

# 虚谷数据库

## ODBC 标准接口 V11.3.2

### 开发指南

文档版本 01

发布日期 2024-07-30



版权所有 © 2024 成都虚谷伟业科技有限公司。

## 声明

未经本公司正式书面许可，任何企业和个人不得擅自摘抄、复制、使用本文档中的部分或全部内容，且不得以任何形式进行传播。否则，本公司将保留追究其法律责任的权利。

用户承诺在使用本文档时遵守所有适用的法律法规，并保证不以任何方式从事非法活动。不得利用本文档内容进行任何侵犯他人权益的行为。

## 商标声明



为成都虚谷伟业科技有限公司的注册商标。

本文档提及的其他商标或注册商标均非本公司所有。

## 注意事项

您购买的产品或服务应受本公司商业合同和条款的约束，本文档中描述的部分产品或服务可能不在您的购买或使用范围之内。由于产品版本升级或其他原因，本文档内容将不定期进行更新。

除非合同另有约定，本文档仅作为使用指导，所有内容均不构成任何声明或保证。

## 成都虚谷伟业科技有限公司

地址：四川省成都市锦江区锦盛路 138 号佳霖科创大厦 5 楼 3-14 号

邮编：610023

网址：[www.xugudb.com](http://www.xugudb.com)

# 前言

## 概述



本文档主要介绍虚谷数据库 ODBC 驱动接口的主要功能和其简单的外围应用，旨在帮助使用虚谷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。

## 读者对象

- 数据库管理员
- 数据库用户

## 符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 <b>注意</b>	用于传递设备或环境安全警示信息，若不避免，可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。
 <b>说明</b>	对正文中重点信息的补充说明。“说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

## 修改记录

文档版本	发布日期	修改说明
01	2024-07-30	第一次发布

# 目录

1	ODBC 概述	1
1.1	接口简介	1
1.2	系统架构	1
1.3	开发流程	1
2	快速上手	4
2.1	简介	4
2.2	句柄	4
2.2.1	句柄说明	4
2.2.2	环境句柄	4
2.2.3	连接句柄	5
2.2.4	语句句柄	6
2.2.5	描述符句柄	7
2.3	ODBC 安装与配置	7
2.3.1	注册虚谷 ODBC 驱动	7
2.3.2	注册数据源	8
3	数据类型	16
3.1	虚谷数据库数据类型	16
3.2	ODBC 标准的数据类型与 C 数据类型的绑定	19
4	编程指导	22
4.1	功能说明	22
4.1.1	连接数据源	22
4.1.2	获取驱动程序和数据源信息	22
4.1.3	设置或者获取驱动程序属性	22
4.1.4	设置或者获取描述符字段	23
4.1.5	准备 SQL 语句	23
4.1.6	提交 SQL 语句	23
4.1.7	检索结果集及其相关信息	23

4.1.8	获取数据源系统表信息 .....	24
4.1.9	终止语句执行 .....	24
4.1.10	中断连接 .....	25
4.2	虚谷数据库 ODBC 连接使用 .....	25
4.2.1	申请环境句柄与连接句柄 .....	25
4.2.2	如何与数据源进行连接 .....	26
4.2.3	编写连接字符串 .....	29
4.2.4	设置与取得连接属性 .....	30
4.2.5	数据库连接断开与释放 .....	31
4.3	虚谷数据库 ODBC 应用编程 .....	33
4.4	使用存储过程和函数 .....	34
4.4.1	存储过程和函数字典信息获取 .....	34
4.4.2	存储模块创建 .....	36
4.4.3	存储模块调用 .....	36
4.4.4	存储函数使用示例 .....	36
4.5	虚谷数据库 ODBC 常用 API 介绍 .....	38
4.5.1	SQLAllocHandle 函数 .....	38
4.5.2	SQLConnect 函数 .....	38
4.5.3	SQLDriverConnect 函数 .....	39
4.5.4	SQLBrowseConnect 函数 .....	40
4.5.5	SQLPrepare 函数 .....	40
4.5.6	SQLBindParamenter 函数 .....	41
4.5.7	SQLBindCol 函数 .....	41
4.5.8	SQLColAttribute 函数 .....	42
4.5.9	SQLColAttribute 函数 .....	43
4.5.10	SQLExecuteDirect 函数 .....	43
4.5.11	SQLFetch 函数 .....	44
4.5.12	SQLTables 函数 .....	44
4.5.13	SQLColumns 函数 .....	45
4.5.14	SQLPrimaryKeys 函数 .....	45

4.5.15	SQLForeignKeys	46
4.5.16	SQLProcedures 函数	47
4.5.17	SQLStatics 函数	48
4.5.18	SQLPutData 函数	48
4.5.19	SQLGetData 函数	49
4.5.20	SQLMoreResults 函数	49
5	示例说明	51
5.1	C/C++	51
5.1.1	执行 Select 语句示例代码	51
5.1.2	执行 Insert 语句示例代码	52
5.1.3	执行 Update 语句示例代码	54
5.1.4	执行 Delete 语句示例代码	55
5.1.5	CLOB 大对象插入及查询示例代码	56
5.1.6	BLOB 大对象插入及查询示例代码	58
5.1.7	SQLPrimaryKeys 主键信息查询示例代码	59
5.1.8	SQLColumns 列信息查询示例代码	60
5.2	C	63
5.2.1	建立连接示例	63
5.2.2	常见数据类型参数方式插入示例	63
5.2.3	非参数方式更改数据示例	64
5.2.4	DataTable 使用示例	64
5.2.5	调用存储过程示例	64
5.2.6	插入二进制数据示例	65
5.3	VB	66
5.4	PHP	66
5.4.1	查询示例	66
5.4.2	Prepare 示例	67
5.4.3	PDO 示例	67
6	错误码与常见错误定位	68
6.1	错误码详解	68

---

6.2	常见错误定位 .....	81
6.2.1	数据库连接失败 .....	81
6.2.2	虚谷 ODBC 驱动无法正常使用 .....	81

# 1 ODBC 概述

## 1.1 接口简介

ODBC (Open Database Connectivity, 开放式数据库连接), 建立了一组规范, 提供了一组数据库访问的标准 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。可为异构数据库提供统一访问接口以进行程序处理。这使开发者不需要以特殊的数据库管理系统 DBMS 为目标, 或者了解不同支撑背景的数据库的详细细节, 就能够开发和发布应用程序。

## 1.2 系统架构

ODBC 总体结构有四个组件:

- 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。
- 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, 或把调用请求传送到驱动程序。
- 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。如有必要, 驱动程序可修改一个应用程序请求, 使请求与相关的 DBMS 支持的语法一致。
- 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。

以上四个组件关系如图1-1所示。

## 1.3 开发流程

ODBC 使用流程框架如图1-2所示。



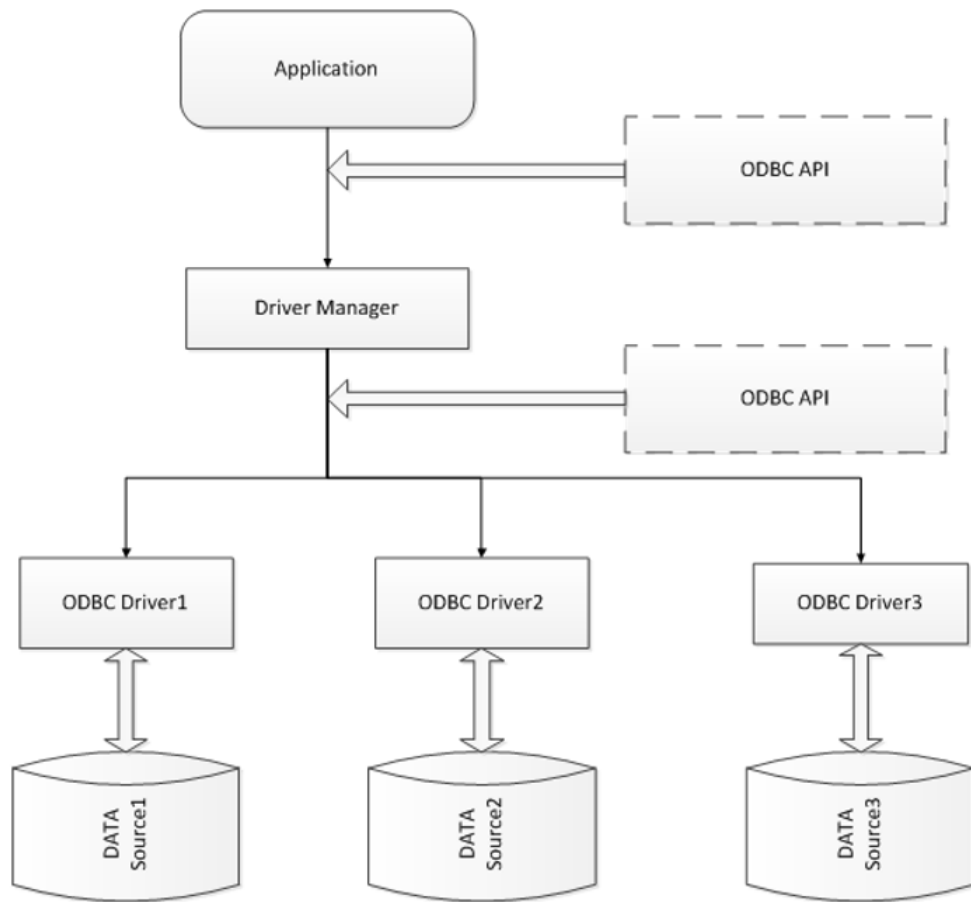


图 1-1 odbc 应用架构图

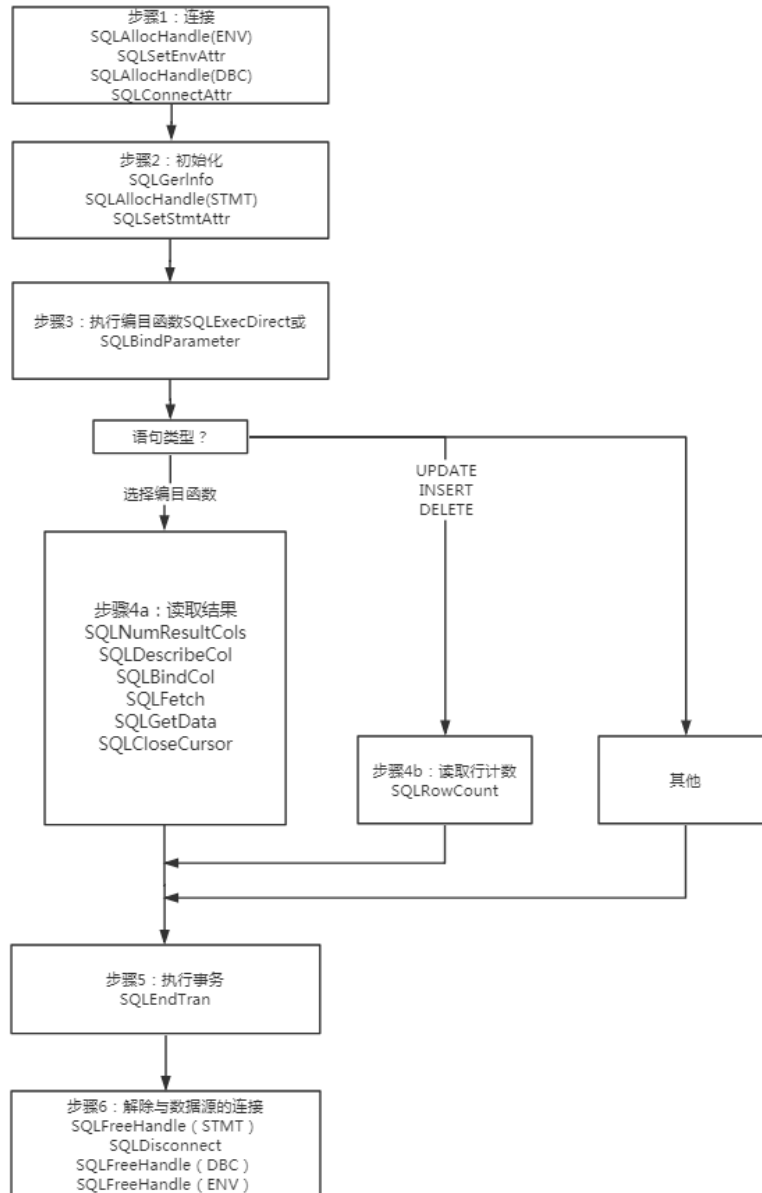


图 1-2 odbc 开发流程图

# 2 快速上手

## 2.1 简介

本章结合虚谷数据库的特点，系统地介绍 ODBC 的基本概念以及虚谷数据库 ODBC 标准接口的使用方法，以使用户更好地使用虚谷数据库 ODBC 编写应用程序。ODBC 提供访问不同类型的数据库的途径。结构化查询语言 SQL 是一种用来访问数据库的语言。通过使用 ODBC，应用程序能够使用相同的源代码和各种各样的数据库交互。这使得开发者不需要以特殊的数据库管理系统 DBMS 为目标，或者了解不同支撑背景的数据库的详细细节，就能够开发和发布客户/服务器应用程序。

虚谷数据库 ODBC 遵照 Microsoft ODBC 3.0 规范设计与开发，实现了 ODBC 应用程序与虚谷数据库的互连接口。用户可以直接调用虚谷数据库 ODBC 接口函数访问虚谷数据库。

## 2.2 句柄

### 2.2.1 句柄说明

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透明的。用户只能通过调用相应的功能 API 函数来得到需要的信息和完成 SQL 的执行。虚谷数据库 ODBC 包含的句柄如表 2-1 所示。

表 2-1 句柄类型

句柄	说明
SQLHENV	环境句柄
SQLHDBC	连接句柄
SQLHSTMT	语句句柄
SQLHDESC	描述符句柄

### 2.2.2 环境句柄

环境句柄是用于访问数据的全局上下文；与环境相关联的任何信息都是全局性的，例如：

- 环境状态
- 当前环境级别诊断
- 当前在环境中分配的连接的句柄
- 每个环境属性的当前设置

在实现 ODBC（驱动程序管理器或驱动程序）的代码段中，环境句柄标识包含此信息的结构。

在 ODBC 应用程序中的环境句柄始终用于以下情形：

- 调用 SQLDataSources
- 调用 SQLDrivers

有时用于以下情形：

- 调用 SQLAllocHandle
- 调用 SQLEndTran
- 调用 SQLFreeHandle
- 调用 SQLGetDiagField
- 调用 SQLGetDiagRec

实现 ODBC 的每个代码段（驱动程序管理器或驱动程序）都包含一个或多个环境句柄。例如，驱动程序管理器为连接到它的每个应用程序维护单独的环境句柄。

环境句柄通过 SQLAllocHandle 分配，并与 SQLFreeHandle 一起释放。

### 2.2.3 连接句柄

连接句柄包含驱动程序和数据源。连接句柄标识每个连接，同时定义使用的驱动程序及其对应的数据源。在实现 ODBC 的代码段（驱动程序管理器或驱动程序）内，连接句柄标识包含连接信息的结构，如下所示：

- 连接的状态
- 当前连接级别诊断
- 连接上当前分配的语句和描述符的句柄
- 每个连接属性的当前设置

驱动程序支持多个连接，ODBC 管理器不会阻止这些连接。在特定 ODBC 环境中，多个连接句柄可能指向各种驱动程序和数据源、同一驱动程序和不同的数据源，甚至是与同一驱动程序和数据源的多个连接。某些驱动程序会限制它们支持的活动连接数，SQLGetInfo 中的 SQL\_MAX\_DRIVER\_CONNECTIONS 选项指定特定驱动程序支持的活动连接数。

连接句柄主要用于以下情形：

- 连接到数据源（SQLConnect、SQLDriverConnect、SQLBrowseConnect）
- 断开与数据源的连接（SQLDisconnect）
- 获取有关驱动程序和数据源的信息（SQLGetInfo）
- 检索诊断（SQLGetDiagField、SQLGetDiagRec）
- 执行事务（SQLEndTran）
- 在设置和获取连接属性（SQLSetConnectAttr、SQLGetConnectAttr）
- 获取 SQL 语句的本机格式（SQLNativeSql）

连接句柄通过 SQLAllocHandle 分配，通过 SQLFreeHandle 释放。

## 2.2.4 语句句柄

语句句柄可被视为一种 SQL 语句，例如 SELECT FROM Employee。但语句句柄不只是 SQL 语句，还包含与该 SQL 语句相关联的所有信息，如语句所创建的所有结果集和执行语句时使用的参数。

每个语句都有独立的语句句柄标识，语句与单个连接相关联，但该连接上可有多个附属语句。

某些驱动程序会限制其支持的活动语句的数量，SQLGetInfo 中的

SQL\_MAX\_CONCURRENT\_ACTIVITIES 选项指定驱动程序在单个连接上支持的活动语句

数。如语句具有挂起的结果，则将其定义为活动状态，其中的结果是结果集或受 INSERT、UPDATE 或 DELETE 语句影响的行数，或正在通过多次调用 SQLPutData 发送数据。

在实现 ODBC（驱动程序管理器或驱动程序）的代码段内，语句句柄标识包含语句信息的结构，如：

- 语句的状态
- 当前语句级别诊断
- 绑定到语句的参数和结果集列的应用程序变量的地址
- 每个语句特性的当前设置

语句句柄用于大多数 ODBC 函数，在函数中主要用于以下情形：

- 绑定参数和结果集列（SQLBindParameter 和 SQLBindCol）
- prepare 和 execute 语句执行（SQLPrepare、SQLExecute 和 SQLExecDirect）
- 检索元数据（SQLColAttribute 和 SQLDescribeCol）
- 提取结果（SQLFetch）
- 检索诊断（SQLGetDiagField 和 SQLGetDiagRec）
- 目录函数（SQLColumns、SQLTables 等）

语句句柄通过 SQLAllocHandle 分配，通过 SQLFreeHandle 释放。

## 2.2.5 描述符句柄

描述符句柄是描述 SQL 语句的参数或结果集的列的元数据的集合，描述符角色包括：

- 应用程序参数描述符（APD）：包含有关绑定到 SQL 语句中的参数的应用程序缓冲区的信息，如其地址、长度和 C 数据类型。
- 实现参数描述符（IPD）：包含 SQL 语句中参数的相关信息，例如 SQL 语句的 SQL 数据类型、长度和 NULL 属性（是否可以取空值）。
- 应用程序行描述符（ARD）：包含有关绑定到结果集中的列的应用程序缓冲区的信息，如其地址、长度和 C 数据类型。
- 实现行描述符（IRD）：包含有关结果集中的列的信息，如其 SQL 数据类型、长度和 NULL 属性。

分配语句时，会自动分配四个描述符并逐个填充角色，这些描述符称为自动分配的描述符，始终与该语句相关联；应用程序还可以分配带有 SQLAllocHandle 的描述符，称为显式分配的描述符。它们在连接上进行分配，并可与该连接上的一个或多个语句相关联，以满足这些语句上的 APD 或 ARD 的角色。

## 2.3 ODBC 安装与配置

### 2.3.1 注册虚谷 ODBC 驱动

#### 操作步骤

- Windows 版本
  1. 进入 ODBC 目录下，会有 XuguOdbc.dll 和 XuguODBC\_install.exe 文件。
  2. 以管理员身份执行 XuguODBC\_install.exe 可以将虚谷 ODBC 驱动注册到 ODBC 管理器的驱动程序列表中。

#### 说明

ODBC 驱动严格区分 X86 与 X64 架构，应用程序应与配置使用的 ODBC 源一致。

- Linux 版本

目前，虚谷 ODBC 驱动已对 RedHat、CentOS 和 Fedora 等 Linux 版本进行支持，可通过官方渠道获取 RPM 安装包进行安装。

## 2.3.2 注册数据源

### 2.3.2.1 Windows32 位创建数据源

Windows32 位 ODBC 数据源管理程序 odbcad32.exe 位于 C:\\_Windows\_SysWOW64 目录下。

#### 操作步骤

1. 运行 odbcad32.exe 或者选择控制面板-> 管理工具->ODBC 数据源。如图 2-1 所示。



图 2-1 运行 odbcad32.exe 文件

2. 选择系统 DSN > 添加 > 选择数据源 XuguSQL 11.2 > 完成，如图 2-2 所示。
3. 配置连接参数，连接参数说明如表 2-2 所示。

表 2-2 连接参数

参数	参数值
数据源名称	XGDB
数据源描述	XuguSQL Server ODBC 11.2 Driver DSN
服务端地址	127.0.0.1

参数	参数值
端口	5138
数据库名称	SYSTEM
用户名	SYSDBA
密码	SYSDBA

注册数据源时，有多种参数可供选择如图 2-3 所示。

参数的详细解释如表 2-3 所示。

**表 2-3** 注册数据源参数说明

参数名	是否为必填项	参数说明
数据源名称 (DSN)	必填	DSN 是应用程序调用的依据，是一个数据源的标识
数据源描述 (Desc)	选填	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
服务端地址 (Server)	必填	数据库服务端的 IP 地址
端口 (Port)	必填	虚谷数据库端口号
数据库名称 (Database)	必填	此属性决定了应用程序调用该数据源时连接的数据库名
用户名 (User、UID)	必填	此属性描述了连接数据源时的用户名信息
密码 (Password、PWD)	必填	与用户名相匹配的密码



参数名	是否为必填项	参数说明
启用安全连接方式 (UseSSL)	选填	此属性决定了应用程序与数据库服务端之间的数据传输是否加密
自动提交 (AutoCommit)	选填	取值为 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 语句到服务端, 驱动程序都会自动提交; 而取值为非自动提交时, 则需要应用程序自己发送 commit 到服务端, 之前的 SQL 语句才会被提交
严格提交 (StrictCommit)	选填	取值为 true 或 false。当设置为 true 时, 数据库服务端记实了事务日志才向客户端返回
返回模式名 (RetSchema)	选填	取值为 true 或 false。当设置为 true 时, 数据库服务端以模式名-表名-返回表名, 否则仅返回表名
返回游标 ID (RetCursorId)	选填	取值为 true 或 false。当设置为 true 时, 存储过程的游标型输出参数返回其 ID, 否则以结果集形式返回

参数名	是否为必填项	参数说明
返回 RowID (RetRowID)	选填	取值为 true 或 false。当设置为 true 时，数据库服务端会在查询的结果集的最后一列追加 RowID 列
启动服务端游标 (UseServer-Cursor)	选填	取值为 true 或 false。当设置为 true 时，驱动程序不会一次性接收所有结果集，通过 SQLFetch 从服务端获取当前所需的行数；否则驱动程序将在内存中缓存所有结果集
启用大对象描述符 (LobRet)	选填	取值为 true 或 false。当设置为 true 时，若查询的结果集中包含大对象字段，大对象字段返回其描述符；否则以其数据返回。若需要处理大对象字段则应将其设置为 false 以获得更高的效率，否则应将其设置为 true 以加快结果集接收并节省内存
异步接收结果集 (RecvIsAsyn)	选填	取值为 true 或 false。当设置为 true 时，若查询的结果集行数大于设置的启用异步接收的结果集行数时，驱动程序将开启子线程接收剩余结果集，主线程将返回；否则将使用主线程接收所有结果集

参数名	是否为必填项	参数说明
缓存只进游标结果集 (CacheFOCR)	选填	取值为 true 或 false。当设置为 false 时, SQLFetch (或 SQLFetchScroll 的 FetchOrientation 参数为 SQL_FETCH_NEXT) 访问过的结果集行数据将被释放。通过 SQLSetStmtAttr 设置 SQL_ATTR_CURSOR_SCROLLABLE 和 SQL_ATTR_CURSOR_TYPE 会影响此参数取值
字符集 (CharSet)	选填	连接的字符集属性。取值可以为: GBK、UTF8、GB2312 等
时区 (Timezone)	选填	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)
隔离级别 (IsoLevel)	选填	读未提交 (READ UNCOMMITTED), 读已提交 (READ COMMITED), 可重复读 (REPEATABLE READ), 序列化 (SERIALIZABLE)。一般选择读已提交

参数名	是否为必填项	参数说明
加锁超时 (LockTimeout)	选填	单位：毫秒。表示本连接中的事务在争用锁时，最多等候的时间，若在此时间内加锁不成功，则回滚事务并报错
启用异步接收的结果集行数 (SubthreadRows)	选填	当启用异步接收结果集时，若查询的结果集行数大于此设定的值，驱动程序将开启子线程接收剩余结果集

4. 测试连接完成。
5. 点击确定按钮，退出注册程序。

### 2.3.2.2 Windows64 位创建数据源

Windows64 位 ODBC 数据源管理程序 odbcad32.exe 位于 C:\Windows\System32 目录下。

#### 操作步骤

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 XuguSQL 11.2 > 完成。
3. 配置连接参数。
4. 测试连接完成。
5. 点击确定按钮，退出注册程序。

### 2.3.2.3 Linux 创建数据源

阅读 README.md 文件，完成对 ODBC 数据源信息的注册工作，更多关于注册 ODBC 数据源的信息可查阅/usr/local/etc/odbc.ini 文件或/etc/odbc.ini 文件。注册数据源如图 2-2 所示。



图 2-2 注册数据源



图 2-3 数据源配置

# 3 数据类型

## 3.1 虚谷数据库数据类型

虚谷数据库数据类型说明和 ODBC 的通用 SQL 数据类型的映射关系如表 3-1 所示。

表 3-1 虚谷数据库数据类型信息

虚谷数据库数据类型	数据长度（所占字节）	ODBC SQL 公共数据类型	说明
Char(n)	n 字节，最大不超过 64K	SQL_CHAR	固定串长度为 n 的字符串
Varchar(n)	n 字节，最大不超过 64K	SQL_CHAR	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	SQL_BINARY	固定长度为 n 的二进制数据
Tinyint	1 字节	SQL_TINYINT	精度为 3，标度为 0 的有符号精确数值，范围：-128 127
Smallint	2 字节	SQL_SMALLINT	精度为 5，标度为 0 的有符号精确数值，范围：-32768 32767
Integer	4 字节	SQL_INTEGER	精度为 10，标度为 0 的有符号精确数值，范围：-2147483648 2147483647

虚谷数据库数据类型	数据长度（所占字节）	ODBC SQL 公共数据类型	说明
Bigint	8 字节	SQL_BIGINT	精度为 19，标度为 0 的有符号精确数值，范围：-9223372036854775808 9223372036854775807
Float	4 字节	SQL_FLOAT	单精度浮点数
Double	8 字节	SQL_DOUBLE	双精度浮点数
Bool	1 字节	SQL_TINYINT、SQLCHAR	布尔类型，取值 true/false 或者'T'/'F'
Numeric(p,s)	20 字节	SQL_NUMERIC	精度为 p，标度为 s 的有符号精确数值
Time	4 字节	SQL_TIME	时间数据类型，时分秒字段
Datetime	8 字节	SQL_DATETIME	时间戳数据类型，年月日时分秒字段
Date	4 字节	SQL_DATE	日期数据类型，年月日字段
Time with time zone	6 字节	SQL_CHAR	时间数据类型，时分秒，时区字段
Datetime with time zone	10 字节	SQL_CHAR	时间戳数据类型，年月日时分秒，时区字段
Blob	最大不超过 4G	SQL_LONGVARCHAR	二进制大对象类型字段



虚谷数据库数据类型	数据长度（所占字节）	ODBC SQL 公共数据类型	说明
Clob	最大不超过 4G	SQL_LONGVARCHAR	字符大对象的存储字段
Interval year	4 字节	SQL_INTERVAL_YEAR	年间隔，即两个日期之间的年数
Interval month	4 字节	SQL_INTERVAL_MONTH	月间隔，即两个日期之间的月数
Interval day	4 字节	SQL_INTERVAL_DAY	日间隔，即两个日期之间的天数
Interval hour	4 字节	SQL_INTERVAL_HOUR	时间间隔，即为两个日期/时间之间的小时数
Interval minute	4 字节	SQL_INTERVAL_MINUTE	分间隔，即为两个日期/时间之间的分钟数
Interval second	8 字节	SQL_INTERVAL_SECOND	秒间隔，即为两个日期/时间之间的秒数
Interval day to hour	4 字节	SQL_INTERVAL_DAY_TO_HOUR	日时间间隔，即为两个日期/时间之间的日小时数
Interval day to minute	4 字节	SQL_INTERVAL_DAY_TO_MINUTE	日时分间隔，即为两个日期/时间之间的日小时分钟数
Interval day to second	8 字节	SQL_INTERVAL_DAY_TO_SECOND	日时分秒间隔，即为两个日期/时间之间的日小时分钟秒数

虚谷数据库数据类型	数据长度（所占字节）	ODBC SQL 公共数据类型	说明
Interval hour to minute	4 字节	SQL_INTERVAL_HOUR_TO_MINUTE	时分间隔，即为两个日期/时间之间的小时分钟数
Interval hour to second	8 字节	SQL_INTERVAL_HOUR_TO_SECOND	时分秒间隔，即为两个日期/时间之间的小时分钟秒数
Interval minute to second	8 字节	SQL_INTERVAL_MINUTE_TO_SECOND	分秒间隔，即为两个日期/时间之间的分钟秒数
Interval year to month	4 字节	SQL_INTERVAL_YEAR_TO_MONTH	年月间隔，即两个日期之间的年月数

## 3.2 ODBC 标准的数据类型与 C 数据类型的绑定

表 3-2 虚谷数据库数据类型信息

虚谷数据库数据类型	ODBC SQL 公共数据类型	ODBC 绑定的 C 映射的数据类
Char(n)	SQL_CHAR	SQL_C_CHAR
Varchar(n)	SQL_CHAR	SQL_C_CHAR
Binary(n)	SQL_BINARY	SQL_C_BINARY
Tinyint	SQL_TINYINT	SQL_C_TINYINT
Smallint	SQL_SMALLINT	SQL_C_SHORT
Integer	SQL_INTEGER	SQL_C_LONG

虚谷数据库数据类型	ODBC SQL 公共数据类型	ODBC 绑定的 C 映射的数据类
Bigint	SQL_BIGINT	SQL_C_SBIGINT
Float	SQL_FLOAT	SQL_C_FLOAT
Double	SQL_DOUBLE	SQL_C_DOUBLE
Bool	SQL_TINYINT、SQLCHAR	SQL_C_NUMERIC/SQL_C_CHAR
Numeric(p,s)	SQL_NUMERIC	SQL_C_CHAR
Time	SQL_TIME	SQL_C_TIME/SQL_C_TYPE_TIME/SQL_C_CHAR
Datetime	SQL_DATETIME	SQL_C_TIMESTAMP/SQL_C_TYPE_TIMESTAMP/SQL_C_CHAR
Date	SQL_DATE	SQL_C_DATE/SQL_C_TYPE_DATE/SQL_C_CHAR
Time with time zone	SQL_CHAR	SQL_C_CHAR
Datetime with time zone	SQL_CHAR	SQL_C_CHAR
Blob	SQL_LONGVARBINARY	SQL_C_BINARY
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

虚谷数据库数据类型	ODBC SQL 公共数据类型	ODBC 绑定的 C 映射的数据类
Interval hour	SQL_INTERVAL_HOUR	SQL_C_INTERVAL_HOUR/S QL_C_CHAR
Interval minute	SQL_INTERVAL_MINUTE	SQL_C_INTERVAL_MINUTE/ SQL_C_CHAR
Interval second	SQL_INTERVAL_SECOND	SQL_C_INTERVAL_SECON D/SQL_C_CHAR
Interval day to hour	SQL_INTERVAL_DAY_TO_H OUR	SQL_C_INTERVAL_DAY_TO _HOUR/SQL_C_CHAR
Interval day to minute	SQL_INTERVAL_DAY_TO_M INUTE	SQL_C_INTERVAL_DAY_TO _MINUTE/SQL_C_CHAR
Interval day to second	SQL_INTERVAL_DAY_TO_S ECOND	SQL_C_INTERVAL_DAY_TO _SECOND/SQL_C_CHAR
Interval hour to minute	SQL_INTERVAL_HOUR_TO_ MINUTE	SQL_C_INTERVAL_HOUR_T O_MINUTE/SQL_C_CHAR
Interval hour to second	SQL_INTERVAL_HOUR_TO_ SECOND	SQL_C_INTERVAL_HOUR_T O_SECOND/SQL_C_CHAR
Interval minute to second	SQL_INTERVAL_MINUTE_T O_SECOND	SQL_C_INTERVAL_MINUTE _TO_SECOND/SQL_C_CHA R
Interval year to month	SQL_INTERVAL_YEAR_TO_ MONTH	SQL_C_INTERVAL_YEAR_T O_MONTH/SQL_C_CHAR

# 4 编程指导

## 4.1 功能说明

### 4.1.1 连接数据源

下列函数用于连接数据源：

- **SQLAllocHandle**：分配环境、连接、语句或者描述符句柄。
- **SQLConnect**：建立与驱动程序或者数据源的连接。访问数据源的连接句柄包含了状态、事务申明和错误信息的所有连接信息。
- **SQLDriverConnect**：与 **SQLConnect** 相似，用来连接到驱动程序或者数据源。但它比 **SQLConnect** 支持数据源更多的连接信息，提供了一个对话框来提示用户设置所有的连接信息以及系统信息表没有定义的数据源。
- **SQLBrowseConnect**：支持一种交互方法来检索或者列出连接数据源所需要的属性和属性值。每次调用函数可以获取一个连接属性字符串，当检索完所有的属性值，就建立起与数据源的连接，并且返回完整的连接字符串，否则提示缺少连接属性信息，用户根据此信息重新输入连接属性值再次调用此函数进行连接。

### 4.1.2 获取驱动程序和数据源信息

下列函数用于获取驱动程序和数据源信息：

- **SQLDataSources**：能够被多次调用以获取应用程序所有可用的数据源的名字。
- **SQLDrivers**：返回所有已安装的驱动程序信息，包括驱动描述以及驱动属性。
- **SQLGetInfo**：返回连接的驱动程序和数据源的元信息。
- **SQLGetFunctions**：返回指定的驱动程序是否支持某个特定函数的信息。
- **SQLGetTypeInfo**：返回指定的数据源支持的数据类型的信息。

### 4.1.3 设置或者获取驱动程序属性

下列函数用于设置或者获取驱动程序属性：

- **SQLSetConnectAttr**：设置连接属性值。
- **SQLGetConnectAttr**：获取连接属性值。
- **SQLSetEnvAttr**：设置环境属性值。

- SQLGetEnvAttr: 获取环境属性值。
- SQLSetStmtAttr: 设置语句属性值。
- SQLGetStmtAttr: 获取语句属性值。

#### 4.1.4 设置或者获取描述符字段

下列函数用于设置或者获取描述符字段：

- SQLGetDescField: 获取单个描述符字段的值。
- SQLGetDescRec: 获取当前描述符记录的多个字段的值。
- SQLSetDescField: 设置单个描述符字段的值。
- SQLSetDescRec: 设置描述符记录的多个字段。

#### 4.1.5 准备 SQL 语句

- SQLPrepare: 准备要执行的 SQL 语句。
- SQLBindParameter: 在 SQL 语句中分配参数的缓冲区。
- SQLGetCursorName: 获取与语句句柄相关的游标名称。
- SQLSetCursorName: 设置与语句句柄相关的游标名称。

#### 4.1.6 提交 SQL 语句

下列函数用于提交 SQL 请求：

- SQLExecute: 执行准备好的 SQL 语句。
- SQLExecDirect: 执行一条 SQL 语句。
- SQLNativeSql: 返回驱动程序对一条 SQL 语句的翻译。
- SQLDescribeParam: 返回对 SQL 语句中指定参数的描述。
- SQLNumParams: 返回 SQL 语句中参数的个数。
- SQLParamData: 与 SQLPutData 联合使用在运行时给参数赋值。
- SQLPutData: 在 SQL 语句运行时给部分或者全部参数赋值。

#### 4.1.7 检索结果集及其相关信息

下列函数用于检索结果集及其相关信息：

- SQLRowCount: 返回 INSERT、UPDATE 或者 DELETE 等语句影响的行数。
- SQLNumResultCols: 返回结果集中列的数目。
- SQLDescribeCol: 返回结果集中列的描述符记录。

- SQLColAttribute: 返回结果集中列的属性。
- SQLBindCol: 为结果集中的列分配缓冲区。
- SQLFetch: 在结果集中检索下一行记录。
- SQLFetchScroll: 返回指定的结果行。
- SQLGetData: 返回结果集中当前行某一列的值。
- SQLSetPos: 在取到的数据集中设置游标的位置, 这个记录集中的数据能够刷新、更新或者删除。
- SQLBulkOperations: 执行块插入和块书签操作, 其中包括根据书签更新、删除或者取数据。
- SQLMoreResults: 确定是否能够获得更多的结果集, 如果能就执行下一个结果集的初始化操作。
- SQLGetDiagField: 返回一个字段值或者一个诊断数据记录。
- SQLGetDiagRec: 返回多个字段值或者一个诊断数据记录。

### 4.1.8 获取数据源系统表信息

下列函数用于取得数据源系统表信息:

- SQLColumnPrivileges: 返回关于指定表的列的列表以及相关的权限信息。
- SQLColumns: 返回指定表的列信息的列表。
- SQLForeignKeys: 返回指定表的外键信息的列表。
- SQLPrimaryKeys: 返回指定表的主键信息的列表。
- SQLProcedureColumns: 返回指定存储过程的参数信息的列表。
- SQLProcedures: 返回指定数据源的存储过程信息的列表。
- SQLSpecialColumns: 返回唯一确定某行的列的信息, 或者当某事务修改行时自动更新各列的信息。
- SQLStatistics: 返回单表的相关统计信息和索引信息。
- SQLTablePrivileges: 返回相关各表的名称以及相关的权限信息。
- SQLTables: 返回指定数据源中表信息。

### 4.1.9 终止语句执行

下列函数用于终止语句执行:

- SQLFreeStmt: 终止语句执行, 关闭所有相关的游标, 放弃没有提交的结果, 选择释放

与指定语句句柄相关的资源。

- SQLCloseCursor: 关闭一个打开的游标, 放弃没有提交的结果。
- SQLCancel: 放弃执行一条 SQL 语句。
- SQLEndTran: 提交或者回滚事务。

### 4.1.10 中断连接

下列函数处理中断连接的任务:

- SQLDisconnect: 关闭指定连接。
- SQLFreeHandle: 释放环境、连接、语句或者描述符句柄。

## 4.2 虚谷数据库 ODBC 连接使用

### 4.2.1 申请环境句柄与连接句柄

应用程序要和远程的数据库服务端进行通讯, 须与数据库服务端建立连接。为建立 ODBC 数据源连接, 需要使用环境句柄以及连接句柄。句柄有层次的概念, 连接句柄总是和唯一的环境句柄相联系, 所有的连接句柄必须在环境句柄释放之前释放。

应用程序可通过调用函数 SQLAllocHandle 申请环境句柄, 调用函数 SQLAllocHandle 时必传入句柄类型 SQL\_HANDLE\_ENV, 申请环境句柄成功后, 可在此环境句柄上申请连接句柄。申请环境句柄和连接句柄的示例代码如下:

```
#include <windows.h>
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
/* 检测返回代码是否为成功标志, 当为成功标志时返回 TRUE, 否则返回 FALSE */
#define RC_SUCCESSFUL(rc) ((rc) == SQL_SUCCESS || (rc) == SQL_SUCCESS_WITH_INFO)
/* 检测返回代码是否为失败标志, 当为失败标志时返回 TRUE, 否则返回 FALSE */
#define RC_NOTSUCCESSFUL(rc) (!(RC_SUCCESSFUL(rc)))
void main(void)
{
    SQLHENV henv; /* 环境句柄 */
    SQLHDBC hdbc; /* 连接句柄 */
    SQLRETURN ret; /* 返回代码 */
    /* 申请一个环境句柄 */
    ret = SQLAllocHandle(SQL_HANDLE_ENV, NULL, &henv);
    if(RC_NOTSUCCESSFUL(ret))
    {
        /* 环境句柄分配失败 */
        return;
    }
}
```



```
}
/* 设置环境句柄的 ODBC 版本 */
ret = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
    SQL_OV_ODBC3, SQL_IS_INTEGER);
/* 申请一个连接句柄 */
ret = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
if(RC_NOTSUCCESSFUL(ret))
{
    /* 连接句柄分配失败 */
    SQLFreeHandle(SQL_HANDLE_ENV, henv);
    return;
}
/* 释放连接句柄 */
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
/* 释放环境句柄 */
SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

## 4.2.2 如何与数据源进行连接

ODBC 连接从数据源开始，数据源是 ODBC 对一个特定的数据库连接的别称。为访问由数据源提供的数据库，应用程序须先建立和数据源之间的连接，在环境句柄和连接句柄正确分配后，才能通过连接访问并管理数据。

为产生建立连接时所需的参数，须完成以下几项工作：

- 调用 SQLAllocHandle 申请一个环境句柄。
- 调用 SQLAllocHandle 申请一个连接句柄。
- 创建一个数据源 DSN。
- 一个有效的用户名。
- 与用户名匹配的密码。
- 提供给驱动程序的其它参数信息。

连接 ODBC 数据源时，ODBC 提供三种不同的连接函数：

- SQLConnect。
- SQLDriverConnect。
- SQLBrowseConnect。

各函数都有不同的参数以及不同级别的一致性，如下表所示：

表 4-1 连接到数据源的 ODBC 函数

函数	ODBC 版本	一致性	主要参数
SQLConnect	1.0	核心级	hdbc, 数据源, 用户 ID, 口令
SQLDriverConnect	1.0	1 级	hdbc, 窗口句柄, 输入连接字符串
SQLBrowseConnect	1.0	2 级	hdbc, 输入连接字符串, 输出连接字符串

SQLConnect 是连接 ODBC 数据源的基本方法，支持最高一致性级别。

SQLConnect 函数有以下参数：连接句柄、数据源名称、数据源名称长度、用户名（用户 ID）、用户名长度、密码以及密码长度。

SQLConnect 函数返回代码有：SQL\_SUCCESS，SQL\_SUCCESS\_WITH\_INFO，SQL\_ERROR 或 SQL\_INVALID\_HANDLE。

使用 SQLConnect 函数连接数据源的示例代码如下：

```
ret = SQLConnect(hdbc, (SQLCHAR *) "XGDB", SQL_NTS,
(SQLCHAR *) "SYSDBA", SQL_NTS,
(SQLCHAR *) "SYSDBA", SQL_NTS);
if (RC_NOTSUCCESSFUL(ret))
{
/* 连接数据源失败! */
... 错误处理 ...
}
```

SQLDriverConnect 提供了比 SQLConnect 更灵活的方法来建立 ODBC 连接。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未在系统信息表中定义的数据源。

SQLDriverConnect 提供以下连接方法：

- 用连接字符串建立连接，此字符串包括建立连接的所有连接属性数据，如 DSN，用户名和密码，以及其他的数据库所需要的连接信息。
- 用一个并不完整的连接字符串尝试建立连接，使 ODBC 驱动程序管理器提示用户输入所需要连接属性信息。
- 用未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。

- 用特定连接字符串建立连接，此字符串在 DSN 配置文件中确定。

SQLDriverConnect 函数 fDriverCompletion 参数说明：

- SQL\_DRIVER\_PROMPT：此选项用于显示一个对话框以提示用户输入连接信息。
- SQL\_DRIVER\_COMPLETE：如函数调用中包含的连接属性信息足够，ODBC 尝试进行连接，否则弹出对话框提示用户输入连接信息，此时效果等同于 SQL\_DRIVER\_PROMPT。
- SQL\_DRIVER\_COMPLETE\_REQUIRED：此参数属性与 SQL\_DRIVER\_COMPLETE 参数相似，唯一不同是用户不能改变由函数提供的信息
- SQL\_DRIVER\_NOPROMPT：如函数调用时连接属性信息足够，ODBC 进行连接，否则返回 SQL\_ERROR。

使用 SQLDriverConnect 连接数据源的示例代码如下：

```
SQLCHAR szConnStrIn[256] = "DSN=XGDB;DATABASE=SYSTEM;UID=SYSDBA;PWD=SYSDBA;PORT=5138";
SQLCHAR szConnStrOut[256];
SQLSMALLINT cbConnStrOut;
ret = SQLDriverConnect(hdbc, NULL, szConnStrIn, SQL_NTS,
    szConnStrOut, 256, &cbConnStrOut, SQL_DRIVER_NOPROMPT);
if (RC_NOTSUCCESSFUL(ret))
{
    /* 连接数据源失败! */
    ... 错误处理 ...
}
```

SQLBrowseConnect 函数与 SQLDriverConnect 函数相似，在调用 SQLBrowseConnect 函数情况下，程序运行时会输出一个连接字符串，可用交互方式决定连接到数据源时需要部分信息。

使用 SQLBrowseConnect 函数连接数据源的示例代码如下：

```
SQLCHAR szConnStrIn[256] = "";
SQLCHAR szConnStrOut[256];
SQLSMALLINT cbConnStrOut;
strcpy(szConnStrIn, "DRIVER=XuguSQL 11.2");
ret = SQLBrowseConnect(hdbc, szConnStrIn, SQL_NTS, szConnStrOut,
    256, &cbConnStrOut);
if (ret != SQL_NEED_DATA)
{
    /* 连接数据源失败! */
    ... 错误处理 ...
}
strcpy(szConnStrIn, "SERVER=127.0.0.1;PORT=5138");//PORT=5138 可选
ret = SQLBrowseConnect(hdbc, szConnStrIn, SQL_NTS, szConnStrOut,
    256, &cbConnStrOut);
if (ret != SQL_NEED_DATA)
{
```

```
/* 连接数据源失败! */  
... 错误处理 ...  
}  
strcpy(szConnStrIn, "UID=SYSDBA;PWD=SYSDBA;");  
ret = SQLBrowseConnect(hdbc, szConnStrIn, SQL_NTS, szConnStrOut  
    , 256, &cbConnStrOut);  
if (ret != SQL_SUCCESS)  
{  
/* 连接数据源失败! */  
... 错误处理 ...  
}  
/* 连接成功 */
```

### 4.2.3 编写连接字符串

ODBC 连接字符串由形如 key=value 的键值对构成，多个键值对以英文分号 (;) 分隔。虚谷 ODBC 连接串支持的参数见本文档第四章注册数据源参数说明，此表格中列举了所有驱动支持的属性名及取值。若在连接串中填写了除此之外的属性，驱动将其忽略。若部分属性未填写，驱动将使用默认值。连接串中的属性除了传递到驱动中以实现差异性的功能外，还有部分属性在连接数据库前发挥重要作用，它们分别是 DSN、Driver 和 FileDSN。驱动管理器管理各种 ODBC 驱动，驱动管理器正是通过上述三个属性实现了应用程序和对应驱动的映射。所以连接串可以有三种写法，如下所示：

- DSN=XGDB;UID= SYSDBA;PWD= SYSDBA
- Driver=XuguSQL 11.2;Server=127.0.0.1;Port=5138;UID=SYSDBA;PWD=SYSDBA
- FileDSN=C:\_XGDB.dsn

使用 DSN 属性则需要先配置 DSN，DSN 中包含驱动名等其它所有属性。若 DSN 中属性完整，则连接串仅需 DSN=XGDB 即可，若此时连接串中提供了其它驱动支持的属性，则驱动使用连接串中提供的属性。

使用 Driver 属性则无需配置 DSN，Driver 指定驱动名，所有需要的属性均需填写至连接串。

使用 FileDSN 则需要先配置文件 DSN，文件 DSN 与 DSN 类似，文件 DSN 中包含驱动名等其它所有属性。若文件 DSN 中属性完整，则连接串仅需 FileDSN=C:\_XGDB.dsn 即可，若此时连接串中提供了其它驱动支持的属性，则驱动使用连接串中提供的属性。

 说明

虚谷 ODBC 驱动为 `SQLDriverConnect` 提供了配置文件，配置文件由换行符分隔的形如 `key=value` 的键值对构成。配置文件支持的属性与连接串完全相同。配置文件中的属性拥有最高的优先级，若配置文件存在，`SQLDriverConnect` 将使用配置文件中提供的属性值。在 Windows 下，驱动在 C:\_ 和 D:\_ 搜索 `xgodbc.cfg`；在 Linux 下，驱动在用户目录搜索 `.xgodbc.cfg`。

### 4.2.4 设置与取得连接属性

建立连接后，应用程序可通过调用 `SQLSetConnectAttr` 函数设置连接属性，对连接进行管理。一些常用的连接属性如下表所示。

表 4-2 虚谷 JDBC 实现接口类

属性	描述
<code>SQL_ATTR_ACCESS_MODE</code>	用来设置访问模式，即只读或者读写连接模式，可以用来优化并发控制策略。当前版本不支持该功能
<code>SQL_ATTR_ASYNC_ENABLE</code>	是否支持异步执行
<code>SQL_ATTR_AUTOCOMMIT</code>	是否使用自动提交功能
<code>SQL_ATTR_CONNECTION_TIMEOUT</code>	设置连接上的超时
<code>SQL_ATTR_CURRENT_CATALOG</code>	当前连接使用的编目
<code>SQL_ATTR_LOGIN_TIMEOUT</code>	设定登录超时
<code>SQL_ATTR_ODBC_CURSORS</code>	设置驱动程序管理器使用游标的方式
<code>SQL_ATTR_PACKET_SIZE</code>	设置网络传输包的大小
<code>SQL_ATTR_QUIET_MODE</code>	使弹出对话框有效/无效

更多连接属性请参见《Microsoft ODBC 3.0 程序员参考手册》。

应用程序可通过调用 `SQLGetConnectAttr` 函数取得当前连接属性。

设置与取得连接属性的示例代码如下：

```
SQLINTEGER autocommit_mode;
/* 设置连接句柄属性, 关闭自动提交功能 */
SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)
    SQL_AUTOCOMMIT_OFF, SQL_IS_INTEGER);
/* 取得连接句柄属性, 取得提交的模式 */
SQLGetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER) &
    autocommit_mode, sizeof(SQLINTEGER), NULL);
```

## 4.2.5 数据库连接断开与释放

在应用完成对数据库的操作后, 需进行断开连接和释放资源操作。如连接使用后不与服务端断开, 则会影响其它新建连接服务。如客户端申请资源不释放则会影响应用后续资源申请。连接的释放对于应用的长期稳定运行有极重要的作用。

频繁的建连接, 旧资源得不到释放, 将极大地影响后续应用的使用体验。连接的释放主要分为以下步骤:

- 在有语句句柄的情况下先释放语句句柄
- 断开连接
- 释放连接句柄
- 释放环境句柄

操作流程图如下图所示。此段程序的 C 代码示例如下:

```
#include <windows.h>
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
/* 检测返回代码是否为成功标志, 当为成功标志返回 TRUE, 否则返回
    FALSE */
#define RC_SUCCESSFUL(rc) ((rc) == SQL_SUCCESS || (rc) ==
    SQL_SUCCESS_WITH_INFO)
/* 检测返回代码是否为失败标志, 当为失败标志返回 TRUE, 否则返回
    FALSE */
#define RC_NOTSUCCESSFUL(rc) (!(RC_SUCCESSFUL(rc)))

void main(void)
{
    SQLHENV henv;    /* 环境句柄 */
    SQLHDBC hdbc;    /* 连接句柄 */
    SQLHSTMT hstmt; /* 语句句柄 */
    SQLRETURN ret;   /* 返回代码 */
    SQLINTEGER autocommit_mode;

    /* 申请一个环境句柄 */
    SQLAllocHandle(SQL_HANDLE_ENV, NULL, &henv);
    /* 设置环境句柄的 ODBC 版本 */
    SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3
        , SQL_IS_INTEGER);
```

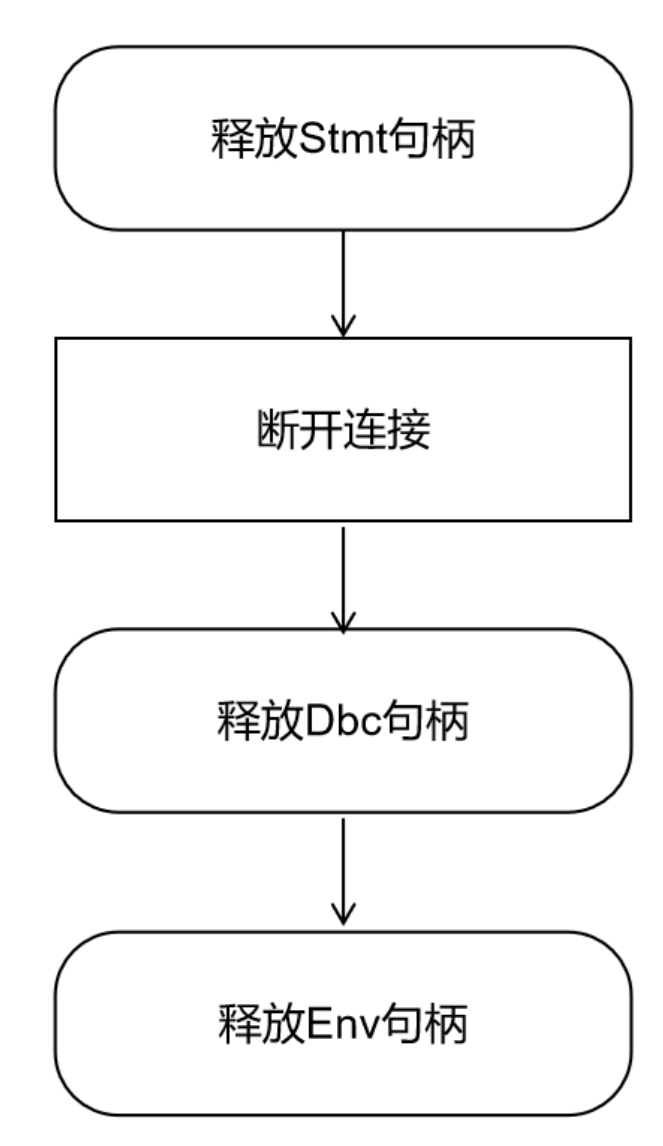


图 4-1 连接释放流程图

```
/* 申请一个连接句柄 */
SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
ret = SQLConnect(hdbc, (SQLCHAR *)"XGDB", SQL_NTS, (SQLCHAR *)"
    SYSDBA", SQL_NTS, (SQLCHAR *)"SYSDBA", SQL_NTS);
if (RC_NOTSUCCESSFUL(ret)) {
/* 连接数据源失败! */
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return;
}
/* 设置连接句柄属性, 关闭自动提交功能 */
SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)
    SQL_AUTOCOMMIT_OFF, SQL_IS_INTEGER);
/* 取得连接句柄属性, 取得提交的模式 */
SQLGetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)&
    autocommit_mode, sizeof(SQLINTEGER), NULL);
/* 申请一个语句句柄 */
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
...在这里可以使用语句句柄进行相应的数据库操作...
/* 释放语句句柄 */
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
/* 提交连接上的事务 */
SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
/* 断开与数据源之间的连接 */
SQLDisconnect(hdbc);
/* 释放连接句柄 */
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
/* 释放环境句柄 */
SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

## 4.3 虚谷数据库 ODBC 应用编程

应用程序可以使用 ODBC 访问数据源。

### 操作步骤

- 调用函数 SQLAllocHandle 申请环境句柄、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。
- 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、SQLBindParameter 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。



- 应用程序可以通过调用 ODBC 目录函数 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。
- 如果连接属性自动提交选项设置为手动提交状态，应用程序需调用函数 SQLEndTran 来提交或回滚事务，进行相关的事务处理。
- 调用函数 SQLFreeHandle 来释放申请的语句句柄。
- 调用函数 SQLDisconnect 来断开应用程序与数据源之间的连接。
- 调用函数 SQLFreeHandle 来释放申请的连接句柄、环境句柄。

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。详细信息请参见 ODBC 的示例说明章节。

## 4.4 使用存储过程和函数

### 4.4.1 存储过程和函数字典信息获取

虚谷数据库 ODBC 支持字典函数 SQLProcedures 的调用，用户可调用此函数获取虚谷数据库存储过程与函数的字典信息。

调用方法如下：

```
SQLProcedures(hstmt, (SQLCHAR*)"SYSTEM", SQL_NTS, (SQLCHAR*)"SYSDBA", SQL_NTS, (SQLCHAR*)"TEST_PROC", SQL_NTS);
```

返回字典信息格式如下表所示。

表 4-3 SQLProcedures 字典信息说明表

字典项	相关说明
PROCEDURE_CAT	存储模块编目信息
PROCEDURE_SCHEM	存储模块模式信息
PROCEDURE_NAME	存储模块名
PROCEDURE_TYPE	存储模块类型

虚谷数据库 ODBC 支持字典函数 SQLProcedureColumns 的调用，用于返回存储模块的参数信息。调用方法如下：

```
SQLProcedureColumns (hstmt, (SQLCHAR*) "SYSTEM", SQL_NTS, (SQLCHAR*) "
    SYSDBA", SQL_NTS, (SQLCHAR*) "TEST_PROC", SQL_NTS, NULL, 0);
```

返回字典信息格式如表 2 所示：

**表 4-4** SQLProcedureColumns 字典信息说明表

字典项	相关说明
PROCEDURE_CAT	存储模块编目信息
PROCEDURE_SCHEM	存储模块模式信息
PROCEDURE_NAME	存储模块名
COLUMN_NAME	过程列名称。驱动程序为没有名称的过程列返回空字符串
COLUMN_TYPE	参数类型，即输入参数还是输出参数
DATA_TYPE	参数的 SQL 数据类型
TYPE_NAME	参数的类型名
COLUMN_SIZE	参数的精度
BUFFER_LENGTH	参数所占的字符长度
DECIMAL_DIGITS	参数的刻度
NUM_PREC_RADIX	仅对数值类型有效，仅为 10 或者 2，如果为 10 表示为精确数字，如果为 2 表示为非精确数字
NULLABLE	参数是否接受空值标志
REMARK	参数说明
COLUMN_DEF	参数的缺省值
SQL_DATA_TYPE	参数的 SQL 数据类型

字典项	相关说明
SQL_DATETIME_SUB	日期时间类型或时间间隔类型的子代码
CHAR_OCTET_LENGTH	字符数据类型以字节计算的最大长度
ORDINAL_POSITION	参数的顺序
IS_NULLABLE	参数是否包含空值

## 4.4.2 存储模块创建

用户可使用 SQLExecDirect 函数执行创建存储模块的 SQL 语句创建存储模块，如下所示：

```
SQLExecDirect(hstmt, (SQLCHAR *) "create or replace procedure test\
    _proc (c1 in int) as declare"
"c2 int"
"begin"
"c2 := c1 + 100;"
"end;", SQL\ _NTS);
```

## 4.4.3 存储模块调用

调用存储模块分两种方式：

- 直接调用如存储过程需要设置参数，在调用存储模块时，已为所有的 IN、INOUT 类型的参数进行了赋值，带有 OUT 属性的参数值通过取得存储过程结果集的方法获取。立即执行存储过程的示例如下：

```
SQLExecDirect(hstmt, (SQLCHAR*) "call test_proc(123);",
SQL _NTS);
```

- 参数调用本调用方法指在调用模块时，其参数值用问号占位代替，发送给数据库服务端后，服务端返回参数的准备信息，用户依据服务端返回的参数描述信息进行参数绑定，然后执行。其参数的处理方法与普通的参数处理方法相同。

## 4.4.4 存储函数使用示例

```
create or replace function GetName(uid int) return char
as
declare
username char;
begin
select name into username from mytable where id = uid;
return username;
end;
```

```
#include <windows.h>
#include <stdio.h>
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>

void main(void)
{
    SQLHENV henv;      /* 环境句柄 */
    SQLHDBC hdbc;     /* 连接句柄 */
    SQLHSTMT hstmt;   /* 语句句柄 */
    SQLRETURN ret;    /* 返回代码 */

    /* 申请一个环境句柄 */
    SQLAllocHandle(SQL_HANDLE_ENV, NULL, &henv);
    /* 设置环境句柄的 ODBC 版本 */
    SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3
        , SQL_IS_INTEGER);
    /* 申请一个连接句柄 */
    SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
    ret = SQLConnect(hdbc, (SQLCHAR*)"XGDB", SQL_NTS, (SQLCHAR*)"SYSDBA
        ", SQL_NTS, (SQLCHAR*)"SYSDBA", SQL_NTS);
    /* 申请一个语句句柄 */
    SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

    strcpy(sql, "GetName(?)");
    int id = 100;
    char name[20] = {};
    SQLLEN len1 = 4;
    SQLLEN len2 = 20;
    /* 绑定参数 */
    ret = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
        SQL_INTEGER, 0, 0, &id, 4, &len1);
    /* 绑定函数返回值 */
    ret = SQLBindParameter(hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR,
        SQL_CHAR, 0, 0, name, 20, &len2);
    /* 执行函数 */
    ret = SQLExecDirect(hstmt, (SQLCHAR*)sql, SQL_NTS);

    printf("name: %s\n", name);

    /* 释放语句句柄 */
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    /* 提交连接上的事务 */
    SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
    /* 断开与数据源之间的连接 */
    SQLDisconnect(hdbc);
    /* 释放连接句柄 */
    SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
    /* 释放环境句柄 */
    SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

## 4.5 虚谷数据库 ODBC 常用 API 介绍

### 4.5.1 SQLAllocHandle 函数

#### 功能

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 SQLAllocHandle(  
SQLSMALLINT HandleType,  
SQLHANDLE InputHandle,  
SQLHANDLE * OutputHandlePtr);
```

#### 参数解释

- HandleType: 句柄类型。
- InputHandle: 输入句柄类型，通常为申请句柄类型的父句柄。
- OutputHandlePtr: 申请的目标句柄（输出句柄），申请句柄成功后，此参数返回申请的目标句柄地址。

#### 返回值

成功时返回 SQL\_SUCCESS。有异常时返回 SQL\_SUCCESS\_WITH\_INFO, SQL\_INVALID\_HANDLE 或 SQL\_ERROR。

### 4.5.2 SQLConnect 函数

#### 功能

连接客户端和服务端，生成会话。

#### 函数原型

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

#### 参数解释

- ConnectionHandle: 连接句柄。
- ServerName: ODBC 配置的 DSN，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，可根据 DSN

名长度自动匹配长度。

- **UserName**: 用户名, 为字符串格式, 连接数据库的身份权限标识。
- **NameLength2**: 以字节为单位描述用户名长度, 可选填默认值 `SQL_NTS` 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**: 以字节为单位描述用户口令长度, 可选填默认值 `SQL_NTS` 根据用户口令长度自动匹配长度。

## 返回值

成功时返回 `SQL_SUCCESS`。

## 4.5.3 SQLDriverConnect 函数

### 功能

根据用户的需求, 设置连接属性, 新建连接。

### 函数原型

```
SQLRETURN SQLDriverConnect(  
SQLHDBC          ConnectionHandle,  
SQLHWND          WindowHandle,  
SQLCHAR *        InConnectionString,  
SQLSMALLINT      StringLength1,  
SQLCHAR *        OutConnectionString,  
SQLSMALLINT      BufferLength,  
SQLSMALLINT *    StringLength2Ptr,  
SQLUSMALLINT     DriverCompletion);
```

### 参数解释

- **ConnectionHandle**: 连接句柄。
- **WindowHandle**: 窗口句柄。
- **InConnectionString**: 连接属性串, 用户可选择 DSN, 并在此设置更改需要的连接参数属性。
- **StringLength1**: 以字节为单位描述连接串属性长度, 可以选填默认值 `SQL_NTS`。
- **OutConnectionString**: 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。
- **BufferLength**: 输出连接串的最大长度。
- **StringLength2Ptr**: 以字节为单位描述连接输出信息串长度, 可选填默认值 `SQL_NTS`。
- **DriverCompletion**: 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

成功时返回 SQL\_SUCCESS。

## 4.5.4 SQLBrowseConnect 函数

### 功能

根据用户的需求，设置连接属性，新建连接。

### 函数原型

```
SQLRETURN SQLBrowseConnect (
SQLHDBC      ConnectionHandle,
SQLCHAR *    InConnectionString,
SQLSMALLINT  StringLength1,
SQLCHAR *    OutConnectionString,
SQLSMALLINT  BufferLength,
SQLSMALLINT * StringLength2Ptr);
```

### 参数解释

- ConnectionHandle: 连接句柄。
- InConnectionString: 连接属性串。
- StringLength1: 以字节为单位描述连接串属性长度，可选填默认值 SQL\_NTS。
- OutConnectionString: 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。
- BufferLength: 输出连接串的最大长度。
- StringLength2Ptr: 以字节为单位描述连接输出信息串长度，可选填默认值 SQL\_NTS。

## 返回值

成功时返回 SQL\_SUCCESS。

## 4.5.5 SQLPrepare 函数

### 功能

准备一个 SQL 语句，生成执行计划。

### 函数原型

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

### 参数解释

- StatementHandle: 语句句柄。

- StatementText: SQL 语句字符串。
- TextLength: 以字节为单位描述语句长度。

### 返回值

成功时返回 SQL\_SUCCESS。

## 4.5.6 SQLBindParameter 函数

### 功能

绑定执行 SQL 时需要的参数。

### 函数原型

```
SQLRETURN SQLBindParameter(  
SQLHSTMT          StatementHandle,  
SQLUSMALLINT      ParameterNumber,  
SQLSMALLINT       InputOutputType,  
SQLSMALLINT       ValueType,  
SQLSMALLINT       ParameterType,  
SQLULEN           ColumnSize,  
SQLSMALLINT       DecimalDigits,  
SQLPOINTER        ParameterValuePtr,  
SQLLEN            BufferLength,  
SQLLEN *          StrLen_or_IndPtr);
```

### 参数解释

- StatementHandle: 语句句柄。
- ParameterNumber: 参数序号（从 1 开始）。
- InputOutputType: 参数 In/Out 类型。
- ValueType: 参数 C 类型。
- ParameterType: 参数对应的 sqltype 类型。
- ColumnSize: 参数精度（string 类型时参数的长度大小）。
- DecimalDigits: 数值 numeric 时参数的标度。
- ParameterValuePtr: 参数值指针（引用地址）。
- BufferLength: 参数字节宽度。
- StrLen\_or\_IndPtr: 参数的实际长度，指针值。

### 返回值

成功时返回 SQL\_SUCCESS。

## 4.5.7 SQLBindCol 函数

### 功能



将 C 数据类型变量绑定到结果集的列。

### 函数原型

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

### 参数解释

- StatementHandle: 语句句柄。
- ColumnNumber: 列序号（从 1 开始）。
- TargetType: 列的 C 类型。
- TargetValuePtr: 接收列数据的缓冲区指针。
- BufferLength: 列宽度，以字节为单位限制 string 类型的长度。
- StrLen\_or\_Ind: 实际列数据大小。

### 返回值

成功时返回 SQL\_SUCCESS。

## 4.5.8 SQLColAttribute 函数

### 功能

本函数用于输出列绑定之前，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，辅助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,  
SQLLEN *          NumericAttributePtr);
```

### 参数解释

- StatementHandle: 语句句柄。
- ColumnNumber: 列序号（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 指定输出字符串的 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。

### 返回值

成功时返回 SQL\_SUCCESS。

## 4.5.9 SQLColAttribute 函数

### 功能

此函数执行 SQLPrepare 准备的语句。

### 函数原型

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

### 参数解释

StatementHandle: 语句句柄。成功时返回 SQL\_SUCCESS。

## 4.5.10 SQLExecuteDirect 函数

### 功能

直接执行 SQL 语句到服务端, 无需 prepare 的准备过程。

### 函数原型

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

### 参数解释

- StatementHandle: 语句句柄。
- StatementText: SQL 语句, 字符串类型。
- TextLength: 以字节为单位描述 SQL 字符串的长度, 可以选填默认值 SQL\_NTS。

### 返回值

成功时返回 SQL\_SUCCESS。

### 4.5.11 SQLFetch 函数

#### 功能

对 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 SQLFetch(  
SQLHSTMT      StatementHandle);
```

#### 参数解释

StatementHandle: 语句句柄。

#### 返回值

成功时返回 SQL\_SUCCESS，失败时返回 SQL\_ERROR。

### 4.5.12 SQLTables 函数

#### 功能

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);
```

#### 参数解释

- StatementHandle: 语句句柄。
- CatalogName: 数据库名，char 类型的字符串。当值为 NULL 时表示所有。
- NameLength1: 以字节为单位描述参数 CatalogName 字符串的长度。
- SchemaName: 模式名，char 类型的字符串。当值为 NULL 时表示所有。
- NameLength2: 以字节为单位描述参数 SchemaName 字符串的长度。
- TableName: 表名，char 类型的字符串。当值为 NULL 时表示所有。
- NameLength3: 以字节为单位描述参数 TableName 字符串的长度。

- **TableType**: 要匹配的表类型。比如: “table”, “view” 等。
- **NameLength4**: 以字节为单位描述参数 **TableType** 字符串的长度。

### 返回值

成功时返回 **SQL\_SUCCESS**, 失败时返回 **SQL\_ERROR**。

## 4.5.13 SQLColumns 函数

### 功能

返回指定表中的列名列表。

### 函数原型

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

### 参数解释

- **StatementHandle**: 语句句柄。
- **CatalogName**: 数据库名, char 类型的字符串。当值为 **NULL** 时表示所有。
- **NameLength1**: 以字节为单位描述参数 **CatalogName** 字符串的长度。
- **SchemaName**: 模式名, char 类型的字符串。当值为 **NULL** 时表示所有。
- **NameLength2**: 以字节为单位描述参数 **SchemaName** 字符串的长度。
- **TableName**: 表名, char 类型的字符串。当值为 **NULL** 时表示所有。
- **NameLength3**: 以字节为单位描述参数 **TableName** 字符串的长度。
- **ColumnName**: 列名, char 类型的字符串。
- 以字节为单位描述参数 **ColumnName** 字符串的长度。

### 返回值

成功时返回 **SQL\_SUCCESS**, 失败时返回 **SQL\_ERROR**。

## 4.5.14 SQLPrimaryKeys 函数

### 功能

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);
```

## 参数解释

- StatementHandle: 语句句柄。
- CatalogName: 数据库名, char 类型的字符串。
- NameLength1: 以字节为单位描述参数 CatalogName 字符串的长度。
- SchemaName: 模式名, char 类型的字符串。
- NameLength2: 以字节为单位描述参数 SchemaName 字符串的长度。
- TableName: 表名, char 类型的字符串。
- NameLength3: 以字节为单位描述参数 TableName 字符串的长度。

## 返回值

成功时返回 SQL\_SUCCESS, 失败时返回 SQL\_ERROR。

## 4.5.15 SQLForeignKeys

### 功能

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);
```

## 参数解释

- StatementHandle: 语句句柄。

- PKCatalogName: 外键所引用的主键列所在表的数据库名。
- NameLength1: 以字节为单位描述参数 PKCatalogName 字符串的长度。
- PKSchemaName: 外键所引用的主键列所在表的模式名。
- NameLength2 : 以字节为单位描述参数 PKSchemaName 字符串的长度。
- PKTableName: 外键所引用的主键列所在表的表名。
- NameLength3: 以字节为单位描述参数 PKTableName 字符串的长度。
- FKCatalogName: 外键列所在表的数据库名。
- NameLength4: 以字节为单位描述参数 FKCatalogName 字符串的长度。
- FKSchemaName: 外键列所在表的模式名。
- NameLength5: 以字节为单位描述参数 FKSchemaName 字符串的长度。
- FKTableName: 外键列所在表的表名。
- NameLength6: 以字节为单位描述参数 FKTableName 字符串的长度。

### 返回值

成功时返回 SQL\_SUCCESS, 失败时返回 SQL\_ERROR。

## 4.5.16 SQLProcedures 函数

### 功能

用户可指定数据库名, 模式名, 存储过程名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);
```

### 参数解释

- StatementHandle: 语句句柄。
- CatalogName: 数据库名, char 类型的字符串。
- NameLength1: 以字节为单位描述参数 CatalogName 字符串的长度。
- SchemaName: 模式名, char 类型的字符串。
- NameLength2: 以字节为单位描述参数 SchemaName 字符串的长度。
- ProcName: 存储过程名, char 类型的字符串。

- NameLength3: 以字节为单位描述参数 ProcName 字符串的长度。

### 返回值

成功时返回 SQL\_SUCCESS, 失败时返回 SQL\_ERROR。

## 4.5.17 SQLStatics 函数

### 功能

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);
```

### 参数解释

- StatementHandle: 语句句柄。
- CatalogName: 数据库名, char 类型的字符串。
- NameLength1: 以字节为单位描述参数 CatalogName 字符串的长度。
- SchemaName: 模式名, char 类型的字符串。
- NameLength2: 以字节为单位描述参数 SchemaName 字符串的长度。
- TableName: 表名, char 类型的字符串。
- NameLength3: 以字节为单位描述参数 TableName 字符串的长度。
- Unique: 是否指定唯一值索引。可填值为: SQL\_INDEX\_UNIQUE、SQL\_INDEX\_AL。
- 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。

### 返回值

成功时返回 SQL\_SUCCESS。

## 4.5.18 SQLPutData 函数

### 功能

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,  
SQLLEN        StrLen_or_Ind);
```

## 参数解释

- StatementHandle: 语句句柄。
- DataPtr: 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: 以字节为单位描述参数 DataPtr 数据值的长度。

## 返回值

成功时返回 SQL\_SUCCESS, 失败时返回 SQL\_ERROR。

## 4.5.19 SQLGetData 函数

### 功能

用于返回结果行的单独列数据, 当返回的目标数据较大时, 可拆分部分多次调用返回。

## 函数原型

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

## 参数解释

- StatementHandle: 语句句柄。
- Col\_or\_Param\_Num: 列序号 (从 1 开始)。
- TargetType: C 数据类型标识。
- TargetValuePtr: 指向一块数据返回的 buffer。
- BufferLength: 以字节为单位描述数据返回的 buffer 的长度大小, 当返回数据大于此长度时, 将发生截断。

## 返回值

成功时返回 SQL\_SUCCESS, 失败时返回 SQL\_ERROR。

## 4.5.20 SQLMoreResults 函数

### 功能



判断是否有多结果集，如果有则提供初始化操作。

### 函数原型

```
SQLRETURN SQLMoreResults(  
SQLHSTMT StatementHandle);
```

### 参数解释

- StatementHandle: 语句句柄。

### 返回值

成功时返回 SQL\_SUCCESS，失败时返回 SQL\_ERROR。

# 5 示例说明

## 5.1 C/C++

### 5.1.1 执行 Select 语句示例代码

```
void main()
{
char szDSN[] = "XGDB";
char szUID[] = "SYSDBA";
char szAuth[] = "SYSDBA";
SQLHENV henv = NULL;
SQLHDBC hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN ret = 0;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
ret = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
    SQL_OV_ODBC3, SQL_IS_INTEGER);
ret = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
ret = SQLConnect(hdbc, (SQLCHAR*)szDSN, SQL_NTS, (SQLCHAR*)szUID,
    SQL_NTS, (SQLCHAR*)szAuth, SQL_NTS);
if (ret != SQL_SUCCESS)
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, 1, sqlState, &nativeErr, errMsg
    , 200, &len);
printf("Can't connect to server, reason: %s\n", errMsg);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return;
}
printf("Connect successful!\n");
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
char sql[] = "select * from mytable";
ret = SQLExecDirect(hstmt, (SQLCHAR*)sql, SQL_NTS);
if (ret != SQL_SUCCESS)
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, sqlState, &nativeErr,
    errMsg, 200, &len);
printf("SQLExecDirect failed, reason: %s\n", errMsg);
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

```

return;
}
SQLLEN cbLen1, cbLen2, cbLen3, cbLen4, cbLen5;
char col1[30] = { 0 };
char col2[30] = { 0 };
char col3[30] = { 0 };
char col4[30] = { 0 };
char col5[30] = { 0 };
SQLBindCol(hstmt, 1, SQL_C_CHAR, col1, 30, &cbLen1);
SQLBindCol(hstmt, 2, SQL_C_CHAR, col2, 30, &cbLen2);
SQLBindCol(hstmt, 3, SQL_C_CHAR, col3, 30, &cbLen3);
SQLBindCol(hstmt, 4, SQL_C_CHAR, col4, 30, &cbLen4);
SQLBindCol(hstmt, 5, SQL_C_CHAR, col5, 30, &cbLen5);
while (ret == SQL_SUCCESS)
{
ret = SQLFetch(hstmt);
if (ret == SQL_ERROR || ret == SQL_NO_DATA)
break;
printf("%s, %s, %s, %s, %s\n", col1, col2, col3, col4, col5);
}
ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);

ret = SQLDisconnect(hdbc);
ret = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
ret = SQLFreeHandle(SQL_HANDLE_ENV, henv);
}

```

## 5.1.2 执行 Insert 语句示例代码

```

void main()
{
char szDSN[] = "XGDB";
char szUID[] = "SYSDBA";
char szAuth[] = "SYSDBA";
SQLHENV henv = NULL;
SQLHDBC hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN ret = 0;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
ret = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
SQL_OV_ODBC3, SQL_IS_INTEGER);
ret = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
ret = SQLConnect(hdbc, (SQLCHAR*)szDSN, SQL_NTS, (SQLCHAR*)szUID,
SQL_NTS, (SQLCHAR*)szAuth, SQL_NTS);
if (ret != SQL_SUCCESS)
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, 1, sqlState, &nativeErr, errMsg
, 200, &len);
printf("Can't connect to server, reason: %s\n", errMsg);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
}

```

```

return;
}
printf("Connect successful!\n");
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
char sql[] = "insert into mytable(c1,c2,c3,c4,c5) values(?,?,?,?);";
ret = SQLPrepare(hstmt, (SQLCHAR*)sql, SQL_NTS);
if (ret != SQL_SUCCESS)
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, sqlState, &nativeErr,
    errMsg, 200, &len);
printf("SQLPrepare failed, reason: %s\n", errMsg);
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return;
}
SQLLEN cbLen1, cbLen2, cbLen3, cbLen4, cbLen5;
DATE_STRUCT date;
TIME_STRUCT time;
TIMESTAMP_STRUCT dt;
date.year = 2022;
date.month = 5;
date.day = 1;
time.hour = 11;
time.minute = 12;
time.second = 13;
dt.year = 2022;
dt.month = 5;
dt.day = 1;
dt.hour = 11;
dt.minute = 12;
dt.second = 13;
char timeWTZ[] = "10:11:12.0 +08:00";
char dtWTZ[] = "1990-01-01 01:02:03 +08:00";
cbLen1 = sizeof(DATE_STRUCT);
cbLen2 = sizeof(TIME_STRUCT);
cbLen3 = sizeof(TIMESTAMP_STRUCT);
cbLen4 = strlen(timeWTZ);
cbLen5 = strlen(dtWTZ);
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_DATE, SQL_DATE
    , 10, 0, &date, 10, &cbLen1);
SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_TIME, SQL_TIME
    , 11, 0, &time, 11, &cbLen3);
SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_TIMESTAMP,
    SQL_TIMESTAMP, 11, 0, &dt, 11, &cbLen3);
SQLBindParameter(hstmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR
    , 21, 0, timeWTZ, 21, &cbLen4);
SQLBindParameter(hstmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR
    , 30, 0, dtWTZ, 30, &cbLen5);

```

```

ret = SQLExecute(hstmt);
if (SQL_SUCCESS != ret)
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, sqlState, &nativeErr,
    errMsg, 200, &len);
printf("SQLExecute failed, reason: %s\n", errMsg);
}
else
printf("Inesrt successful!\n");
ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);

ret = SQLDisconnect(hdbc);
ret = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
ret = SQLFreeHandle(SQL_HANDLE_ENV, henv);
}

```

### 5.1.3 执行 Update 语句示例代码

```

void main()
{
char szDSN[] = "XGDB";
char szUID[] = "SYSDBA";
char szAuth[] = "SYSDBA";
SQLHENV henv = NULL;
SQLHDBC hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN ret = 0;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
ret = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
    SQL_OV_ODBC3, SQL_IS_INTEGER);
ret = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
ret = SQLConnect(hdbc, (SQLCHAR*)szDSN, SQL_NTS, (SQLCHAR*)szUID,
    SQL_NTS, (SQLCHAR*)szAuth, SQL_NTS);
if (ret != SQL_SUCCESS)
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, 1, sqlState, &nativeErr, errMsg
    , 200, &len);
printf("Can't connect to server, reason: %s\n", errMsg);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return;
}
printf("Connect successful!\n");
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
char sql[] = "update mytable set c2 = 'update' where c1 = 1";
ret = SQLExecDirect(hstmt, (SQLCHAR*)sql, SQL_NTS);
if (ret != SQL_SUCCESS)

```

```
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, sqlState, &nativeErr,
    errMsg, 200, &len);
printf("SQLExecDirect failed, reason: %s\n", errMsg);
}
else
printf("Update successful!\n");
ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
ret = SQLDisconnect(hdbc);
ret = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
ret = SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

### 5.1.4 执行 Delete 语句示例代码

```
void main()
{
char szDSN[] = "XGDB";
char szUID[] = "SYSDBA";
char szAuth[] = "SYSDBA";
SQLHENV henv = NULL;
SQLHDBC hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN ret = 0;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
ret = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
    SQL_OV_ODBC3, SQL_IS_INTEGER);
ret = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
ret = SQLConnect(hdbc, (SQLCHAR*)szDSN, SQL_NTS, (SQLCHAR*)szUID,
    SQL_NTS, (SQLCHAR*)szAuth, SQL_NTS);
if (ret != SQL_SUCCESS)
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, 1, sqlState, &nativeErr, errMsg
    , 200, &len);
printf("Can't connect to server, reason: %s\n", errMsg);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return;
}
printf("Connect successful!\n");
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
char sql[] = "delete from mytable where id = ?";
ret = SQLPrepare(hstmt, (SQLCHAR*)sql, SQL_NTS);
SQLLEN cbLen = 4;
int id = 1;
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER
    , 4, 0, &id, 4, &cbLen);
```

```
ret = SQLExecute(hstmt);
if (ret != SQL_SUCCESS)
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, sqlState, &nativeErr,
  errMsg, 200, &len);
printf("SQLExecDirect failed, reason: %s\n", errMsg);
}
else
printf("Update successful!\n");
ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
ret = SQLDisconnect(hdbc);
ret = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
ret = SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

### 5.1.5 CLOB 大对象插入及查询示例代码

在启用大对象描述符时，从数据库中取 Clob 类型数据需要注意编码问题。如果存放 Clob 数据的数据库编码为 GBK，而 ODBC 连接数据库使用的编码为 UTF-8，服务端向 ODBC 驱动发送数据时，服务端会将 GBK 编码的文本转码至 UTF-8 编码的文本。在调用 SQLGetData 时，参数 BufferLength 的值不得超过 TargetValuePtr 所指向数据缓存区实际大小的 2/3 崩溃。原因是 UTF-8 编码比 GBK 编码使用更多的内存空间，在这种情况下 StrLen\_or\_IndPtr 的值将大于 BufferLength 的值。

```
void main()
{
char szDSN[] = "XGDB";
char szUID[] = "SYSDBA";
char szAuth[] = "SYSDBA";
SQLHENV henv = NULL;
SQLHDBC hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN ret = 0;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
ret = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
  SQL_OV_ODBC3, SQL_IS_INTEGER);
ret = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
ret = SQLConnect(hdbc, (SQLCHAR*)szDSN, SQL_NTS, (SQLCHAR*)szUID,
  SQL_NTS, (SQLCHAR*)szAuth, SQL_NTS);
if (ret != SQL_SUCCESS)
{
SQLCHAR sqlState[128];
SQLINTEGER nativeErr;
SQLCHAR errMsg[256];
SQLSMALLINT len;
SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, 1, sqlState, &nativeErr, errMsg
  , 200, &len);
}
```

```

printf("Can't connect to server, reason: %s\n", errMsg);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return;
}
printf("Connect successful!\n");
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

FILE* fp = fopen("ODBC.txt", "rb");

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

int id = 1;
SQLLEN cbIdLen = 4;
SQLLEN cbLen = SQL_LEN_DATA_AT_EXEC(0);
char sql[] = "insert into mytable(id, txt) values(?,?)";
ret = SQLPrepare(hstmt, (SQLCHAR*)sql, SQL_NTS);
ret = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
    SQL_INTEGER, 0, 0, &id, 4, &cbIdLen);
ret = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
    SQL_LONGVARCHAR, 123456789, 0, (SQLPOINTER)2, 0, &cbLen);
ret = SQLExecute(hstmt);
while (ret == SQL_NEED_DATA)
{
    ret = SQLParamData(hstmt, &pToken);
    if (ret == SQL_NEED_DATA && (int)pToken == 2)
    {
        do {
            cbRead = fread(data, 1, 4096, fp);
            if (cbRead <= 0)break;
            SQLPutData(hstmt, data, cbRead);
        } while (1);
    }
}
fclose(fp);
ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);

ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
fp = fopen("Output.txt", "wb");
strcpy(sql, "select txt from mytable where id = 1");
ret = SQLExecDirect(hstmt, (SQLCHAR*)sql, SQL_NTS);
ret = SQLFetch(hstmt);
SQLLEN retlen;
while (TRUE)
{
    ret = SQLGetData(hstmt, 1, SQL_C_CHAR, data, 4096, &retlen);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO)
        break;
    fwrite(data, 1, retlen, fp);
}
fclose(fp);
ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
ret = SQLDisconnect(hdbc);

```



```
ret = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);  
ret = SQLFreeHandle(SQL_HANDLE_ENV, henv);  
}
```

## 5.1.6 BLOB 大对象插入及查询示例代码

```
void main()  
{  
char szDSN[] = "XGDB";  
char szUID[] = "SYSDBA";  
char szAuth[] = "SYSDBA";  
SQLHENV henv = NULL;  
SQLHDBC hdbc = NULL;  
SQLHSTMT hstmt = NULL;  
SQLRETURN ret = 0;  
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);  
ret = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)  
SQL_OV_ODBC3, SQL_IS_INTEGER);  
ret = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);  
ret = SQLConnect(hdbc, (SQLCHAR*)szDSN, SQL_NTS, (SQLCHAR*)szUID,  
SQL_NTS, (SQLCHAR*)szAuth, SQL_NTS);  
if (ret != SQL_SUCCESS)  
{  
SQLCHAR sqlState[128];  
SQLINTEGER nativeErr;  
SQLCHAR errMsg[256];  
SQLSMALLINT len;  
SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, 1, sqlState, &nativeErr, errMsg  
, 200, &len);  
printf("Can't connect to server, reason: %s\n", errMsg);  
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);  
SQLFreeHandle(SQL_HANDLE_ENV, henv);  
return;  
}  
printf("Connect successful!\n");  
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);  
  
FILE* fp = fopen("ODBC.png", "rb");  
  
int cbRead = 0;  
char data[4096];  
SQLPOINTER pToken = NULL;  
  
int id = 1;  
SQLLEN cbIdLen = 4;  
SQLLEN cbLen = SQL_LEN_DATA_AT_EXEC(0);  
char sql[] = "insert into mytable(id, img) values(?,?)";  
ret = SQLPrepare(hstmt, (SQLCHAR*)sql, SQL_NTS);  
ret = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,  
SQL_INTEGER, 0, 0, &id, 4, &cbIdLen);  
ret = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,  
SQL_LONGVARCHAR, 123456789, 0, (SQLPOINTER)2, 0, &cbLen);  
ret = SQLExecute(hstmt);  
while (ret == SQL_NEED_DATA)  
{
```

```
ret = SQLParamData(hstmt, &pToken);
if (ret == SQL_NEED_DATA && (int)pToken == 2)
{
do {
cbRead = fread(data, 1, 4096, fp);
if (cbRead <= 0)break;
SQLPutData(hstmt, data, cbRead);
} while (1);
}
}
fclose(fp);
ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);

ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
fp = fopen("Output.png", "wb");
strcpy(sql, "select img from mytable where id = 1");
ret = SQLExecDirect(hstmt, (SQLCHAR*)sql, SQL_NTS);
ret = SQLFetch(hstmt);
SQLLEN retlen;
while (TRUE)
{
ret = SQLGetData(hstmt, 1, SQL_C_CHAR, data, 4096, &retlen);
if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO)
break;
fwrite(data, 1, retlen, fp);
}
fclose(fp);
ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
ret = SQLDisconnect(hdbc);
ret = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
ret = SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

### 5.1.7 SQLPrimaryKeys 主键信息查询示例代码

```
typedef struct
{
char tableCat[50];
SQLLEN tableCatLen;
char tableSchem[50];
SQLLEN tableSchemLen;
char tableName[50];
SQLLEN tableNameLen;
char columnName[50];
SQLLEN columnNameLen;
short keySeq;
SQLLEN keySeqLen;
char pkName[50];
SQLLEN pkNameLen;
}PrimaryKeysInfo;

void main()
{
char szDSN[] = "XGDB";
char szUID[] = "SYSDBA";
```

```
char szAuth[] = "SYSDBA";
SQLHENV henv = NULL;
SQLHDBC hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN ret = 0;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
ret = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
    SQL_OV_ODBC3, SQL_IS_INTEGER);
ret = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
ret = SQLConnect(hdbc, (SQLCHAR*)szDSN, SQL_NTS, (SQLCHAR*)szUID,
    SQL_NTS, (SQLCHAR*)szAuth, SQL_NTS);
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

ret = SQLPrimaryKeys(hstmt, (SQLCHAR*)NULL, 0, (SQLCHAR*)"SYSDBA",
    SQL_NTS, (SQLCHAR*)"MYTALBE", SQL_NTS);

PrimaryKeysInfo info;
ret = SQLBindCol(hstmt, 1, SQL_C_CHAR, info.tableCat, 50, &info.
    tableCatLen);
ret = SQLBindCol(hstmt, 2, SQL_C_CHAR, info.tableSchem, 50, &info.
    tableSchemLen);
ret = SQLBindCol(hstmt, 3, SQL_C_CHAR, info.tableName, 50, &info.
    tableNameLen);
ret = SQLBindCol(hstmt, 4, SQL_C_CHAR, info.columnName, 50, &info.
    columnNameLen);
ret = SQLBindCol(hstmt, 5, SQL_C_SHORT, &info.keySeq, 2, &info.
    keySeqLen);
ret = SQLBindCol(hstmt, 6, SQL_C_CHAR, info.pkName, 50, &info.
    pkNameLen);

while (TRUE)
{
    ret = SQLFetch(hstmt);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO)
        break;
    printf("Catalog: %s, Schema: %s, Name: %s, ColName: %s, Seq: %d,
        PkName: %s\n",
        info.tableCat, info.tableSchem, info.tableName, info.columnName,
        info.keySeq, info.pkName);
}
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

### 5.1.8 SQLColumns 列信息查询示例代码

```
typedef struct {
char tableCat[50];
SQLLEN tableCatLen;
char tableSchem[50];
SQLLEN tableSchemLen;
char tableName[50];
SQLLEN tableNameLen;
```

```

char columnName[50];
SQLLEN columnNameLen;
short dataType;
SQLLEN dataTypeLen;
char typeName[50];
SQLLEN typeNameLen;
int columnSize;
SQLLEN columnSizeLen;
int bufferLength;
SQLLEN bufferLengthLen;
short decimalDigits;
SQLLEN decimalDigitsLen;
short numPrecRadix;
SQLLEN numPrecRadixLen;
short nullable;
SQLLEN nullableLen;
char comments[50];
SQLLEN commentsLen;
char columnDef[50];
SQLLEN columnDefLen;
short sqlDataType;
SQLLEN sqlDataTypeLen;
short sqlDatetimeSub;
SQLLEN sqlDatetimeSubLen;
int charOctetLength;
SQLLEN charOctetLengthLen;
int ordinalPosition;
SQLLEN ordinalPositionLen;
char isNullable[50];
SQLLEN isNullableLen;
}ColInfo;

void main()
{
char szDSN[] = "XGDB";
char szUID[] = "SYSDBA";
char szAuth[] = "SYSDBA";
SQLHENV henv = NULL;
SQLHDBC hdbc = NULL;
SQLHSTMT hstmt = NULL;
SQLRETURN ret = 0;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
ret = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
    SQL_OV_ODBC3, SQL_IS_INTEGER);
ret = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
ret = SQLConnect(hdbc, (SQLCHAR*)szDSN, SQL_NTS, (SQLCHAR*)szUID,
    SQL_NTS, (SQLCHAR*)szAuth, SQL_NTS);
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

ret = SQLColumns(hstmt,
    (SQLCHAR*)"SYSTEM", SQL_NTS,
    (SQLCHAR*)"SYSDBA", SQL_NTS,
    (SQLCHAR*)"MYTABLE", SQL_NTS,
    (SQLCHAR*)"%", SQL_NTS);

```

```
ColInfo info;
SQLBindCol(hstmt, 1, SQL_C_CHAR, info.tableCat, 50, &info.
    tableCatLen);
SQLBindCol(hstmt, 2, SQL_C_CHAR, info.tableSchem, 50, &info.
    tableSchemLen);
SQLBindCol(hstmt, 3, SQL_C_CHAR, info.tableName, 50, &info.
    tableNameLen);
SQLBindCol(hstmt, 4, SQL_C_CHAR, info.columnName, 50, &info.
    columnNameLen);
SQLBindCol(hstmt, 5, SQL_C_SHORT, &info.dataType, 2, &info.
    dataTypeLen);
SQLBindCol(hstmt, 6, SQL_C_CHAR, info.typeName, 50, &info.
    typeNameLen);
SQLBindCol(hstmt, 7, SQL_C_LONG, &info.columnSize, 4, &info.
    columnSizeLen);
SQLBindCol(hstmt, 8, SQL_C_LONG, &info.bufferLength, 4, &info.
    bufferLengthLen);
SQLBindCol(hstmt, 9, SQL_C_SHORT, &info.decimalDigits, 2, &info.
    decimalDigitsLen);
SQLBindCol(hstmt, 10, SQL_C_SHORT, &info.numPrecRadix, 2, &info.
    numPrecRadixLen);
SQLBindCol(hstmt, 11, SQL_C_SHORT, &info.nullable, 2, &info.
    nullableLen);
SQLBindCol(hstmt, 12, SQL_C_CHAR, info.comments, 50, &info.
    commentsLen);
SQLBindCol(hstmt, 13, SQL_C_CHAR, info.columnDef, 50, &info.
    columnDefLen);
SQLBindCol(hstmt, 14, SQL_C_SHORT, &info.sqlDataType, 2, &info.
    sqlDataTypeLen);
SQLBindCol(hstmt, 15, SQL_C_SHORT, &info.sqlDatetimeSub, 2, &info.
    sqlDatetimeSubLen);
SQLBindCol(hstmt, 16, SQL_C_LONG, &info.charOctetLength, 4, &info.
    charOctetLengthLen);
SQLBindCol(hstmt, 17, SQL_C_LONG, &info.ordinalPosition, 4, &info.
    ordinalPositionLen);
SQLBindCol(hstmt, 18, SQL_C_CHAR, info.isNullable, 50, &info.
    isNullableLen);

while (TRUE)
{
    ret = SQLFetch(hstmt);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO)
        break;
    printf("ColName: %s, Comments: %s\n",
        info.columnName, info.comments);
}
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
}
```

## 5.2 C

### 5.2.1 建立连接示例

```
private DbConnection Login(string connstr)
{
    DbConnection conn = new OdbcConnection(connstr);
    try
    {
        conn.Open();
        return conn;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return null;
    }
}
```

### 5.2.2 常见数据类型参数方式插入示例

```
string connStr = "DSN=XGDB;USER=SYSDBA;PWD=SYSDBA";
OdbcConnection conn = new OdbcConnection(connStr);
try
{
    conn.Open();
    OdbcCommand cmd = conn.CreateCommand();
    cmd.CommandType = CommandType.Text;
    cmd.CommandText = "insert into mytable (c1,c2,c3) values(?,?,?)";

    OdbcParameter[] parameters ={
        new OdbcParameter("c1",OdbcType.BigInt,8),
        new OdbcParameter("c2",OdbcType.SmallInt,2),
        new OdbcParameter("c3",OdbcType.TinyInt,1) };
    parameters[0].Value = 3141592653;
    parameters[1].Value = (short)31415;
    parameters[2].Value = 100;
    parameters[0].Direction = ParameterDirection.Input;
    parameters[1].Direction = ParameterDirection.Input;
    parameters[2].Direction = ParameterDirection.Input;

    cmd.Parameters.Add(parameters[0]);
    cmd.Parameters.Add(parameters[1]);
    cmd.Parameters.Add(parameters[2]);

    cmd.ExecuteNonQuery();

    conn.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

### 5.2.3 非参数方式更改数据示例

```
string connStr = "DSN=XGDB;USER=SYSDBA;PWD=SYSDBA";
OdbcConnection conn = new OdbcConnection(connStr);
try
{
    conn.Open();
    OdbcCommand cmd = conn.CreateCommand();
    cmd.CommandText = "update mytable set c2 = 'update' where id = 1";

    if (cmd.ExecuteNonQuery() > 0)
        Console.WriteLine("Update successful!");
    else
        Console.WriteLine("Update failed!");

    conn.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

### 5.2.4 DataTable 使用示例

```
private DataSet GetDataSet(OdbcConnection conn)
{
    try
    {
        string sql = "select * from mytable";
        OdbcDataAdapter adapter = new OdbcDataAdapter(sql, conn);
        DataSet dataSet = new DataSet();
        adapter.Fill(dataSet, "mytable");
        return dataSet;
    }
    catch
    {
        return null;
    }
}
```

### 5.2.5 调用存储过程示例

```
create or replace procedure "SYSDBA"."PROC_TEST" (c1 char, c2 out
    varchar)
AS
BEGIN
select col2 into c2 from mytable where col1 = c1;
END;
```

```
string connStr = "DSN=XGDB;USER=SYSDBA;PWD=SYSDBA";
OdbcConnection conn = new OdbcConnection(connStr);
conn.Open();
OdbcCommand cmd = conn.CreateCommand();
```

```
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = "execute PROC_TEST(?,?)";

OdbcParameter[] parameters = { new OdbcParameter("c1", OdbcType.
    Char, 30), new OdbcParameter("c2", OdbcType.VarChar, 50) };

parameters[0].Value = "XuguDB";
parameters[1].Direction = ParameterDirection.Output;

cmd.Parameters.Add(parameters[0]);
cmd.Parameters.Add(parameters[1]);

cmd.ExecuteNonQuery();

Console.WriteLine((string)parameters[1].Value);

conn.Close();
```

## 5.2.6 插入二进制数据示例

```
string connStr = "DSN=XGDB;USER=SYSDBA;PWD=SYSDBA;";
try
{
    OdbcConnection conn = new OdbcConnection(connStr);
    conn.Open();
    FileStream fs = new FileStream("in.png", FileMode.Open);
    byte[] data = new byte[fs.Length];
    fs.Read(data, 0, (int)fs.Length);
    fs.Close();

    OdbcCommand cmd = conn.CreateCommand();
    cmd.CommandText = "INSERT INTO MYTABLE(ID, NAME, IMG) VALUES(?,?,?)
        ";

    cmd.Parameters.Add("ID", OleDbType.Integer).Value = 100;
    cmd.Parameters.Add("NAME", OleDbType.VarChar).Value = "张三";
    cmd.Parameters.Add("IMG", OleDbType.Binary).Value = data;
    int res = cmd.ExecuteNonQuery();
    Console.WriteLine(res == 1 ? "Insert successful" : "Insert failed")
        ;

    cmd.CommandText = "SELECT IMG FROM MYTABLE WHERE ID = 100";
    OdbcDataReader reader = cmd.ExecuteReader();
    reader.Read();
    byte[] buf = new byte[reader.GetBytes(0, 0, null, 0, int.MaxValue)
        ];
    reader.GetBytes(0, 0, buf, 0, buf.Length);
    reader.Close();

    FileStream output = new FileStream("out.png", FileMode.Create,
        FileAccess.Write);
    output.Write(buf, 0, buf.Length);
    output.Close();
}
catch (Exception ex)
```



```
{  
Console.WriteLine(ex);  
}
```

## 5.3 VB

```
Imports System.Data.Odbc  
  
Module XuguOdbcDemo  
  
Sub Main()  
Dim connStr As String  
Dim conn As OdbcConnection  
Dim cmd As OdbcCommand  
Dim reader As OdbcDataReader  
  
connStr = "Driver=XuguSQL 11.2;Server=127.0.0.1;Port=5138;DB=SYSTEM  
;UID=SYSDBA;PWD=SYSDBA;"  
conn = New OdbcConnection(connStr)  
conn.Open()  
cmd = conn.CreateCommand()  
cmd.CommandText = "select * from mytable where id = ?"  
cmd.Parameters.Add("id", OdbcType.Int).Value = 100  
reader = cmd.ExecuteReader()  
While reader.Read()  
Console.WriteLine(reader.GetInt32(0) & ", " &  
reader.GetString(1) & ", " &  
reader.GetString(2) & ", " &  
reader.GetString(3))  
End While  
reader.Close()  
conn.Close()  
Console.ReadKey()  
End Sub  
  
End Module
```

## 5.4 PHP

### 5.4.1 查询示例

```
$conn = odbc_connect("XGDB", "SYSDBA", "SYSDBA");  
  
$sql = "select id, name from odbc_test where id < 10";  
$rs = odbc_exec($conn, $sql);  
  
while(odbc_fetch_row($rs)){  
$id = odbc_result($rs, "id");  
$name = odbc_result($rs, "name");  
echo "$id, $name\n";  
}
```

```
}  
  
odbc_close($conn);
```

## 5.4.2 Prepare 示例

```
$conn = odbc_connect("XGDB", "SYSDBA", "SYSDBA");  
  
$sql = "select id,name from odbc_test where id = ?";  
$stmt = odbc_prepare($conn, $sql);  
  
$id = 1;  
odbc_execute($stmt, array($id));  
  
while(odbc_fetch_row($stmt)){  
    $id = odbc_result($stmt, 1);  
    $name = odbc_result($stmt, 2);  
    echo "$id, $name\n";  
}  
  
odbc_close($conn);
```

## 5.4.3 PDO 示例

```
$connStr = "odbc:DSN=XGDB;";  
$pdo = new PDO($connStr);  
$stmt = $pdo->prepare("select * from mytable where id = :id");  
$stmt->bindValue(":id", 1);  
$stmt->execute();  
while($row = $stmt->fetch()) {  
    echo $row["ID"] . "\n";  
}
```

# 6 错误码与常见错误定位

## 6.1 错误码详解

表 6-1 错误码详解

错误码	错误描述	函数返回值	错误出现原因	分析与建议
01000	General warning	-1	一般警告	可以执行成功，但非完全正确，需检查数据精度是否有损失，数据输入是否有截断等
01001	Disconnect error	-1	连接断开错误	检查数据连接是否仍存活，是否有被关闭
01002	游标操作冲突	-1	连接断开错误	检查游标操作是否在允许的范围内
01003	NULL value eliminated in set function	-1	set 函数中空值被排除	检查空值是否在需要的范围内，是的话另行处理
01004	String data, right truncated	-1	字符串数据，右截断	数据宽度超过缓冲长度，右侧数据截断，建议增大缓冲区的长度，或者扩大数据类型中的长度或精度
01006	Privilege not revoked	-1	特权未被撤销	建议检查用户的权限的授予与回收
01007	Privilege not granted	-1	未授权	建议检查用户的权限的授予与回收
01S00	Invalid connection string attribute	-1	无效的连接字符串属性	建议检查用户的连接串的各个属性的值
01S02	Option value changed	-1	选项值已更改	在执行操作时，先检查选项值是否符合要求

接下页

表 6-1 错误码详解 (续)

错误码	错误描述	函数返回值	错误出现原因	分析与建议
01S03	No rows updated/deleted	-1	没有更新/删除行	检查 sql 的检索条件是否可以选中需要的行数据
01S04	No rows updated/deleted	-1	更新/删除了多行	请检查 sql 的检索条件是否唯一定位需要的行数据
01S06	Attempt to fetch before the result set returned the first rowset	-1	尝试在结果集返回首行集之前提取	应在结果集返回首行数据之后进行数据提取, 建议先取结果集行值属性查看
01S07	Fractional truncation	-1	数据缓冲长度不够, 部分截断	建议给与适当的数据缓冲宽度, 保证数据获取的完整性
01S08	Error saving File DSN	-1	保存文件 DSN 时出错	确保文件路径正确以及有在目标路径创建和写文件的权限
01S09	Invalid keyword	-1	无效关键字	建议检查使用的关键字, 替换为可用的关键字
07001	Wrong number of parameters	-1	参数数量错误	参数数量应与 sql 里面的占位符数量相匹配, 建议检查
07002	COUNT field incorrect	-1	计数字段不正确	检查 count 的字段是在结果集检索的范围
07005	Prepared statement not a cursor-specification	-1	准备好的语句不是游标规范	可以用作游标的都是合法的 select 查询语句, 建议先检查 sql 语句的正确性
				接下页

表 6-1 错误码详解 (续)

错误码	错误描述	函数返回值	错误出现原因	分析与建议
07006	Restricted data type attribute violation	-1	受限数据类型属性冲突	检查对应数据类型的限制情况, 根据需要设置属性或选择其他的数据类型替换
07009	Invalid descriptor index	-1	无效的索引描述符	检查索引描述符的合法性, 对照数据库里面的索引描述符的情况
07S01	Invalid use of default parameter	-1	默认参数使用无效	根据需要和 API 函数需求选择可用的参数填充
08001	Client unable to establish connection	-1	客户端不能建立连接	排查原因, 是否 sock 耗尽, 是否网络未联通, 服务端是否正常提供服务, 是否之前连续多次连接属性输入错误
08002	Connection name in use	-1	连接名正在使用中	排查连接名的使用占用情况
08003	Connection does not exist	-1	连接不存在	检查连接时未创建还是丢失
08004	Server rejected the connection	-1	服务器拒绝了连接	检查是连接属性是否正确, 主要是用户名和密码, 然后就是之前是否多次尝试连接, 但连接不成功, 如果之前多次连接不成功, 那么服务器端会在一段时间内拒绝本 IP 的连接请求
				接下页

表 6-1 错误码详解 (续)

错误码	错误描述	函数返回值	错误出现原因	分析与建议
08007	Connection failure during transaction	-1	事务中连接失效	在事务环境下语句执行后，事务尚未提交，连接已经不在，那么应检查连接失效原因，因未提交事务，所以本事务内的执行都会被回滚，解决连接问题后，重做本事务即可
08S01	Communication link failure	-1	通信链接故障	请排查网络问题
21S01	Column count does not match value count	-1	列数与值个数不匹配	检查 SQL 语句的列与值的映射关系，保证其一一对应
21S02	Degree of derived table does not match column list	-1	派生表字段列不匹配	从基表或视图来的列与数据列的链不匹配，建议检查相关列的对应情况
22001	String data, right-truncated	-1	字符串数据右侧截断	数据超过存放的缓冲的长度，建议增大缓冲区的长度
22002	Indicator variable required but not supplied	-1	指标变量需要，但未提供	建议检查具体需要哪些指示列变量，看是否正确提供指示变量的值
22003	Numeric value out of range	-1	Numeric 值超界	建议检查 numeric 的定义和它应该存放的列值范围，根据业务需求扩大定义范围或排除异常值
				接下页

表 6-1 错误码详解（续）

错误码	错误描述	函数返回值	错误出现原因	分析与建议
22007	Invalid datetime format	-1	datetime 格式无效	Datetime 的常见可以格式有 YYYYMMDDHH24MISS,YYYY-MM-DD HH24:MI:SS , YYYY-MM-DD HH24:MI:SSSSS 等
22008	Datetime field overflow	-1	Datetime 类型字段溢出	请参考 datetime 的格式分配适当的空间，常见的格式有 YYYYMMDDHH24MISS,YYYY-MM-DD HH24:MI:SS , YYYY-MM-DD HH24:MI:SSSSS 等
22012	Division by zero	-1	0 作除数	0 不能做除数，请排查输入值
22015	Interval field overflow	-1	Interval 字段溢出	Interval 的不同类型有各自的类型长度，基于秒的都是 int64 的数值存放，非基于秒的都是 int32 数值存放，请参考使用
22018	Invalid character value for cast specification	-1	强制转换的字符值无效	当前字符值与目标类型之间不能进行合理的转换，建议根据业务需要转换为其它类型
22019	Invalid escape character	-1	无效转义字符	转义字符不能正确转义，建议根据业务需求进行更改
22026	String data, length mismatch	-1	字符串数据值和长度不匹配	根据字符串的数据值的情况合理调整其长度
				接下页

表 6-1 错误码详解 (续)

错误码	错误描述	函数返回值	错误出现原因	分析与建议
23000	Integrity constraint violation	-1	完整性约束冲突	检查 SQL 语句中的值与对应的数据表的映射关系, 修改 SQL 使其符合完整性约束
24000	Invalid cursor state	-1	无效的游标状态	合理使用游标确保游标状态在合理范围
25000	Invalid transaction state	-1	无效的事务状态	检查事务的语句运行情况 and 过程, 以及连接的情况, 保证事务运行情况处于合理范围
25S01	Transaction state unknown	-1	事务状态未知	事务状态丢失, 建议重建事务环境, 分析事务状态变化情况
28000	Invalid authorization specification	-1	无效的授权规范	根据业务需求, 给予目标用户相应的适当权限
34000	Invalid cursor name	-1	无效的游标名	游标处于非可用状态, 建议检查连接的可用性
3C000	Duplicate cursor name	-1	重复的游标名	建议检查之前使用的游标名, 和当前游标名是否有无重复
3D000	Invalid catalog name	-1	无效的数据库名	数据库名不是当前系统的有效数据库名, 建议检查数据库内的数据库运行情况
3F000	Invalid schema name	-1	无效的模式名	模式名不是当前数据库的有效模式名, 建议检查数据库内的模式名
				接下页



表 6-1 错误码详解 (续)

错误码	错误描述	函数返回值	错误出现原因	分析与建议
40001	Serialization failure	-1	序列化失败	检查序列化流程
40002	Integrity constraint violation	-1	完整性约束冲突	检查数据库内的完整性约束条件, 调整 sql 使其满足完整性约束后再执行
40003	Statement completion unknown	-1	语句完成未知	分析 sql 语句可能出现的情况, 结合连接状态和服务端的情况分析语句的完成度情况
42000	Syntax error or access violation	-1	语法错误或访问冲突	修改 SQL 语句使其语法合法, 或确保其访问状态一致
42S01	Base table or view already exists	-1	基表或视图已经存在	检查其基表的使用情况选择换表或者是删表
42S02	Base table or view not found	-1	找不到基表或视图	找不到对应的基表或者视图, 根据业务需求进行创建
42S11	Index already exists	-1	索引已存在	检查索引名是否与已有索引有冲突, 检查索引列是否与已有索引是完全一致的
42S12	Index not found	-1	索引已存在	根据需要创建索引或者选择其他的索引
42S21	Column already exists	-1	列已存在	选择其他的列或者操作表修改列
42S22	Column not found	-1	找不到列	根据需要调整 SQL 语句
				接下页

表 6-1 错误码详解（续）

错误码	错误描述	函数返回值	错误出现原因	分析与建议
44000	WITH CHECK OPTION violation	-1	违反检查选项	检查服务器端的本 sql 涉及到的列 check 约束的情况，调整 sql 的内容，使其符合检查约束
HYT00	Timeout expired	-1	超时过期	检查是 sql 执行超时过期还是长时间未使用连接导致超时过期
HY000	General driver defined error	-1	通用驱动程序定义的错误	检查错误发生的原因进行代码修改
HY001	Memory allocation error	-1	内存申请错误	检查内存申请的长度合法性和内存剩余，以及当前堆栈是否处于可用状态
HY002	Invalid column number	-1	无效的列号	列号应处于 0 到当前最大列数的范围之内
HY003	Invalid application buffer type	-1	无效的应用程序缓冲区类型	注意其缓冲区的类型和生存周期
HY004	Invalid SQL data type	-1	无效的 SQL 数据类型	检查 sql 的合法性和当前连接的可用性
HY007	Associated statement is not prepared	-1	关联语句未准备好	检查代码是否缺少相应的语句关联过程
HY008	Operation canceled	-1	操作已取消	检查调用 API 函数的过程合理性
HY009	Invalid use of null pointer	-1	无效使用空指针	检查指针的使用情况以及其生存周期

接下页

表 6-1 错误码详解（续）

错误码	错误描述	函数返回值	错误出现原因	分析与建议
HY010	Function sequence error	-1	函数序列错误	检查函数序列调用的情况
HY011	Attribute can not be set now	-1	无法设置当前属性	检查当前属性是否为可编辑属性，以及是否处于当前可编辑阶段
HY012	Invalid transaction operation code	-1	无效的事务操作码	检查事务的当前状态，与应该使用的操作码
HY013	Memory management error	-1	内存管理错误	检查内存管理分配和释放是合理的
HY015	No cursor name available	-1	没有可用的游标名	检查游标名丢失的原因，是否提前释放等等
HY016	Cannot modify an implementation row descriptor	-1	无法修改一个实现的行描述符	实现的行描述符不能在当前阶段被修改，或者说修改了也不再起作用了，应该提前确定其值
HY017	Invalid use of an automatically allocated descriptor handle	-1	自动分配的描述符句柄的无效使用	自动分配的描述符句柄有其默认使用环境
HY018	Server declined cancel request	-1	服务器拒绝了取消请求	某些操作服务器一经执行到某个阶段便不可取消，但是后续可以根据其需求选择回滚等等
HY019	Non-character and non-binary data sent in pieces	-1	分段发送的非字符和非二进制数据	只有字符串或者二进制数据是可以分段发送的，int，double 等类型是不可以拆开分段发送的
				接下页

表 6-1 错误码详解（续）

错误码	错误描述	函数返回值	错误出现原因	分析与建议
HY020	Attempt to concatenate a null value	-1	尝试连接一个空值	应预先做值检查，或取消此连接操作
HY021	Inconsistent descriptor information	-1	描述符信息不一致	描述符的内容与预期的不符
HY024	Invalid attribute value	-1	属性值无效	属性值处于有效范围之外，建议调整
HY090	Invalid string or buffer length	-1	无效的字符串或缓冲区长度	建议调整其长度
HY091	Invalid descriptor field identifier	-1	描述符字段标识符无效	注意检查其标识符的使用周期
HY092	Invalid attribute/option identifier	-1	无效的属性/选项标识符	注意当前环境的属性选项标识符使用
HY093	Invalid parameter number	-1	无效的参数号	注意其参数号的值是否合法
HY095	Function type out of range	-1	函数类型超出范围	注意函数类型的范围与要求是否相符
HY096	Invalid information type	-1	无效的信息类型	注意信息的类型情况
HY097	Column type out of range	-1	列类型超出范围	注意列类型是否处于当前服务范围

接下页

表 6-1 错误码详解（续）

错误码	错误描述	函数返回值	错误出现原因	分析与建议
HY098	Scope type out of range	-1	范围类型超出范围	注意其范围类型的边界
HY099	Nullable type out of range	-1	可空类型超出范围	注意类型的范围
HY100	Uniqueness option type out of range	-1	唯一性选项类型超出范围	注意类型的范围
HY101	Accuracy option type out of range	-1	精度选项类型超出范围	注意类型的范围
HY103	Invalid retrieval code	-1	无效的检索代码	注意检查检索代码的服务端对应情况
HY104	Invalid precision or scale value	-1	精度标度错	注意给定的精度标度值
HY105	Invalid parameter type	-1	参数类型错	注意参数类型的选择范围
HY106	Fetch type out of range	-1	获取类型超界	注意获取类型的选择范围
HY107	Row value out of range	-1	行数据超界	检查整行数据的长度
HY109	Invalid cursor position	-1	游标位置无效	检查游标位置与结果集的状态，确保其处于合理的范围
HY110	Invalid driver completion	-1	无效的驱动完成	检查其驱动运行情况与状态
HY111	Invalid bookmark value	-1	无效的书签值	检查 bookmark 的值

接下页

表 6-1 错误码详解（续）

错误码	错误描述	函数返回值	错误出现原因	分析与建议
HYC00	Optional feature not implemented	-1	未实现的可选功能	建议根据业务需要选择其他的可用分支功能
IM001	Driver does not support this function	-1	驱动程序不支持此功能	根据需求选择其他的代替功能
IM002	Data source name not found and no default driver specified	-1	未找到数据源名称，也未指定默认驱动程序	检查驱动是否安装正确，数据源是否正确注册
IM003	Specified driver could not be loaded	-1	无法加载指定的驱动程序	请检查程序运行环境是否与驱动程序匹配，加载位置是否正确
IM004	Driver's SQLAIlocHandle on SQL_HANDLE_ENV failed	-1	驱动申请句柄时在环境句柄上失败	请检查程序运行环境，动态库是否已正确加载
IM005	Driver's SQLAIlocHandle on SQL_HANDLE_DBC failed	-1	驱动申请句柄时在连接句柄上失败	请检查运行环境，DM 是否可用
IM006	Driver's SQLSetConnectAttr failed	-1	驱动设置连接属性失败	请检查属性值是否正确以及属性项是否正确，环境是否可用
				接下页

表 6-1 错误码详解（续）

错误码	错误描述	函数返回值	错误出现原因	分析与建议
IM007	No data source or driver specified; dialog prohibited	-1	未指定数据源或驱动程序；对话框加载失败	请排查运行环境，确保数据源可用，驱动程序正确注册
IM008	Dialog failed	-1	对话框故障	请检查运行环境，其 windows 底层环境是否与驱动程序版本吻合
IM009	Unable to load translation DLL	-1	无法加载翻译动态库	请检查运行环境，其 windows 底层环境是否与驱动程序版本吻合
IM010	Data source name too long	-1	数据源名称太长	请选择适当长度的数据源名称原则上不超过 128 字节
IM011	Driver name too long	-1	驱动程序名称太长	驱动程序名称不宜超过 128 字节
IM012	DRIVER keyword syntax error	-1	DRIVER 关键字语法错误	请检查驱动名称关键字是否与示例文件一致
IM013	Trace file error	-1	跟踪文件错误	请确认跟踪文件路径正确，且程序运行者有在路径下写文件的权限
IM014	Invalid name of File DSN	-1	文件 DSN 的名称无效	请确认文件 DSN 在所在的位置存在
IM015	Corrupt file data source	-1	文件数据源损坏	文件数据源的格式被损坏，应该选择重新输出文件数据源，确定其格式后再进行编辑
接下页				

表 6-1 错误码详解（续）

错误码	错误描述	函数返回值	错误出现原因	分析与建议
E1998	WINDOWS 连接未建立成功	-1	IP 或者端口错误，或服务端未启动 sock 连接建立失败	检查网络是否畅通，服务器端是否仍处于活动状态，检查端口是否正确，检查 IP 是否正确，检查连接属性

## 6.2 常见错误定位

### 6.2.1 数据库连接失败

#### 6.2.1.1 网络不通

**错误定位方式：**通过本机 PING 虚谷数据库服务器的 IP 地址。

**解决方法：**修改本机或者虚谷数据库 IP 地址，确保两个 IP 地址在同一网段或者中间有 VPN 连接。

#### 6.2.1.2 连接参数错误

**错误定位方式：**检查连接字符串是否与虚谷数据库相应参数（IP 地址、端口号、数据库名、用户名、用户密码等）匹配。

**解决方法：**修改代码中的连接字符串，确保其中的 IP 地址、端口、数据库名、用户名和用户密码正确。

#### 6.2.1.3 用户密码连续错三次

**错误定位方式：**判断是否存在连续三次输入用户名密码失败的情况。

**解决方法：**等待 IP 冻结时间（3 分钟）结束，然后输入正确的用户名密码。

### 6.2.2 虚谷 ODBC 驱动无法正常使用

#### 6.2.2.1 执行 SQL 语句相关错误

##### SQL 语句参数输入不匹配错误



**说明：**针对需要输入参数的 SQL 语句，输入的参数与数据库端的参数不匹配

**错误定位方式：**报错 SQL\_ERROR，检查通过 SQLBindParameter 函数输入的参数与数据库端的参数是否匹配。

**解决方法：**使用 SQLBindParameter 函数对插入参数进行绑定，正确输入参数的数据类型，详情请参考本文档的第四章（数据类型的相关说明）。

**部分代码举例：**

```
char sql[] = "insert into t8_odbc(c1,c2,c3) values(?,?,?)";
rs =SQLPrepare(hstmt, (UCHAR*)sql, strlen(sql)); // SQLPrepare 准备要执行的SQL语句。
SQLLEN cbLen1,cbLen2,cbLen3
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;
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);
rs =SQLExecute(hstmt);
```

### 6.2.2.2 操作系统版本不兼容

**说明：**Windows 版本分为 32 位和 64 位，检查自己主机的操作系统是否与注册的数据源版本相匹配。

**错误定位方式：**报错 SQL\_ERROR，检查通过 SQLBindParameter 函数输入的参数与数据库端的参数是否匹配。

**解决方法：**若不匹配，64 位的操作系统应注册 64 位的数据源，32 位的同理。

### 6.2.2.3 结果集获取失败

**说明：**获取结果集失败，一般是数据类型不匹配和绑定过多列造成。

**错误定位方式：**报错 SQL\_ERROR，检查 SQL 语句所返回的结果集的列是否与 SQLBindCol 函数绑定的列相匹配（数量、数据类型）。

**解决方法：**。使用 SQLBindCol 函数对插入参数进行绑定，正确输入参数的数据类型，详情请参考本文档的第四章（数据类型的相关说明）

**部分代码举例：**

```
char sq2[]="select * from t8_odbc";
FILE* fp;
fp=fopen("D:\\tout_t8_odbc.txt","wb+");
rs =SQLExecDirect(hstmt, (UCHAR*)sq2, strlen(sq2));
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);
```



成都虚谷伟业科技有限公司

联系电话：400-8886236

官方网站：[www.xugudb.com](http://www.xugudb.com)