE 用户名 = '" & 登陆用户.Text & "'", DBcn, adOpenStatic, adLockOptimistic, adCmdText
If DBtmprs.RecordCount = 0 Then
MsgBox ("对不起!你输入的登陆用户名不存在!请查正后输入!谢谢!")
登陆用户 = ""
DBtmprs.Close
DBcn.Close
GoTo 重来
End If

If 权限密码 = DBtmprs!密码 Then
主界面.Show

主界面.Tag = DBtmprs!用户权限 & DBtmprs!用户名
DBtmprs.Close
Else
MsgBox ("对不起!你输入的权限密码错误或权限密码不能为空!请查正后输入!谢谢!")
权限密码 = ""
DBtmprs.Close
DBcn.Close
GoTo 重来
End If
Unload 登陆界面
重来:
End Sub
```

# 九、常见问题以及解答

## 9.1 关于自动提交的问题：

ODBC 数据源的参数配置里面的**自动提交(auto commit),**其默认配置为 true，即命令发往服务器后，则提交。
在当自动提交参数设置为 false 时，命令发往服务器后，需要用户显式的发送”commit”命令到服务器，之前的操作才会被提交，否则被回滚。但有的时候，没有发送 commit 命令到服务器，但还是提交了 SQL 语句，这是为什么呢？
解答： 有一些 SQL 语句，比如 Create Table .Truncate Table 等 DDL 语句涉及到数据库资源的分配回收的问题，这类语句自动带有 COMMIT 的功能，此语句之前发送的 SQL 语句会在本条语句执行之前被提交。所以，这类语句会隐式的提交之前的事务，如果涉及用户主动回滚则需要注意这类 SQL 语句。

## 9.2 Prepare 执行方式的问题

大量的重复的 DML 语句的执行，采用 SQLPrepare 的方式执行，会提高效率。但 create table 等 DDL 语句，不可以采用 SQLPrepare 的方式来提升效率。

## 9.3 关于存储过程及函数的执行的问题

一般情况下我们执行 SQL 命令，直接发送 SQL 语句就可以了，在别的数据库执行存储过程有种调用方式是直接发送存储过程名和参数（有参数的情况），而安谷数据库的存储过程调用不是直接输入存储过程名而是需要键入 execute+存储过程名。发送 SQL 命令多一个 execute 关键字。

## 9.4 关于 64 位操作系统下 ODBC 的使用问题：

有时，64 位操作系统下有的用户发现 ODBC 无法配置，或者配置的 ODBC 无法使用
解答：此问题主要与 Driver Manager 的兼容性有关，在 64 位操作系统下，应用程序分为 32 位的应用和 64 位的应用，二者各自使用自身对应的驱动程序。安谷提供这两种驱动程序。需要注意的是他们的注册问题，此问题在第三章有介绍，64 位驱动对应的驱动管理程序在系统安装目录的 SYSTEM32 目录下，而 32 位驱动对应的驱动管理程序在 SysWOW64 的文件夹下

## 9.5 关于参数绑定的参数长度的问题。

在某些时候用户使用 SQLBindParameter 进行参数绑定时，会出现绑定参数正确，但是报错的问题。

解答：此问题多见于以 char 类型绑定某些数据结构。因为函数的最后一个参数需要传入实际参数的长度，某些开发者喜欢用 strlen（）得到数据的长度，因为此长度不会计算结尾处的'\0',所以参数的结尾'\0'不会传入到服务器，当在某些情况下处理此类参数时，服务器会认为此参数格式有问题而报错，解决办法是 计算长度时 strlen（）+1 ，把'\0'的长度纳入参数的长度范围，传入的参数自带'\0'。

## 9.6 字符集编码的问题

某些时候，数据库字符集和客户端字符集会出现不相同的情况，此时涉及转码。用户可能发现插入数据库的是一个值，而查询回来是另一个值(可能为乱码)。

解答：有一种办法是 更改两端字符集的一端使二者字符集一致，就不涉及转码问题。如果做不到，必须要转码，则要保证转码的一致性，即入库的时候转码了，那么出库的时候也需要转码，入库的时候没有转码，出库的时候也不要转码。例如：第八章示例 8.1.6 和 8.1.7 中的两种 LOB 类型的数据的入库和出库操作，CLOB 以 SQL_C_CHAR 的方式绑定进入数据库，如果客户端和服务器端的编码不同，则转码后再入库，出库时，CLOB 转码再出库。而 BLOB 类型的数据以 SQL_C_BINARY 的方式绑定进入数据库，二进制入库不需转码，出库时，二进制文件不需要转码。 但是 如果 CLOB 以 SQL_C_BINARY 的方式绑定进入数据库，二进制入库不需转码，出库时，服务器端与客户端的字符集不同时，CLOB 出库时转码，如有中文或其他特殊字符则会造成乱码问题。

## 9.7 参数绑定和输出绑定的类型映射问题

有的数据类型较为特殊，可能在 SQL 的通用数据类型里面找不到对应的数据类型来绑定，此时若此数据类型可以用字符串类型表示的话，当可以用 SQL_C_CHAR 类型来绑定，如八章的 8.1.3 示例的 time with time zone 和 datetime with time zone 数据类型。