GBASE

GBase 8a 程序员手册 ODBC 篇

www.gbase.cn

GBase 8a 程序员手册 ODBC 篇，南大通用数据技术股份有限公司

通讯方式

南大通用数据技术股份有限公司

天津市西青区工华道 2 号天百中心 3 号楼 3 层

电话：022-58815678　　邮箱：info@gbase.cn

目 录

前言

手册简介

GBase 8a 程序员手册从程序员进行数据库开发的角度对 GBase 8a 进行详细介绍。

本手册介绍供客户端连接 GBase 8a 服务器用的 GBase 8a ODBC 接口驱动程序。GBase 对 ODBC3.X 标准提供支持。本部分告诉用户如何安装和使用 GBase 8a ODBC 驱动。这里也有关于能与 ODBC 一起工作的程序的信息，并回答了一些最常见的关于 ODBC 的问题。

公约

下面的文本约定用于本文档：

约　定	说　明
加粗字体	表示文档标题
大写英文（SELECT）	表示 GBase 8a 关键字
等宽字体	表示代码示例
…	表示被省略的内容。

1 ODBC 概述

1.1 ODBC 简介

ODBC（开放数据库互连）为客户端程序提供了一种通用的接口来存取数据库。ODBC 是标准化的 API，允许到 SQL 数据库服务器的连接。它根据 SQL Access Group 的说明书开发，并定义了一系列的函数调用，错误代码和数据类型，这些可以用来开发独立于数据库的应用程序。ODBC 经常被用在当要求独立于数据库或同步访问不同的数据源时。

ODBC 是应用程序广泛使用的数据库访问接口。它是基于对数据库 APIs 的 X/Open 和 ISO/IEC 标准的分级调用接口（CLI）规范，并且使用结构化查询语句（SQL）作为它的数据库访问语言。

ODBC 支持的 ODBC 函数综述在 ODBC API 参考手册中给出。关于 ODBC 一般和更多信息，参考请参考微软公司网站的相关内容。

1.2 ODBC 驱动管理器

ODBC 驱动管理器是管理 ODBC 应用和驱动程序之间的通信的库。它的主要功能包括：

- 解析数据源名字（DSN）
- 装载和卸载驱动程序
- 处理 ODBC 函数调用或传递它们到驱动程序

下面的驱动程序管理器经常被使用：

- Microsoft Windows 的 ODBC 驱动管理器(odbc32.dll)
- unixODBC Unix 驱动管理器(libodbc.so)
- iODBC Unix 驱动管理器(libiodbc.so)

1.3 ODBC 驱动管理器安装

windows 操作系统已经集成了 ODBC 驱动管理器。通过“控制面板\管理工具\数据源(ODBC)”可以打开。在 64 位 windows 操作系统中集成了 64 位和 32 位的 ODBC 驱动管理器，64 位客户端应用程序需要调用 64 位的 ODBC 驱动管理器，它位于“C:\Windows\System32\odbcad32.exe”；32 位的客户端应用程序需要调用 32 位的 ODBC 驱动管理器，它位于“C:\Windows \SysWOW64\odbcad32.exe”。

linux 操作系统下需要安装 unixODBC 或 iODBC 来使用 GBase 8a ODBC 驱动。推荐使用 unixODBC 驱动管理器。unixODBC 的安装包一般与 GBase 8a ODBC 的安装一同提供。您可以到 unixODBC 官方网站下载。然后使用如下命令安装：

```
# rpm -ivh unixODBC-2.2.14-1.x86_64.rpm

# rpm -ivh unixODBC-devel-2.2.14-1.x86_64.rpm
```

安装成功后您可以通过如下命令查看 unixODBC 的安装信息：

```
# odbcinst -j

unixODBC 2.2.14

DRIVERS.............: /etc/odbcinst.ini

SYSTEM DATA SOURCES: /etc/odbc.ini

FILE DATA SOURCES..: /etc/ODBCDataSources

USER DATA SOURCES..: /home/gbase/.odbc.ini

SQLULEN Size.......: 8

SQLLEN Size........: 8

SQLSETPOSIROW Size.: 8
```

2 GBase 8a ODBC 综述

2.1 GBase 8a ODBC 简介

GBase 8a ODBC 是 GBase 数据库的 ODBC 驱动程序，通过 GBase 8a ODBC 驱动可以访问所有 GBase 数据库。GBase 8a ODBC 支持 ODBC 3.5X 一级规范 （全部 API + 2 级特性）。GBase 8a ODBC 支持 SSL 安全连接数据传输。

注，如需 ODBC 配置 SSL 安全连接传输，需提前准备好 SSL 的 CA 证书、client 客户端秘钥和证书文件，将这些文件存放在 ODBC 数据源服务器上，供 ODBC 配置使用。SSL Server 端和客户端证书生成可参考《GBase 8a MPP Cluster 产品手册》的安全管理章节中“客户端接入认证”部分的内容。

2.2 GBase 8a ODBC 版本

GBase 8a ODBC 版本号	说明
8.3.81.53build53	支持集群的数据库连接池
8.3.81.53build52	支持集群高可用和负载均衡
8.3.81.51	稳定版

2.3 安装文件

我们提供的 ODBC 接口的 rpm 文件(linux 版本)格式如下：

gbaseodbc_<product version>_<build version>_< archtecture>.rpm。

例如：gbaseodbc_8.4_1.0_x86_64.rpm。

我们提供的 ODBC 接口的 bin 文件(window 版本)格式如下：

GBaseODBC-<product version>-<build version>-<os version and archtecture>.rar

例如：

GBaseODBC_8.3.81.53_build53.1_windows_x86.rar

2.4 GBase 8a ODBC 体系结构

GBase 8a ODBC 体系结构是基于五个组件，在下图中所示：

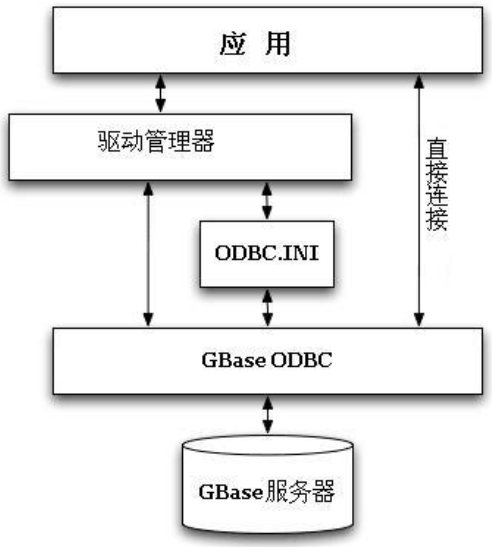

图 2- 1 GBase 8a ODBC 体系结构图

- 应用

应用是通过调用 ODBC API 实现对 GBase 数据访问的程序。应用使用标准的 ODBC 调用与驱动程序管理器通信。应用并不关心数据存储在哪里，如何存储的或者系统如何配置来访问数据。它只需要知道数据源名字（DSN）。

不管应用程序如何使用 ODBC，它们有许多共同的操作：

- 选择 GBase 服务器并连接它
- 提交 SQL 语句执行
- 获得结果(如果有的话)

- 处理错误
- 执行或回滚包含在 SQL 语句中的事务
- 断开到 GBase 服务器的连接

大多数的数据访问工作都是由 SQL 完成的，使用 ODBC 的应用主要任务就是提交 SQL 语句并获得由这些语句产生的结果。

- 驱动管理器

驱动管理器是一个管理应用与驱动程序之间通信的库。它执行下面的任务：

- 解析数据源名字（DSN）
- 驱动程序装载和卸载
- 处理 ODBC 函数调用或将它们传递给驱动程序
- GBase 8a ODBC 驱动程序

GBase 8a ODBC 驱动程序是一个实现了 ODBC API 的函数库。它处理 ODBC 函数调用，提交对 GBase 服务器的 SQL 请求，并返回结果给应用程序。如果需要，GBase 8a ODBC 驱动程序将修改客户发出的请求以便于使该请求符合 GBase 数据库支持的语法。

- ODBC.INI

ODBC.INI 是 ODBC 配置文件，该文件储存了 GBase 8a ODBC 驱动程序连接服务器和数据库的相关信息。例如：GBase 8a ODBC 驱动程序管理器通过 ODBC.INI 中的相关信息来决定装载哪个驱动程序。GBase 8a ODBC 驱动程序基于相对应的 DSN，系统使用它来读取连接参数。

- GBase 服务器

GBase 服务器是数据源。它是一个关系数据库管理系统（RDBMS）。

2.5 支持平台

GBase 8a ODBC 支持以下平台:

- Windows
- Linux 操作系统
- Unix

3 GBase 8a ODBC 使用

本节描述了如何安装 GBase 8a ODBC 驱动，创建 GBase 8a ODBC 数据源，以及创建 GBase 8a ODBC 连接字符串。

3.1 安装 GBase 8a ODBC 驱动

3.1.1 windows 平台上安装 GBase 8a ODBC

首先，执行安装包，如 GBaseODBC_8.3.81.53_build53.11_W64.exe。其中，安装路径要选择没有中文的路径。

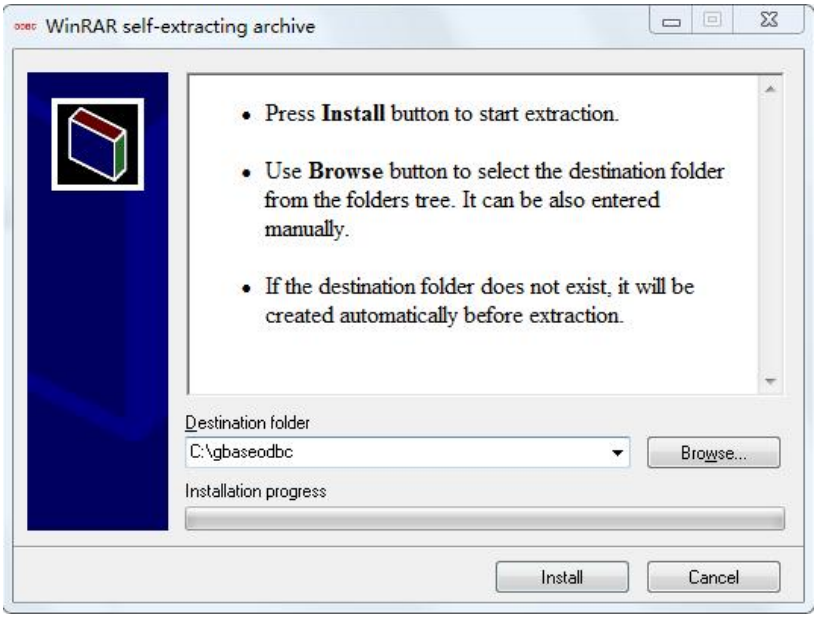

图 3- 1 执行安装包

安装完成后，使用管理员权限打开一个命令行窗口。在命令行下切换到安装目录下，执行 Install.bat 脚本。

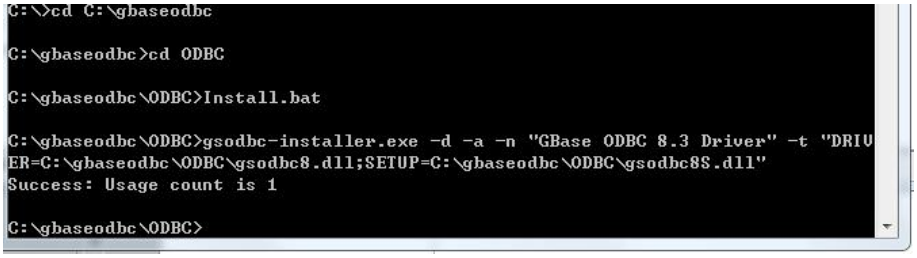

图 3- 2 命令行窗口下执行 Install.bat 脚本

打开数据源管理器，选择“驱动程序”选项卡，可以看到 GBase 8a ODBC 驱动信息，如下图所示。

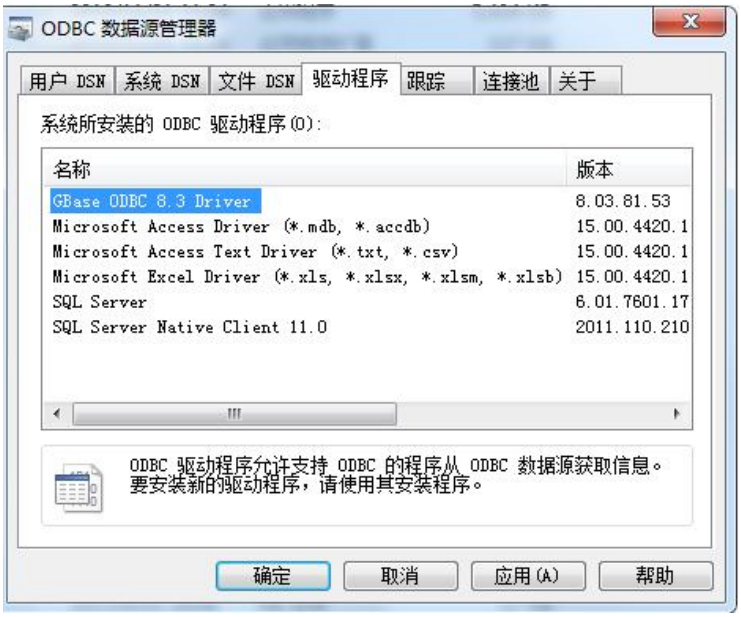

图 3- 3 安装 GBase 8a ODBC 驱动

注：在 64 位系统下，64 位的数据源管理器路径：C:\Windows\System32\odbcad32.exe；32 位的数据源管理器路径：C:\Windows\SysWOW64\odbcad32.exe。

3.1.2 linux 平台上安装 GBase 8a ODBC

获取 ODBC 安装包 gbaseodbc_<product version>_<build version>_<archtecture>.rpm 复制到文件系统的某个目录下，进入该目录。

使用 root 用户执行如下命令进行安装：

```
# rpm -ivh gbaseodbc_8.4_1.0_x86_64.rpm
```

安装包 rpm 包后，会自动在数据源管理器中注册驱动。注册信息如下：

```
cat /etc/odbcinst.ini

[GBase ODBC 8.4 Driver]

Driver       =/usr/lib64/libgsodbc8.so

UsageCount   = 1

DontDLClose  = 1

Threading    = 0
```

3.1.3 AIX 平台上安装 GBase 8a ODBC

- 安装 unixODBC 数据源管理器

使用 root 用户执行 unixODBC-2.3.0-AIX5.3.bin 命令，执行后会将 unixODBC 安装到/usr 目录下（/usr/bin, /usr/lib）；

unixODBC 文件列表如下：

```
├──bin
|       dltest
|       isql
|       iusql
|       odbcinst
|       odbc_config
|
```

```
└──lib
        libgbase.so
        libgbaseclient.a
        libgbaseclient.so
        libgbaseclient_r.so
        libgsodbc8-8.3.81.51.a
        libgsodbc8-8.3.81.51.so
        libgsodbc8.a
        libgsodbc8.la
        libgsodbc8.so
        libltdl.a
        libltdl.la
        libodbc.a
        libodbc.la
        libodbccr.a
        libodbccr.la
```

- 安装 GBaseODBC 驱动

使用 root 用户执行 GBaseODBC_8.3.81.53_build53.5_AIX5.3_64bit.bin 命令，执行后会将 GBase ODBC 驱动安装到/usr/lib 目录下。

GBaseODBC 的文件列表如下:

libgbase.so

libgbaseclient.a

libgbaseclient.so

libgbaseclient_r.so

libgsodbc8-8.3.81.51.a

libgsodbc8-8.3.81.51.so

libgsodbc8.a

libgsodbc8.la

libgsodbc8.so

- 在 unixODBC 数据源管理器中注册 GBase ODBC 驱动

  在/etc/odbcinst.ini 文件中增加如下内容:

```
cat /etc/odbcinst.ini
 [GBase ODBC 8.3 Driver]
Driver=/usr/lib/libgsodbc8.so
UsageCount  = 1
DontDLClose  = 1
Threading  = 0
```

3.2 创建 GBase 8a ODBC 数据源名

“数据源”是一个数据来源的地方。数据源必须有一个持久固定的标识符，就是数据源名字。使用数据源名字，GBase 能访问初始化信息。有了这个初始化信息，GBase 就知道在什么地方访问数据库和当访问开始时使用什么设置。

数据源是访问数据的有效路径。它标识了一个运行的 GBase 服务器，以及在连接时服务器的缺省数据库，和必要的连接信息，比如端口。

3.2.1 windows 平台上创建数据源

在 Windows 中数据源信息可能存在于两个地方：在 Windows 注册表中（对 Windows 系统），或在一个 DSN 文件中（对任何系统）。

如果信息在 Windows 注册表中，它叫做“机器数据源”。它可能是一个“用户数据源”，在这种情况只有一个用户可以看见它。或者它可能是一个“系统数据源”，在这种情况，计算机上的所有用户或所有连接到计算机的用户都能

访问。当用户运行 ODBC 数据源管理器时，用户可以选择是使用“用户数据源”还是“系统数据源”。

如果信息在 DSN 文件中，它叫做“文件数据源”。这是一个文本文件。它的优点是：（a）它对任何类型的计算机都是可用的，不是对带 Windows 操作系统的计算机而言；(b)它的内容可以相对容易的转移或拷贝。ODBC 数据源管理器可以更新用户的数据源连接信息。当用户添加数据源时，ODBC 数据源管理器为用户更新注册表信息。

创建数据源的步骤如下所示：

1）打开控制面板。

2）双击管理工具，然后双击数据源(ODBC)。这时出现 ODBC 数据源管理对话框，如下所示：

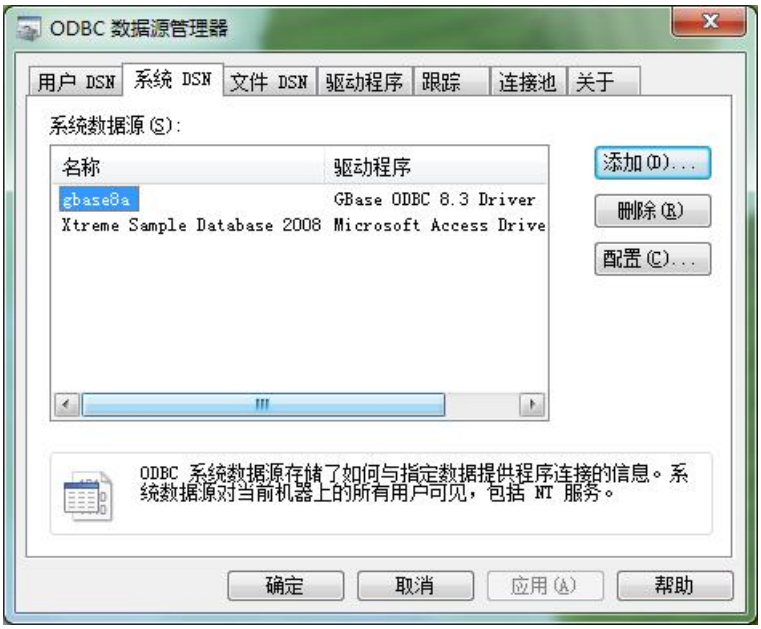

图 3- 4 ODBC 数据源管理对话框

3）如果注册用户数据源，选择“用户 DSN”属性页；如果注册系统数据源，选择“系统 DSN” 属性页，建议使用“系统 DSN” 属性页；

4）在 ODBC 数据源管理器对话框中，单击添加。创建新数据源对话框出现。

图 3- 5 创建新数据源对话框

5) 选择 GBase ODBC 8.3 Driver 驱动程序，然后点击完成。GBase 8a ODBC 驱动程序- DSN 配置对话框出现，如下所示

图 3- 6 GBase 8a ODBC 驱动程序- DSN 配置对话框

6）在数据源名称框中，输入用户想用来访问的数据源名字。可以是用户选择的任何有效名字。

7）在描述框中，输入对 DSN 的描述。

8）在服务器名称框，输入用户要访问的 GBase 服务器主机的 IP 地址。缺省的，它是 localhost。

9）在用户框中，输入用户的 GBase 数据库用户名。

10）在密码框中，输入用户的密码。

11）在数据库框中，输入用户想作为缺省数据库使用的 GBase 数据库名字。

12）在连接选项标签中的端口框中，输入端口号（GBase 的缺省段口号为 5258）。

13）点击确定来完成这个数据源的添加。

注意：当点击了确定，数据源管理对话框出现，ODBC 管理器更新注册表信息。当用户连接它时，用户输入的用户名和连接字符串变成了这个数据源缺省的连接值。

用户也可以测试是否用户的设置对连接服务器合适，通过使用测试数据源按钮。成功的测试出现下面的窗口：

图 3- 7 GBase 8a ODBC 驱动程序测试成功对话框

测试失败出现类似下面的错误提示：

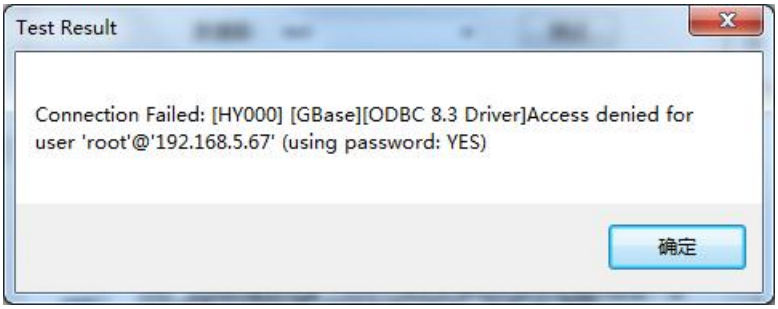

图 3- 8 GBase 8a ODBC 驱动程序测试失败对话框

可以点击“高级”选项，来进行一些高级设置，DSN 配置对话框页有选项按钮。如果用户选择它，会出现下面的选项对话框，显示控制驱动程序行为。关于这些选项的意义请参考连接参数。如下图所示：

图 3- 9 GBase 8a ODBC 驱动程序- DSN 高级配置对话框

在 Windows 中修改一个数据源；

- 打开 ODBC 数据源管理器。单击适当的 DSN 标签；
- 选择用户想修改的 GBase 数据源，然后点击配置。会出现 GBase 8a ODBC Driver 驱动程序- DSN 配置对话框；
- 修改适当的数据源域，然后点击确定。

- 当用户在这个对话框中完成修改后，ODBC 管理器会更新注册信息。

3.2.2 windows 平台数据源 SSL 配置说明

在 Windows 中创建数据源配置 SSL 信息，需要创建文件 DSN，

打开控制面板，双击管理工具，然后双击数据源(ODBC)，出现 ODBC 数据源管理对话框，如图所示选择文件 DSN。

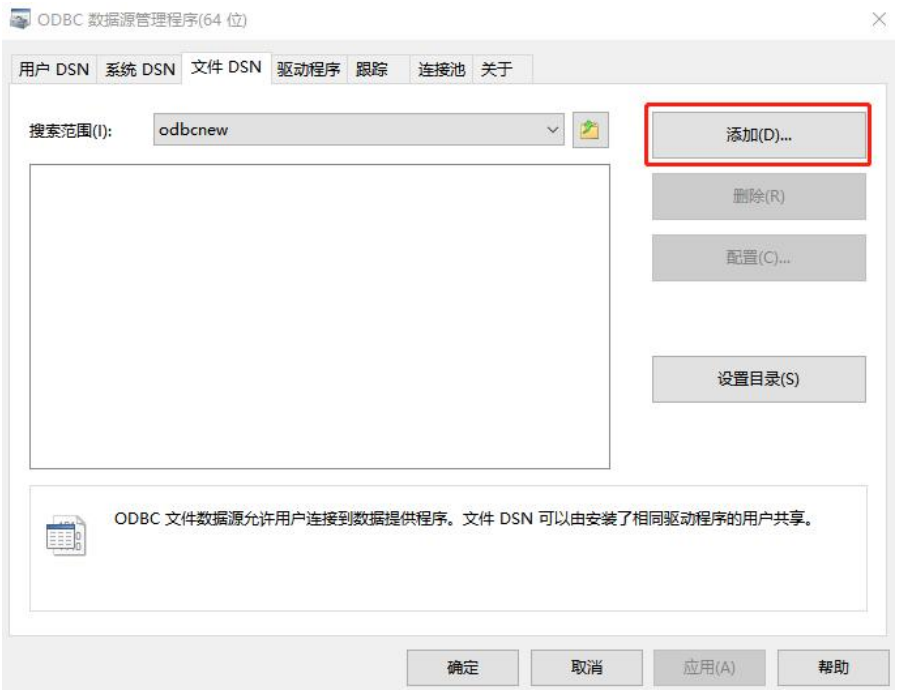

图 3- 10 GBase 8a ODBC 驱动程序- 文件 DSN 配置页

选择添加按钮，出现如下界面：

图 3- 11 GBase 8a ODBC 驱动程序- 驱动选择页

选择 GBase ODBC 驱动，单击高级按钮，在高级文件 DSN 设置页面中，添加 SSL 相关配置信息，如图。

具体的 CA 证书、client 端秘钥和证书需提前生成并已存放在 client 端。下图页面中填写的路径需对应实际环境证书和秘钥的存放路径。

图 3- 12 GBase 8a ODBC 驱动程序- 高级文件 DSN 创建对话框

确定后，继续下一步，输入文件 DSN 的路径和名字，如下图：

图 3- 13 GBase 8a ODBC 驱动程序- 创建数据源对话框

最后完成 DSN 文件的创建。

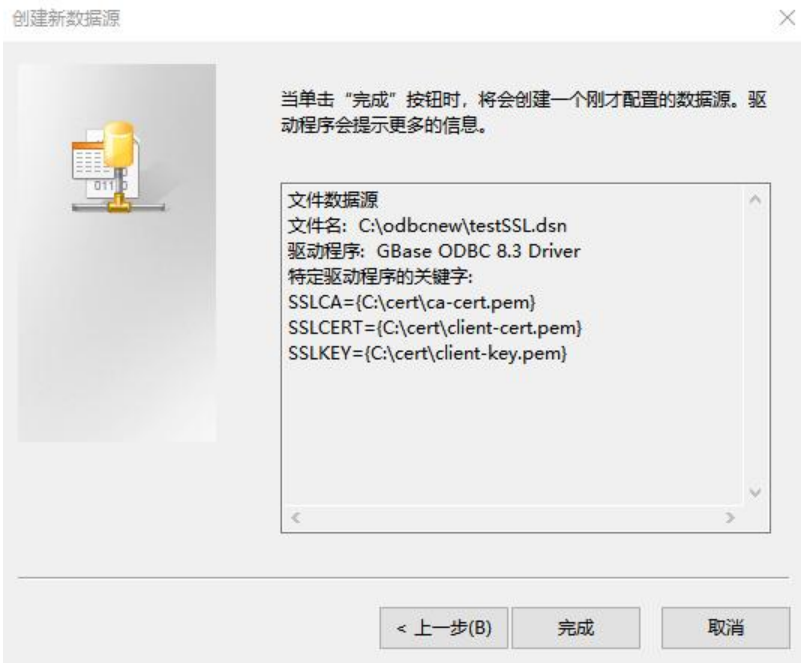

图 3- 14 GBase 8a ODBC 驱动程序- 创建数据源对话框

文件创建完成后，会弹出数据源配置界面，可以按照上一节“windwos 平台上创建数据源”所示，填写服务器名称、用户、密码、数据库等相应配置项，并测试连接成功与否。

图 3- 15 GBase 8a ODBC 驱动程序- 数据源配置对话框

3.2.3 linux 平台上创建数据源

在 linux 上创建数据源有两种方式，一种是修改 unixODBC 的配置文件 odbc.ini，另一种是使用 gsodbc-installer 命令。

3.2.3.1 修改 odbc.ini 配置文件创建数据源

使用 unixODBC 提供的命令“odbcinst -j”可以查找到 odbc.ini 的路径。如下所示：

```
# odbcinst -j
```

unixODBC 2.2.14

DRIVERS.....: /etc/odbcinst.ini

SYSTEM DATA SOURCES: /etc/odbc.ini

FILE DATA SOURCES..: /etc/ODBCDataSources

USER DATA SOURCES..: /root/.odbc.ini

SQLULEN Size.....: 8

SQLLEN Size.....: 8

SQLSETPOSIROW Size.: 8

在/etc/odbc.ini 文件中加入:

[test]

Driver = GBase 8a ODBC 8.3 Driver

DATABASE = test

DESCRIPTION = GBase 8a ODBC 8.3 Driver DSN

SERVER = 192.168.111.96

UID = root

PASSWORD = 1

3.2.3.2 使用 gsodbc-installer 创建数据源

使用 gsodbc-installer 创建数据源时要用到 GBase 8a ODBC 连接字符串。有关 GBase 8a ODBC 连接字符串的相关内容请参考 3.4 小节。使用 gsodbc-installer 创建数据源的命令如下:

#gsodbc-installer -s -a -c2 -n "test" -t "DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=

gbase20110531; SERVER={192.168.111.96};"

3.2.4 Linux 平台数据源 SSL 配置说明

Linux 平台上数据源配置 SSL，需在 odbc.ini 文件中添加如下参数：

vi /etc/odbc.ini：

[test]

Driver = GBase 8a ODBC 8.3 Driver

DATABASE = test

DESCRIPTION = GBase 8a ODBC 8.3 Driver DSN

SERVER = 192.168.111.96

UID = root

PASSWORD = 1

SSLCA=/usr/local/ssl/ca-cert.pem

SSLCERT=/usr/local/ssl/client-cert.pem

SSLKEY=/usr/local/ssl/client-key.pem

注，具体的 CA 证书、client 端秘钥和证书需提前生成并已存放在 ODBC client 端。示例中填写的路径需对应实际环境证书和秘钥的存放路径。

3.2.5 AIX 平台上创建数据源

创建 GBase ODBC 数据源， 在/etc/odbc.ini 文件中增加如下内容：

[test]

Driver=GBase ODBC 8.3 Driver

SERVER=192.168.7.235

```
UID=gbase

PWD=gbase20110531

PORT=5258

DATABASE=test
```

测试创建的数据源连接性，可执行如下命令：

```
# isql test -v
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
```

3.3 GBase 8a ODBC 连接参数

用户可以在 ODBC.INI 文件的[Data Source Name]部分为 GBase 8a ODBC 指明下面的参数或者通过在 SQLDriverConnect()调用中使用 InConnectionString 参数。

参数	默认值	注释
SERVER	localhost	GBase 服务器的主机名或 IP 地址

参数	默认值	注释
UID/USER	空	连接 GBase 服务器的用户名
PWD/ PASSWORD	空	连接 GBase 服务器的密码
DATABASE	空	数据库名称
PORT	5258	连接 GBase 服务器的端口号

在 Windows 中，用户可以通过数据源配置界面中的高级选项选择复选框来配置 GBase 8a ODBC。同时在通过连接字符串使用 GBase 8a ODBC 时，可以使用如下表列出的关键字进行 GBase 8a ODBC 配置。在 linux 平台下使用这些参数时，用户可以将如下参数写入 odbc.ini 文件中并设置该参数值为 1 来使用如下表列出的关键字进行 GBase 8a ODBC 配置。

参数	默认值	注释
连接		
COMPRESSED_PROTO	0	使用压缩协议
AUTO_RECONNECT	0	启动自动重连
MULTI_STATEMENTS	0	允许多 statements
IP_ROUTE	0	使用集群高可用性
CONNECTION_BALANCE	0	使用集群负载均衡
GCLUSTER_ID	空	使用连接负载均衡时需要指定集群 ID
CHECK_INTERVAL	60	节点检测间隔（秒）
CHARSET	utf8	连接字符集
INITSTMT	空	在连接到 server 时执行的语句
NO_PROMPT	0	连接时不产生提示信息
源数据		
NO_BIGINT	0	将 BIGINT 列转换为 INT 类型
NO_BINARY_RESULT	0	将二进制函数结果作为字符型处理
FULL_COLUMN_NAMES	0	SQLDescribeCol()返回含表名
NO_CATALOG	0	不产生目录结果

参数	默认值	注释
游标/结果集		
DYNAMIC_CURSOR	0	使用动态游标
FORWARD_CURSOR	0	强制使用前向游标
AUTO_IS_NULL	0	SQL_AUTO_IS_NULL 生效
ZERO_DATE_TO_MIN	0	由 0 组成的 DATETIME 值视为最小的时间
NO_CACHE	0	不缓存结果(只用于前向游标)
UPDATE_DELETE_LIMIT	0	设置为 1 时，标识数据库支持 update/delete limit 语法。连接 8a 或 8a Cluster 时设置该选项为 0。
FETCH_SIZE	0	设置为大于 0 时,每次从数据库端获取 FETCH_SIZE 行的数据。此参数只适用于单机。
调试		
LOG_QUERY	0	允许将查询记录到 gsodbc.sql
其他		
SAFE	0	安全
USE_MYCNF	0	从配置文件读取设置
NO_TRANSACTIONS	0	不支持事务
连接池		
POOL_INIT_SIZE	0	连接池初始化时缓存的连接数
POOL_MAX_ACTIVE_SIZE	0	连接池中最多能够保存的连接数
POOL_MAX_IDLE	0	连接池中最多存放的空闲连接数
POOL_MIN_IDLE	0	连接池中最少存放的空闲连接数
POOL_IDLE_LIFE	0	连接池中空闲连接的最大生存时间

参数	默认值	注释
POOL_USED_LIFE	0	连接池中借出的连接最大生存时间
POOL_CHECKOUT_TIMEOUT	500	从连接池中再申请连接需要等待的时间，单位毫秒
POOL_TEST_INVALID_CONN_PERIOD	20	清理无效连接运行周期，单位秒
POOL_LBS	0	连接池提供的连接分发策略(负载均衡策略) 0：轮询， 1：最小空闲连接数优先
POOL_TEST_BORROW	0	从连接池中借出连接时,连接池测试连接是否可用
POOL_TEST_RETURN	0	连接返回连接池时,测试连接是否可用
POOL_MANAGER	0	启动连接池管理模块(包括:清理无效连接,维护连接池空闲连接数和清理过期连接)
POOL_KEEP_IDLE	0	启动连接池空闲连接维护功能
POOL_CLEAR_OVERDUE	0	启动连接池清理过期连接功能
POOL_KEEP_IDLE_PERIOD	60	维护空闲连接运行周期,单位秒
POOL_CLEAN_OVERDUE_PERIOD	60	清理过期连接运行周期,单位秒
CACHE_INSERT_VALUES	0	设置为 1 时，开启本功能。当用户 insert 数据时，设置 SQL_ATTR_PARAMSET_SIZE 时，ODBC 驱动会拼成批量 insert values 向集群插入数据。具体使用请参数用例。

3.4 创建 GBase 8a ODBC 连接字符串

用户可以通过 SQLDriverConnect 使用连接字符串连接 GBase 服务器。GBase

8a ODBC 8.3 的连接字符串格式如下：

```
ConnectionString = "DRIVER={GBase 8a ODBC 8.3 Driver}; \
SERVER={ 192.168.111.96}; \
DATABASE=gbase; \
UID=gbase; \
PWD=111111;
```

如果用户的编程语言将空白后的反斜杠转换成空格，更好的方法是指明连接字符串为一个单一的长字符串，或者使用多个字符串的连接，那样不会在其中增加空格。例如：

```
ConnectionString = "DRIVER={GBase 8a ODBC 8.3 Driver};"
"SERVER={ 192.168.111.96};"
"DATABASE=gbase;"
"UID= gbase;"
"PWD=111111;"
```

4 GBase 8a ODBC 高可用功能

4.1 高可用性简介

GBase 8a 为扁平架构集群，考虑到集群的高可用性，当某个节点不可用时，应该把到来的连接请求路由到另外一个可用的节点上。高可用就是来实现这样一个功能。

4.2 高可用性使用方法

使用 GBase 8a ODBC 连接 GBase 8a 数据库有两种方式，分别是：配置并使用数据源名称和使用连接字符串。

在 windows 平台下配置数据源名称时，需要在“服务器名称”一栏中填写 GBase 8a 数据库所有节点的 IP 地址，并使用“;”分割符将其分割开来（如下图所示），并选择“连接”选项卡中的“高可用性”选项。GBase 8a ODBC 的集群 IP 自动路由功能就会开启。除非数据库集群的所有节点都不可用，否则 GBase 8a ODBC 总会将连接路由到数据库群集中的一个可用的节点上。

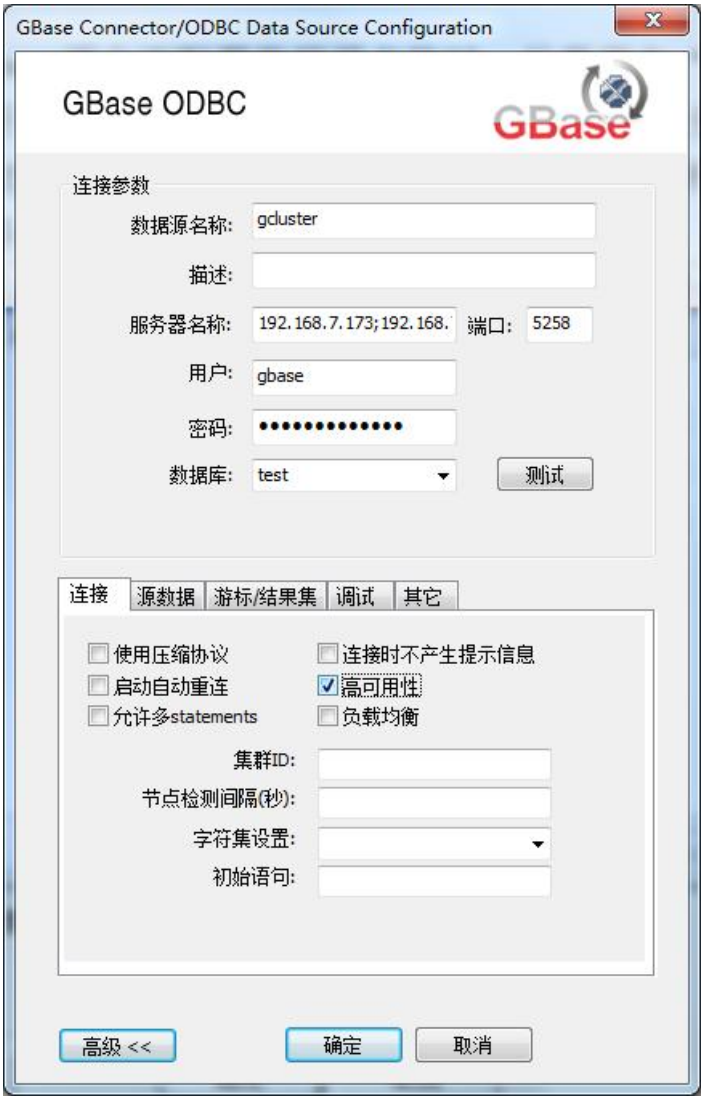

图 4- 1 GBase 8a ODBC IP 路由配置

在 linux 平台上配置数据源名称时，修改 odbc.ini 文件中“SERVER”变量的值为 GBase 8a 数据库所有节点的 IP 地址，使用“;”分割符将其分割，并增加 IP_ROUTE=1 选项。GBase 8a ODBC 的高可用功能就会开启。

```
[ODBC Data Sources]

test = GBase 8a ODBC
```

```
[test]
Driver          = GBase 8a ODBC 8.3 Driver
DATABASE        = test
DESCRIPTION     = GBase 8a ODBC 8.3 Driver test
SERVER          = 192.168.7.172;192.168.7.173;192.168.7.174
UID             = root
PASSWORD        = 1
```

使用 GBase 8a ODBC 连接字符串时，需要将数据库集群所有节点 IP 列在 SERVER=之后的大括号中，并使用“;”分割开，并增加 IP_ROUTE=1;。

```
"DRIVER={GBase 8a ODBC 8.3 Driver};"
"SERVER={192.168.5.65;192.168.5.64};"
"UID=root;PWD=1;DATABASE=test;PORT=5258;IP_ROUTE=1;"
```

在 linux 下使用 GBase 8a ODBC 连接字符串连接时需要在 odbcinst.ini 文件中写入如下信息:

```
[GBase 8a ODBC 8.3 Driver]
Description = GBase 8a ODBC
Driver = /usr/lib/libgsodbc8.so
Setup =
FileUsage =1
Threading =0
DontDLClose =1
```

5 GBase 8a ODBC 负载均衡功能

5.1 负载均衡简介

GBase 8a ODBC 提供的高可用负载均衡功能是指，GBase 8a ODBC 会将客户端请求的数据库集群连接平均分摊到集群所有可用的节点上。

5.2 负载均衡的使用方法

GBase 8a ODBC 提供两种方式来使用高可用负载均衡。一种是配置数据源，另一种是使用连接字符串。

如果通过配置数据源使用 GBase 8a ODBC 的高可用负载均衡，在 windows 平台下需要使用 ODBC 数据源管理器打开 GBase 8a ODBC 的数据源配置界面进行配置。如下图所示:

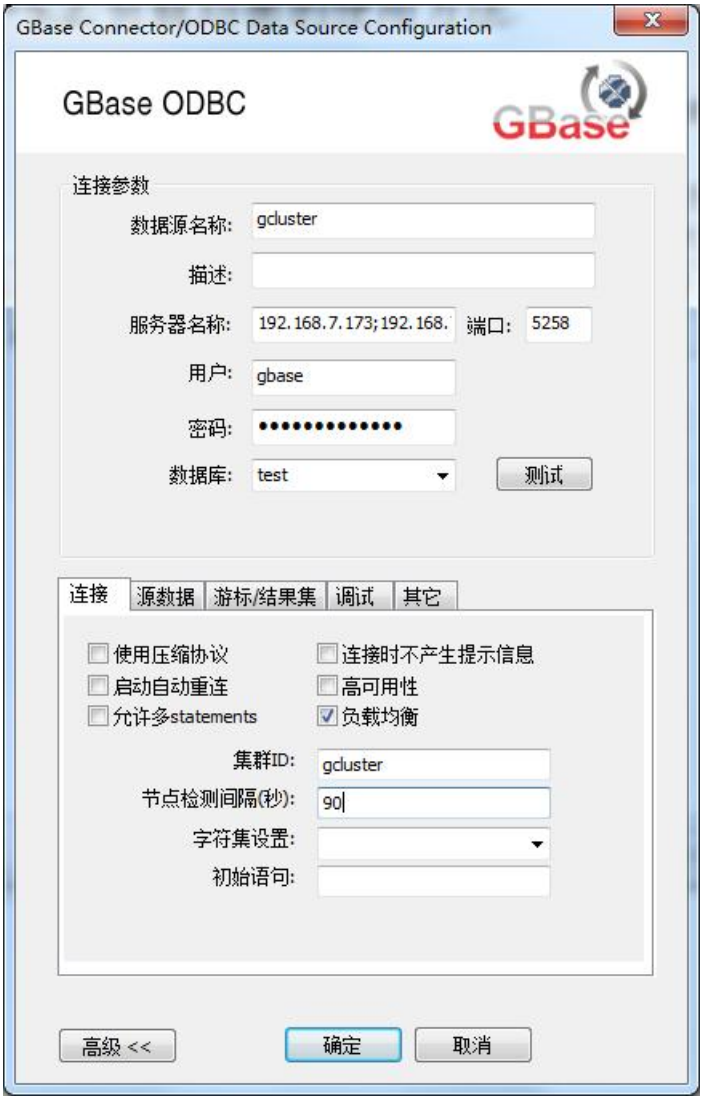

图 5- 1 GBase 8a ODBC 高可用负载均衡配置

首先要在“服务器名称”文本框中输入集群所有节点的 IP 地址，IP 地址间使用“;”进行分割；其次要打开“高级”选项，选择“连接”选项卡中的“负载均衡”选项，并指定“集群 ID”的值，“节点检测间隔（秒）”是使用高可用负载均衡所需要的，它的默认值为 60 秒；最后点击“确定”，一个带有高可用负载均衡的 GBase 8a ODBC 数据源就配置好了。

在 Linux 平台下则需要修改 odbcinst.ini 和 odbc.ini 数据源配置文件。

通常情况下需要将如下内容添加到/etc/odbcinst.ini 和/etc/odbc.ini 的文件尾部。

在/etc/odbcinst.ini 文件中加入:

```
[GBase 8a ODBC 8.3 Driver]
Description = GBase 8a ODBC
Driver      = /usr/lib/libgsodbc8.so
FileUsage   = 1
threading   = 0
DontDLClose = 1
```

在/etc/odbc.ini 文件中加入:

```
[test]
Driver = GBase 8a ODBC 8.3 Driver
DATABASE = test
DESCRIPTION = GBase 8a ODBC 8.3 Driver DSN
SERVER = 192.168.111.96;192.168.5.212;192.168.7.174
UID = gbase
PASSWORD = gbase20110531
CONNECTION_BALANCE = 1
GCLUSTER_ID = gcluster
CHECK_INTERVAL = 90
```

如果通过连接字符串使用 GBase 8a ODBC 的高可用负载均衡，请参考如下示例。

```
"DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=gbase20110531;"
```

"SERVER={192.168.111.96;192.168.5.212;192.168.7.174;192.168.7.173};"

"CONNECTION_BALANCE=1;GCLUSTER_ID=gcluster;"

"CHECK_INTERVAL=90;"

如果在 linux 平台下使用连接字符串,需要在/etc/odbcinst.ini 增加往下内容。

```
[GBase 8a ODBC 8.3 Driver]
Description = GBase 8a ODBC
Driver      = /usr/lib/libgsodbc8.so
FileUsage   = 1
threading   = 0
DontDLClose = 1
```

6 GBase 8a ODBC 连接池

6.1 GBase 8a ODBC 连接池简介

GBase 8a ODBC 连接池是针对 GBase 8a 开发的 ODBC 数据库连接池。它主要有用下几大功能:

- 缓存连接到 GBase 8a 各节点的数据库连接。
- 负载功能，根据负载均衡策略将用户申请的连接分摊到 GBase 8a 的各节点上。
- 周期性的维护 GBase 8a ODBC 连接池中各连接的有效性，清理连接池中无效的连接。
- 周期性的维护连接池中的空闲连接数，保证连接池中的空闲连接数满足用户的设定最大/最小空闲连接数。
- 清理过期的连接。

6.2 GBase 8a ODBC 连接池使用方法

本节以连接字符串为例来展示 GBase 8a ODBC 连接池的使用方法。主要描述关于连接池的关键字使用方法。在 linux 平台下创建数据源时，同样可以将这些关键字写入到 odbc.ini 文件中来使用 GBase 8a ODBC 连接池功能。或者使用 gsodbc-installer -s -a -c2 -n "test" 加上连接字符串来注册一个带连接池的数据源，如下所示:

```
#gsodbc-installer -s -a -c2 -n "test" -t "DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=

gbase20110531; SERVER=192.168.111.96;"
```

6.2.1 开启 GBase 8a ODBC 连接池

通过连接字符串开启 GBase 8a ODBC 连接池时，首先要设置 GCLUSTER_ID 的值，然后设置 POOL_MAX_ACTIVE_SIZE, POOL_MAX_IDLE 的值大于 0，且 POOL_MAX_ACTIVE_SIZE 的值大于等于 POOL_MAX_IDLE 的值。如下所示：

```
"DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=gbase20110531;"
"SERVER={192.168.111.96;192.168.5.212;192.168.7.174;192.168.7.173};"
"GCLUSTER_ID=gcluster; POOL_MAX_ACTIVE_SIZE=80;POOL_MAX_IDLE=60;"
```

6.2.2 设置连接池初始化大小

如果需要在连接初始时让连接池自动创建一定数量的连接，可以通过设置 POOL_INIT_SIZE 的值来实现。通常 POOL_INIT_SIZE 的值小于等于 POOL_MAX_IDLE。如下所示：

```
"DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=gbase20110531;"
"SERVER={192.168.111.96;192.168.5.212;192.168.7.174;192.168.7.173};"
"GCLUSTER_ID=gcluster; POOL_MAX_ACTIVE_SIZE=80;POOL_MAX_IDLE=60;"
"POOL_INIT_SIZE=10;"
```

6.2.3 设置获取连接超时时间

如果需要限制从连接池获取连接的时间，可以通过设置 POOL_CHECKOUT_TIMEOUT 的值来实现。默认情况下 POOL_CHECKOUT_TIMEOUT 值为 500 毫秒。如下所示：

```
"DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=gbase20110531;"
```

"SERVER={192.168.111.96;192.168.5.212;192.168.7.174;192.168.7.173};"

"GCLUSTER_ID=gcluster; POOL_MAX_ACTIVE_SIZE=80;POOL_MAX_IDLE=60;"

"POOL_CHECKOUT_TIMEOUT=2000;"

6.2.4 设置获取连接时使用的负载均衡策略

GBase 8a ODBC 连接池提供了两种负载均衡策略，分别是轮询和最小 BUSY 连接数优先。使用轮询策略时设置 POOL_LBS=0，使用最小 BUSY 连接数优先策略时设置 POOL_LBS=1。默认时使用轮询策略。如下所示：

"DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=gbase20110531;"

"SERVER={192.168.111.96;192.168.5.212;192.168.7.174;192.168.7.173};"

"GCLUSTER_ID=gcluster; POOL_MAX_ACTIVE_SIZE=80;POOL_MAX_IDLE=60;"

"POOL_LBS=1;"

6.2.5 借还连接时测试连接可用性

从连接池中借出连接时对连接进行测试进一步保证了借出连接的效性，归还连接时对连接进行测试时保证了无效的连接不会进入到连接池中。借出连接时检测需要设置 POOL_TEST_BORROW=1，归还连接时检测需要设置 POOL_TEST_RETURN=1。如下所示：

"DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=gbase20110531;"

"SERVER={192.168.111.96;192.168.5.212;192.168.7.174;192.168.7.173};"

"GCLUSTER_ID=gcluster; POOL_MAX_ACTIVE_SIZE=80;POOL_MAX_IDLE=60;"

" POOL_TEST_BORROW=1; POOL_TEST_RETURN=1;"

6.2.6 清理无效连接

如果要连接池定期清理连接池中无效的连接，那么就要设置 POOL_MANAGER=1。清理无效连接功能不仅定期清理连接池中无效的连接，还定期检测 GBase 8a 各节点的状态。如果 GBase 8a 有节点恢复可用，GBase 8a ODBC 连接池将会缓存连接到该节点的 ODBC 连接。同时通过设置 POOL_TEST_INVALID_CONN_PERIOD 的值可以设置清理无效连接的周期。该周期默认值为 20 秒。如下所示：

```
"DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=gbase20110531;"
"SERVER={192.168.111.96;192.168.5.212;192.168.7.174;192.168.7.173};"
"GCLUSTER_ID=gcluster; POOL_MAX_ACTIVE_SIZE=80;POOL_MAX_IDLE=60;"
" POOL_MANAGER=1;POOL_TEST_INVALID_CONN_PERIOD=30;"
```

6.2.7 维持空闲连接数

如果要使连接池中的空闲连接数维持在一范围内，那么可以通过设置 POOL_KEEP_IDLE=1 打开连接池的空闲连接数维持功能，并设置 POOL_MIN_IDLE 的值来实现。POOL_MIN_IDLE 的值小于 POOL_MAX_IDLE 的值。这样当连接池发现空闲连接数小于 POOL_MIN_IDLE 的值时就会创建新的空闲连接，以满足当前空闲连接数大于等于 POOL_MIN_IDLE 的值。打开维持空闲连接数功能必须同时设置 POOL_MANAGER=1。如下所示：

```
"DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=gbase20110531;"
"SERVER={192.168.111.96;192.168.5.212;192.168.7.174;192.168.7.173};"
"GCLUSTER_ID=gcluster; POOL_MAX_ACTIVE_SIZE=80;POOL_MAX_IDLE=60;"
" POOL_MANAGER=1;POOL_TEST_INVALID_CONN_PERIOD=30;"
```

" POOL_KEEP_IDLE=1; POOL_MIN_IDLE=30;"

6.2.8 清理过期连接

用户可以给 GBase 8a ODBC 连接池中的连接设置有效期，并且当连接池发现有连接过期时，会将该连接释放掉。这种功能可以通过设置 POOL_CLEAR_OVERDUE, POOL_USED_LIFE 和 POOL_IDLE_LIFE 来打开。如果 POOL_USED_LIFE 的值为 0，那么已借出的连接不会过期，POOL_USED_LIFE 默认值为 0;如果 POOL_IDLE_LIFE 的值为 0,那么空闲连接不会过期,POOL_IDLE_LIFE 默认值为 0。打开清理过期连接功能必须同时设置 POOL_MANAGER=1 如下所示:

"DRIVER=GBase 8a ODBC 8.3 Driver;UID=gbase;PWD=gbase20110531;"

"SERVER={192.168.111.96;192.168.5.212;192.168.7.174;192.168.7.173};"

"GCLUSTER_ID=gcluster; POOL_MAX_ACTIVE_SIZE=80;POOL_MAX_IDLE=60;"

" POOL_MANAGER=1;POOL_TEST_INVALID_CONN_PERIOD=30;"

" POOL_CLEAR_OVERDUE=1; POOL_USED_LIFE=360000; POOL_IDLE_LIFE=36000;"

6.2.9 释放连接池

释放连接池需要按照如下方式调研释放连接池的接口。

```
rc = SQLDriverConnect(hdbc, NULL, connStr, SQL_NTS,NULL, 0, NULL, 0);
if (!isSuc(rc))
{
    getError(SQL_HANDLE_DBC, hdbc);
```

```
        return;
    }
    SQLSetConnectAttr(hdbc, SQL_ATTR_GBASE_POOL_FREE,
SQL_FREE_GBASE_POOL, NULL);
    SQLDisconnect(hdbc);
```

在归还从连接池借出的连接前，先设置该连接的属性 SQL_ATTR_GBASE_POOL_FREE 的值为 SQL_FREE_GBASE_POOL，然后调用 SQLDisconnect 归还连接。这样 ODBC 在归还连接的同时会释放掉该连接所在的连接池。

7 GBase 8a ODBC 应用开发

将 GBase 8a ODBC 应用程序与 GBase 服务器结合起来包括下面的操作：

- 配置 GBase 8a ODBC DSN
- 连接到 GBase server
- 初始化操作
- 执行 SQL 语句
- 重新得到结果
- 执行事务
- 与服务器断开连接

大多数的应用程序使用这些步骤的一些变化方式。基本的应用程序步骤显示在下面的图中：

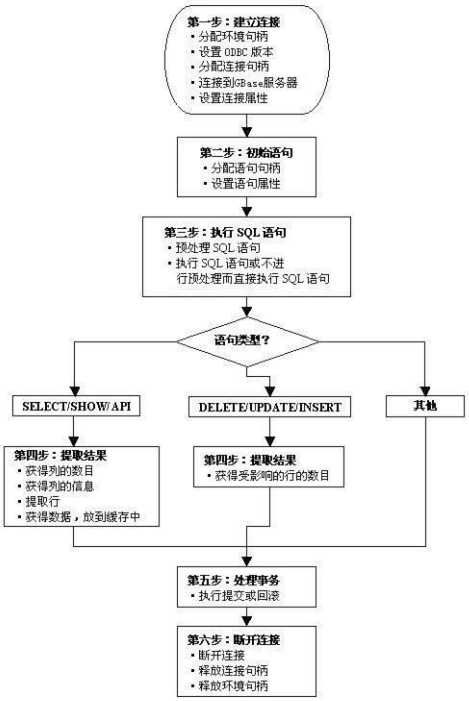

图 7- 1 GBase 8a ODBC 接口应用步骤

7.1 GBase 8a ODBC API 介绍

本节总结了 ODBC 程序，按照功能对它们进行了分类。

对于完整的 ODBC API 参考，请参考微软公司技术支持网站中“ODBC 程序员参考”相关内容。

一个应用程序可以调用 SQLGetInfo 函数来获取关于 GBase 8a ODBC 的一致信息。要获取关于在驱动中支持的特定函数的信息，应用程序可以调用 SQLGetFunctions。

注意：对于向后兼容的功能上，GBase 8a ODBC Driver 驱动支持所有不推荐的函数。下面我们将按照功能分类列出常用的 GBase 8a ODBC API 函数。

连接到一个数据源，如下表格所示：

函数名	GBase ODBC	一致性	目 的
SQLAllocHandle	是	ISO 92	获得一个环境、连接、语句、或者描述符句柄。
SQLConnect	是	ISO 92	通过数据源名、用户 ID 和密码连接到一个指定的驱动。
SQLDriverConnect	是	ODBC	通过连接字符串，或者驱动管理器和用户使用的驱动显示连接对话框的请求来连接到指定驱动。
SQLAllocEnv	是	不推荐	获得一个从驱动分配的环境句柄。
SQLAllocConnect	是	不推荐	获得一个连接句柄。

获得关于驱动和数据源的信息，如下表格所示：

函数名	GBase ODBC	一致性	目 的
SQLDataSources	否	ISO 92	返回可用的的数据源列表，由驱动管理器执行。
SQLDrivers	否	ODBC	返回已经安装的驱动和它们的属性，由驱动管理器执行。
SQLGetInfo	是	ISO 92	返回指定驱动和数据源的信息。
SQLGetFunctions	是	ISO 92	返回支持的驱动函数。
SQLGetTypeInfo	是	ISO 92	返回支持的数据类型信息。

设置和获得驱动属性，如下表格所示：

函数名	GBase ODBC	一致性	目 的
SQLSetConnectAttr	是	ISO 92	设置一个连接的属性。
SQLGetConnectAttr	是	ISO 92	返回一个连接的属性值。
SQLSetConnectOption	是	不推荐	设置一个连接选项。
SQLGetConnectOption	是	不推荐	返回一个连接的选项值。
SQLSetEnvAttr	是	ISO 92	设置环境属性。
SQLGetEnvAttr	是	ISO 92	返回环境属性的值。
SQLSetStmtAttr	是	ISO 92	设置一个语句属性。

函数名	GBase	一致性	目　的
SQLGetStmtAttr	是	ISO 92	返回语句属性的值。
SQLSetStmtOption	是	不推荐	设置一个语句选项。
SQLGetStmtOption	是	不推荐	返回一个语句选项的值。

预处理 SQL 请求，如下表所示：

函数名	GBase ODBC	一致性	目　的
SQLAllocStmt	是	不推荐	分配一个语句句柄。
SQLPrepare	是	ISO 92	为稍后的执行预处理一个 SQL 语句。
SQLBindParameter	是	ODBC	为一个 SQL 语句中的参数分配存储空间。
SQLGetCursorName	是	ISO 92	返回连接到语句句柄的游标名。
SQLSetCursorName	是	ISO 92	指定一个游标名。
SQLSetScrollOptions	是	ODBC	设置控制游标行为的选项。

提交请求，如下表格所示：

函数名	GBase ODBC	一致性	目　的
SQLExecute	是	ISO 92	执行一个预处理语句。
SQLExecDirect	是	ISO 92	执行一个语句。
SQLNativeSql	是	ODBC	返回一个由驱动翻译的 SQL 语句文本。
SQLDescribeParam	是	ODBC	返回一个语句中指定参数的描述。
SQLNumParams	是	ISO 92	返回一个语句中的参数个数。
SQLParamData	是	ISO 92	和 SQLPutData 一起使用来在执行时用来提供参数数据（用于长数据值）。
SQLPutData	是	ISO 92	发送一个数值的部分或者全部到参数中（用于长数据值）。

接收结果和关于结果的信息，如下表格所示：

函数名	GBase ODBC	一致性	目　的
SQLRowCount	是	ISO 92	返回受插入、更新、或者删除请求影响的行数。
SQLNumResultCols	是	ISO 92	返回结果集中的列数。
SQLDescribeCol	是	ISO 92	描述结果集中的一个列。
SQLColAttribute	是	ISO 92	描述结果集中一个列的属性。
SQLColAttributes	是	不推荐	描述结果集中一个列的属性。
SQLFetch	是	ISO 92	返回多行结果。
SQLFetchScroll	是	ISO 92	返回可卷动的结果行。
SQLExtendedFetch	是	不推荐	返回可卷动的结果行。
SQLSetPos	是	ODBC	在取得的块中定位一个游标，并允许一个应用程序刷新结果集中的数据，或者更新、删除结果集中的数据。
SQLBulkOperations	是	ODBC	执行批量的插入和批量的书签操作，包含通过书签进行更新、删除、和获取数据。

返回错误或者诊断信息，如下表格所示：

函数名	GBase ODBC	一致性	目　的
SQLError	是	不推荐	返回额外错误或者状态信息。
SQLGetDiagField	是	ISO 92	返回额外的诊断信息(一个单域的诊断信息数据结构)。
SQLGetDiagRec	是	ISO 92	返回额外的诊断信息(一个多域的诊断信息数据结构)。

获得关于数据源系统表（分类函数）项目的信息：

函数名	GBase ODBC	一致性	目　的
SQLTransact	是	不推荐	提交或者回滚一个事务。
SQLEndTran	是	ISO 92	提交或者回滚一个事务。

执行事务，如下表所示：

函数名	GBase ODBC	一致性	目　的
SQLTransact	是	不推荐	提交或者回滚一个事务。
SQLEndTran	是	ISO 92	提交或者回滚一个事务。

终止一个语句，如下表格所示：

函数名	GBase ODBC	一致性	目　的
SQLFreeStmt	是	ISO 92	结束语句进程，丢弃未确认的结果，另外可选的释放所有和语句句柄相关的资源。
SQLCloseCursor	是	ISO 92	关闭一个在处理语句时打开的游标。
SQLCancel	是	ISO 92	取消一个 SQL 语句。

终止一个连接，如下表格所示：

函数名	GBase ODBC	一致性	目　的
SQLDisconnect	是	ISO 92	关闭连接。
SQLFreeHandle	是	ISO 92	释放一个环境、连接、语句或者描述符句柄。
SQLFreeConnect	是	不推荐	释放连接句柄。
SQLFreeEnv	是	不推荐	释放一个环境句柄。

7.2 GBase 8a ODBC 数据类型

下表指明了驱动如何映射本地服务器数据类型到默认的 SQL 和 C 数据类型：

本地值	SQL 类型	C 类型
bit	SQL_BIT	SQL_C_BIT
tinyint	SQL_TINYINT	SQL_C_STINYINT
tinyint unsigned	SQL_TINYINT	SQL_C_UTINYINT
bigint	SQL_BIGINT	SQL_C_SBIGINT
bigint unsigned	SQL_BIGINT	SQL_C_UBIGINT
long varbinary	SQL_LONGVARBINARY	SQL_C_BINARY
blob	SQL_LONGVARBINARY	SQL_C_BINARY
longblob	SQL_LONGVARBINARY	SQL_C_BINARY
tinyblob	SQL_LONGVARBINARY	SQL_C_BINARY
mediumblob	SQL_LONGVARBINARY	SQL_C_BINARY
long varchar	SQL_LONGVARCHAR	SQL_C_CHAR
text	SQL_LONGVARCHAR	SQL_C_CHAR
mediumtext	SQL_LONGVARCHAR	SQL_C_CHAR
char	SQL_CHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_CHAR
decimal	SQL_DECIMAL	SQL_C_CHAR
integer	SQL_INTEGER	SQL_C_SLONG
integer unsigned	SQL_INTEGER	SQL_C_ULONG
int	SQL_INTEGER	SQL_C_SLONG
int unsigned	SQL_INTEGER	SQL_C_ULONG
mediumint	SQL_INTEGER	SQL_C_SLONG
mediumint unsigned	SQL_INTEGER	SQL_C_ULONG
smallint	SQL_SMALLINT	SQL_C_SSHORT
smallint unsigned	SQL_SMALLINT	SQL_C_USHORT

本地值	SQL 类型	C 类型
real	SQL_FLOAT	SQL_C_DOUBLE
double	SQL_FLOAT	SQL_C_DOUBLE
float	SQL_REAL	SQL_C_FLOAT
double precision	SQL_DOUBLE	SQL_C_DOUBLE
date	SQL_DATE	SQL_C_DATE
time	SQL_TIME	SQL_C_TIME
year	SQL_SMALLINT	SQL_C_SHORT
datetime	SQL_TIMESTAMP	SQL_C_TIMESTAMP
timestamp	SQL_TIMESTAMP	SQL_C_TIMESTAMP
longtext	SQL_LONGVARCHAR	SQL_C_CHAR
tinytext	SQL_LONGVARCHAR	SQL_C_CHAR
varchar	SQL_VARCHAR	SQL_C_CHAR
enum	SQL_VARCHAR	SQL_C_CHAR
set	SQL_VARCHAR	SQL_C_CHAR
bit	SQL_CHAR	SQL_C_CHAR
bool	SQL_CHAR	SQL_C_CHAR

7.3 GBase 8a ODBC 错误代码

下表列出了驱动返回的来自服务器的错误代码。

本地代码	SQLSTATE 2	SQLSTATE 3	错误信息
500	01000	01000	一般警告
501	01004	01004	字符串数据右截断
502	01S02	01S02	选项改变
503	01S03	01S03	没有行更新/删除
504	01S04	01S04	超过一行更新/删除
505	01S06	01S06	尝试在结果集返回第一行之前取得数据
506	07001	07002	SQLBindParameter 不用于所有的参数

本地代码	SQLSTATE 2	SQLSTATE 3	错误信息
507	07005	07005	预处理语句没有使用游标
508	07009	07009	无效的描述符索引
509	08002	08002	连接名正在使用中
510	08003	08003	连接不存在
511	24000	24000	无效游标状态
512	25000	25000	无效事务状态
513	25S01	25S01	未知事务状态
514	34000	34000	无效的游标名
515	S1000	HY000	一般驱动定义错误
516	S1001	HY001	内存分配错误
517	S1002	HY002	无效的列号
518	S1003	HY003	无效的应用程序缓冲类型
519	S1004	HY004	无效的 SQL 数据类型
520	S1009	HY009	无效的 null 指针
521	S1010	HY010	函数顺序错误
522	S1011	HY011	现在不能设置属性
523	S1012	HY012	无效的事务操作代码
524	S1013	HY013	内存管理错误
525	S1015	HY015	无可用的游标名
526	S1024	HY024	无效的属性值
527	S1090	HY090	无效字符串或者缓冲长度
528	S1091	HY091	无效描述符域的标识符
529	S1092	HY092	无效属性/选项标识符
530	S1093	HY093	无效参数个数
531	S1095	HY095	函数类型超出范围
532	S1106	HY106	获取的类型超出范围
533	S1117	HY117	行值超出范围
534	S1109	HY109	无效的游标位置
535	S1C00	HYC00	没有执行优化特征
0	21S01	21S01	列数和值数不匹配

本地代码	SQLSTATE 2	SQLSTATE 3	错误信息
0	23000	23000	整数约束冲突
0	42000	42000	语法错误或者访问冲突
0	42S02	42S02	没有找到基本表或者视图
0	42S12	42S12	没有找到索引
0	42S21	42S21	列已经存在
0	42S22	42S22	没有找到列
0	08S01	08S01	通信连接失败

7.4 GBase 8a ODBC 应用示例

以下示例中 DRIVER=GBase 8a ODBC 8.3 Driver 为已经创建好的 ODBC 源，配置方法可参考“3.2 创建 GBase 8a ODBC 数据源名”。如仅单次测试使用也可直接在连接串中给 DRIVER 赋值 odbc 驱动文件，如 DRIVER=/usr/lib64/libgsodbc8.so。

7.4.1 使用 ODBC 访问 GBase 数据库

```
#ifdef WIN32
#  include <windows.h>
#else
#  include <unistd.h>
#endif
#  include <stdio.h>
#include <sql.h>
#include <sqlext.h>

#define FAIL 1
#define PRINT_TABLE 1

/*conn string*/
char* connStr = "DRIVER=GBase 8a ODBC 8.3
```

```
Driver;UID=gbase;PWD=gbase20110531;PORT=5258;"
                        "SERVER={192.168.9.173};"
                        "DATABASE=test;";

char* table_name = "t_tttt";

/*sql*/
char* sql_ddl               = "create table t_tttt(a int, b
varchar(10))";
char* sql_insert            = "insert into t_tttt values (1,
\'aaaa\')";
char* sql_select            = "select * from t_tttt";
char* sql_delete            = "delete from t_tttt where a = 1";
char* sql_update            = "update t_tttt set b=\'update\' where
a =1";
char* sql_create_proce  = "create procedure demo_p(in a
varchar(100), out b varchar(100)) "
                                "begin "
                                "set b = CONCAT('InOutParam
',a,'  works!'); "
                                "end;";
char* sql_drop_proce        = "drop procedure demo_p";
char* sql_call_proce        = "CALL demo_p(?, @out)";
char* sql_select_out        = "select @out";


SQLHENV henv;
SQLHDBC hdbc;
const char* usage = "%s s[D(DDL), d(delete), u(update), i(insert),
p(proce), P(drop proce), c(call proce)]\n";

int print_select(SQLHSTMT hstmt, char* sql);
```

```
void print_table(SQLHSTMT hstmt, char* table_name);
void print_diag(SQLRETURN rc, SQLSMALLINT htype, SQLHANDLE handle,
                const char *text, const char *file, int line);

#define ok_env(environ, call) \
do { \
  SQLRETURN rc= (call); \
  print_diag(rc, SQL_HANDLE_ENV, (environ), #call, __FILE__,
__LINE__); \
  if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO) \
    return FAIL; \
} while (0)

#define ok_con(con, call) \
do { \
  SQLRETURN rc= (call); \
  print_diag(rc, SQL_HANDLE_DBC, (con), #call, __FILE__, __LINE__);
\
  if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO) \
    return FAIL; \
} while (0)

#define ok_sql(statement, query) \
do { \
  SQLRETURN rc= SQLExecDirect((statement), (SQLCHAR *)(query),
SQL_NTS); \
  print_diag(rc, SQL_HANDLE_STMT, (statement), \
             "SQLExecDirect(" #statement ", \"" query "\",
SQL_NTS)", \
             __FILE__, __LINE__); \
  if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO) \
    return FAIL; \
```

```
} while (0)

#define ok_stmt(statement, call) \
do { \
  SQLRETURN rc= (call); \
  print_diag(rc, SQL_HANDLE_STMT, (statement), #call, __FILE__,
__LINE__); \
  if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO) \
    return FAIL; \
} while (0)


int demo_select()
{
    SQLHSTMT hstmt;
    printf("exec sql:\n\t%s\n", sql_select);
    ok_con(hdbc, SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt));
    printf("select result:\n");
    print_select(hstmt, sql_select);
    return 0;
}

int demo_delete()
{
    SQLHSTMT hstmt;
    printf("exec sql:\n\t%s\n", sql_delete);
    ok_con(hdbc, SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt));
    ok_stmt(hstmt, SQLExecDirect(hstmt, sql_delete, SQL_NTS));
    print_table(hstmt, table_name);
    return 0;
}

int demo_update()
{
```

```
        SQLHSTMT hstmt;
        printf("exec sql:\n\t%s\n", sql_update);
        ok_con(hdbc, SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt));
        ok_stmt(hstmt, SQLExecDirect(hstmt, sql_update, SQL_NTS));
        print_table(hstmt, table_name);
        return 0;
    }

    int demo_insert()
    {
        SQLHSTMT hstmt;
        printf("exec sql:\n\t%s\n", sql_insert);
        ok_con(hdbc, SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt));
        ok_stmt(hstmt, SQLExecDirect(hstmt, sql_insert, SQL_NTS));
        print_table(hstmt, table_name);
        return 0;
    }

    int demo_create_proce()
    {
        SQLHSTMT hstmt;
        printf("create proc:\n\t%s\n", sql_create_proce);
        ok_con(hdbc, SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt));
        ok_stmt(hstmt, SQLExecDirect(hstmt, sql_create_proce,
SQL_NTS));
        printf("create proc success.\n");
        return 0;
    }

    int demo_drop_proce()
    {
        SQLHSTMT hstmt;
        printf("drop proc:\n\t%s\n", sql_drop_proce);
        ok_con(hdbc, SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt));
        ok_stmt(hstmt, SQLExecDirect(hstmt, sql_drop_proce, SQL_NTS));
```

```
        printf("drop proc success\n");
        return 0;
    }

    int demo_call_proce()
    {
        SQLHSTMT hstmt;
        char* str1 = "input";
        SQLINTEGER str1_cb = SQL_NTS;
        char str2[100] = {'\0'};
        SQLINTEGER str2_cb;
        printf("exec proc:\n\t%s\n", sql_call_proce);
        ok_con(hdbc, SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt));

        ok_stmt(hstmt, SQLPrepare(hstmt, sql_call_proce, SQL_NTS));
        ok_stmt(hstmt, SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_VARCHAR, 100, 0, str1, 6, NULL));
        ok_stmt(hstmt, SQLExecute(hstmt));
        printf("proc result:\n");
        print_select(hstmt, sql_select_out);
        return 0;
    }

    int demo_ddl()
    {
        SQLHSTMT hstmt;
        printf("exec:\n\t%s\n", sql_ddl);
        ok_con(hdbc, SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt));
        ok_stmt(hstmt, SQLExecDirect(hstmt, sql_ddl, SQL_NTS));
        printf("exec success.\n");
        return 0;
    }

    int main(int argc, char** argv)
    {
```

```
        SQLRETURN rc = 0;
        if(argc < 2)
        {
            printf(usage, argv[0]);
            exit(0);
        }
        ok_env(henv,
SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv));
        ok_env(henv,
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(void*)SQL_OV_ODBC3,0));
        ok_env(henv, SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc));
        ok_con(hdbc, SQLDriverConnect(hdbc, NULL, connStr,
SQL_NTS,NULL, 0, NULL, 0));
        switch(argv[1][0])
        {
            case 'D':
                demo_ddl();
                break;
            case 's':
                demo_select();
                break;
            case 'd':
                demo_delete();
                break;
            case 'u':
                demo_update();
                    break;
            case 'i':
                demo_insert();
                break;
            case 'p':
                demo_create_proce();
                break;
            case 'P':
                demo_drop_proce();
                break;
            case 'c':
```

```
                demo_call_proce();
                break;
            default:
                printf(usage, argv[0]);
        }
        ok_con(hdbc, SQLDisconnect(hdbc));
        ok_con(henv, SQLFreeHandle(SQL_HANDLE_DBC, hdbc));
        ok_env(henv, SQLFreeEnv(henv));
        return 0;
    }

    int print_select(SQLHSTMT hstmt, char* sql)
    {
        int num_cols = 0;
        int i = 0;
        int buff_len = 1024;
        int val_len = 0;
        SQLSMALLINT col_type, tnum = 0;
        unsigned char* buff = (unsigned char*)malloc(buff_len);
        SQLExecDirect(hstmt, sql, SQL_NTS);
        ok_stmt(hstmt, SQLNumResultCols(hstmt, &tnum));
        num_cols = tnum;
        for(i=1; i<=num_cols; i++)
        {
            ok_stmt(hstmt, SQLDescribeCol(hstmt, i, buff, buff_len,
NULL, NULL, NULL, NULL, NULL));
            printf("|%s\t", buff);
        }
        printf("\n");
        for(i=1; i<=num_cols; i++)
        {
            printf("--------");
        }
        printf("\n");
        while(SQL_SUCCESS == SQLFetch(hstmt))
        {
```

```
            for(i=1; i<=num_cols; i++)
            {
                col_type = 0;
                memset(buff, 0, buff_len);
                val_len = 0;
                ok_stmt(hstmt, SQLGetData(hstmt, i, SQL_C_CHAR, buff,
buff_len, &val_len));
                printf("|%s\t", buff);
            }
            printf("\n");
        }
        free(buff);
        return 0;
    }

    void print_table(SQLHSTMT hstmt, char* table_name)
    {
        char sql_select[500] = "select * from ";
        int i=0;

        if(0 == PRINT_TABLE)
        {
            return;
        }
        printf("print table %s.\n", table_name);

        while(table_name[i]){sql_select[14+i] = table_name[i]; i++;}
        sql_select[14+i] = '\0';
        print_select(hstmt, sql_select);
    }

    void print_diag(SQLRETURN rc, SQLSMALLINT htype, SQLHANDLE handle,
                    const char *text, const char *file, int line)
    {
        if (rc != SQL_SUCCESS)
        {
```

```
            SQLCHAR     sqlstate[6], message[SQL_MAX_MESSAGE_LENGTH];
            SQLINTEGER  native_error;
            SQLSMALLINT length;
            SQLRETURN   drc;

            /** @todo map rc to SQL_SUCCESS_WITH_INFO, etc */
            printf("# %s = %d\n", text, rc);

            /** @todo Handle multiple diagnostic records. */
            drc= SQLGetDiagRec(htype, handle, 1, sqlstate,
&native_error,message, SQL_MAX_MESSAGE_LENGTH - 1, &length);

            if (SQL_SUCCEEDED(drc))
                printf("# [%6s] %*s in %s on line %d\n",
sqlstate, length, message, file, line);
            else
                printf("# Did not get expected diagnostics from
SQLGetDiagRec() = %d in file %s on line %d\n", drc, file, line);
        }
    }
```

7.4.2 高可用性示例

```
#ifdef WIN32
#include<windows.h>
#else
#include<unistd.h>
#endif
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sql.h>
#include<sqlext.h>
```

```
#define isSuc(result) ((result) == SQL_SUCCESS || (result) ==
SQL_SUCCESS_WITH_INFO)

char hosts[][100] = {"192.168.5.64",
"192.168.5.65",
"192.168.5.64;192.168.5.65",
"192.168.5.65;192.168.5.64",
"192.168.5.64;192.168.5.64",
"192.168.5.64;192.168.5.64;192.168.5.65",
"192.168.5.64;192.168.5.65;192.168.5.64",
"192.168.5.65;192.168.5.64;192.168.5.64",
"192.168.5.65;192.168.5.66",
NULL};
int main(void)
{
    char conn_format[200] = "DRIVER={GBase 8a ODBC 8.3 Driver};"
    "SERVER={%s};UID=root;PWD=1;DATABASE=test;PORT=5258;";
        char conn[200] = {'\0'};
        int i = 0;
        short sret; //返回代码
        void* henv;  //环境句柄
        void* hdbc;  //连接句柄
        /** 申请环境句柄**/
        sret =
SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
        if(!isSuc(sret))printf("申请环境句柄出错\n");
        /** 设置环境属性,声明 ODBC 版本**/
        sret =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(void*)SQL_OV_ODBC3,0);
    if(!isSuc(sret))printf("声明 ODBC 版本出错\n");
    /** 申请连接句柄**/
    sret = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
    if(!isSuc(sret))printf("申请连接句柄出错\n");
    /** 设置连接属性**/
    sret = SQLSetConnectAttr(hdbc,SQL_ATTR_AUTOCOMMIT,
```

```
        (void*)SQL_AUTOCOMMIT_OFF,SQL_IS_INTEGER);
    if(!isSuc(sret))printf("设置连接属性出错\n");

    while(*hosts[i])
    {
        sprintf(conn, conn_format, hosts[i]);
        /** 连接数据源**/
        sret = SQLDriverConnect(hdbc,NULL,(unsigned char *)conn,
                                SQL_NTS, NULL, 0, NULL, 0);
        if(!isSuc(sret))printf("连接%s 出错\n", hosts[i]);
        else printf("连接%s 成功\n", hosts[i]);
        SQLDisconnect(hdbc);
        memset(conn, '\0',  strlen(conn));
        i++;
    }
    return 0;
}
```

7.4.3 负载均衡示例

```
#ifdef WIN32
#  include <windows.h>
#else
#  include <unistd.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>

#define isSuc(result) ((result) == SQL_SUCCESS || (result) ==
SQL_SUCCESS_WITH_INFO)
char* conn_str = "DRIVER=GBase 8a ODBC 8.3
```

```
    Driver;UID=gbase;PWD=gbase20110531;"

        "SERVER={192.168.7.172;192.168.7.173;192.168.7.174};"
                        "CONNECTION_BALANCE=1;GCLUSTER_ID=gcluster;"
                        "CHECK_INTERVAL=90;";
    int main(void)
    {
        short sret;
        void* henv;
        void* hdbc;
        char message[512];
        char hostinfo[100];
        int i = 0;

        sret = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
        if(!isSuc(sret))
        {
            SQLGetDiagRec(SQL_HANDLE_ENV, henv, 1, NULL, NULL,
                                (SQLCHAR*)message, 1023, NULL);
            printf("%s\n", message);
            return -1;
        }
        sret =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(void*)SQL_OV_ODBC3,0);
        if(!isSuc(sret))
        {
            SQLGetDiagRec(SQL_HANDLE_ENV, henv, 1, NULL, NULL,
                                (SQLCHAR*)message, 1023, NULL);
            printf("%s\n", message);
            return -1;
        }
        sret = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
        if(!isSuc(sret))
        {
            SQLGetDiagRec(SQL_HANDLE_ENV, henv, 1, NULL, NULL,
(SQLCHAR*)message, 1023, NULL);
```

```
            printf("%s\n", message);
            return -1;
        }

        for (i=0; i<6; i++)
        {
            sret = SQLDriverConnect(hdbc, NULL,(SQLCHAR*)conn_str,
SQL_NTS,NULL, 0, NULL, 0);
            if(!isSuc(sret))
            {
                SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, 1, NULL, NULL,
                      (SQLCHAR*)message, 1023, NULL);
                printf("%s\n", message);
                return -1;
            }

            SQLGetInfo(hdbc, SQL_SERVER_NAME, hostinfo, 200, NULL);

            printf("使用 GBase 8a ODBC 高可用负载均衡连接 GBase 集群节
点: %s.\n", hostinfo);

            sret = SQLDisconnect(hdbc);
            if(!isSuc(sret))
            {
                SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, 1, NULL, NULL,
                       (SQLCHAR*)message, 1023, NULL);
                printf("%s\n", message);
                return -1;
            }
        }
        sret = SQLFreeConnect(hdbc);
        if(!isSuc(sret))
        {
            SQLGetDiagRec(SQL_HANDLE_DBC, hdbc, 1, NULL, NULL,
                                 (SQLCHAR*)message, 1023, NULL);
            printf("%s\n", message);
            return -1;
```

```
        }
        sret = SQLFreeEnv(henv);
        if(!isSuc(sret))
        {
            SQLGetDiagRec(SQL_HANDLE_ENV, henv, 1, NULL, NULL,
                          (SQLCHAR*)message, 1023, NULL);
            printf("%s\n", message);
            return -1;
        }
        return 0;
    }
```

7.4.4 连接池示例

```
#ifdef WIN32
#  include <windows.h>
#  define sleep(x) Sleep(x*1000)
#else
#  include <unistd.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <time.h>
#include "sqlgbase.h"

#define isSuc(result) ((result) == SQL_SUCCESS || (result) ==
SQL_SUCCESS_WITH_INFO)
#define FOR_TIMES  50
#define CON_NUM  50
```

```
    SQLHANDLE henv;
    int seg;
    #ifdef WIN32
    CRITICAL_SECTION thread_quit_lock;
    #define THREAD_ID() GetCurrentThreadId()
    #else
    pthread_mutex_t thread_quit_lock;
    #define THREAD_ID() pthread_self()
    #endif

    void getError(SQLSMALLINT type, SQLHANDLE handle)
    {
        SQLCHAR sqlstate[1024];
        SQLCHAR errormes[1024];
        SQLINTEGER numerror;
        SQLSMALLINT errlen;
        SQLGetDiagRec(type, handle, 1, sqlstate, &numerror, errormes,
1024, &errlen);
        printf("sqlstate:%s\terrmsg:%s\terrno:%d\n",sqlstate,
errormes, numerror);
    }

    char* connStr = "DRIVER=GBase 8a ODBC 8.3
    Driver;UID=gbase;PWD=gbase20110531;"

        "SERVER={192.168.9.173;192.168.9.174;192.168.9.175};DATABASE
    =test; PORT=5258;"

        "GCLUSTER_ID=aaa;POOL_MAX_ACTIVE_SIZE=70;POOL_MAX_IDLE=65;"
                    "POOL_INIT_SIZE=10;"
                    "POOL_TEST_BORROW=0;"
                    "POOL_LBS=1;"
                    "POOL_CHECKOUT_TIMEOUT=5000;"
```

```
                    "POOL_MANAGER=1;"
                    "POOL_TEST_INVALID_CONN_PERIOD=2000;"
                    "POOL_KEEP_IDLE=1; POOL_MIN_IDLE=12;"
                    "POOL_CLEAR_OVERDUE=1; POOL_USED_LIFE=400000;
    POOL_IDLE_LIFE=0;";

    #ifdef WIN32
    DWORD WINAPI thread_bf(LPVOID arg)
    #else
    int thread_bf(void* arg)
    #endif
    {
        SQLHDBC hdbc;
        SQLHANDLE hsmt;
        int sret = 0;
        int i =0 ;
        clock_t t1, t2;
        time_t t;
        int rc = 0;

        while(i < FOR_TIMES)
        {
            t1 = t2 = clock();
            SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
            sret = SQLDriverConnect(hdbc, NULL, connStr, SQL_NTS,NULL,
0, NULL, 0);
            if (!isSuc(sret))
            {
                getError(SQL_HANDLE_DBC, hdbc);
            }
            else
            {
                t2 = clock();
                time(&t);
                printf("thread id %u\t\tconnect success! use %lu
```

```
msec\t%s.\n",THREAD_ID(), (t2 - t1), ctime(&t));
                rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc,  &hsmt);
                if (rc == SQL_SUCCESS)
                {
                    SQLExecDirect(hsmt, "show databases", strlen("show
databases"));
                    SQLFreeHandle(SQL_HANDLE_STMT, hsmt);
                }
                SQLDisconnect(hdbc);
                SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
            }
            i++;
        }

        #ifdef WIN32
            EnterCriticalSection(&thread_quit_lock);
            seg--;
            LeaveCriticalSection(&thread_quit_lock);
        #else
            pthread_mutex_lock(&thread_quit_lock);
            seg--;
            pthread_mutex_unlock(&thread_quit_lock);
        #endif
        return 0;
    }

    void t1_bf()
    {
        SQLHDBC hdbc;
        int rc = 0, i = 0;

        #ifdef WIN32
            InitializeCriticalSection(&thread_quit_lock);
        #else
            pthread_mutex_init(&thread_quit_lock, NULL);
        #endif
```

```
            SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
            SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(void*)SQL_OV_O
DBC3,0);

        seg = CON_NUM;
        while(i<CON_NUM)
        {
            #ifdef WIN32
                CreateThread(NULL, 0, thread_bf, NULL, 0, NULL);
            #else
                pthread_t thread;
            pthread_create(&thread, 0, thread_bf, NULL);
    #endif
            i++;
        }

        while(seg > 0)sleep(1);
        SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
        rc = SQLDriverConnect(hdbc, NULL, connStr, SQL_NTS,NULL, 0, NULL,
0);
        if (!isSuc(rc))
        {
            getError(SQL_HANDLE_DBC, hdbc);
            return;
        }
        SQLSetConnectAttr(hdbc, SQL_ATTR_GBASE_POOL_FREE,
SQL_FREE_GBASE_POOL, NULL);
        SQLDisconnect(hdbc);

        SQLFreeHandle(SQL_HANDLE_ENV, henv);
    }
```

7.4.5 cache_insert_values 示例

odbc_err.h
#include <iostream>

odbc_err.h

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

class RV
{
public:
    RV()
    {
        ErrNo = 0;
        ErrMsg = "";
    }
    RV(int eno, std::string emsg)
    {
        ErrNo = eno;
        ErrMsg = emsg;
    }
    bool IsSuccess()
    {
        if(0 == ErrNo)
        {
            return true;
        }
        return false;
    }
public:
    int   ErrNo;
    std::string ErrMsg;
};

RV GetDbcError(SQLHDBC handle)
{
    RV rv;
    std::string res;
    SQLCHAR     sqlstate[6], message[SQL_MAX_MESSAGE_LENGTH];
    SQLINTEGER  native_error;
    SQLSMALLINT length;
    SQLRETURN   drc;
    drc   =   SQLGetDiagRec(SQL_HANDLE_DBC,   handle,   1,   sqlstate,
&native_error, message, SQL_MAX_MESSAGE_LENGTH - 1, &length);
    if((drc != SQL_SUCCESS && drc != SQL_SUCCESS_WITH_INFO))
```

odbc_err.h

```
    {
        rv.ErrNo = -1;
        rv.ErrMsg = "Fail to get ODBC's error message.";
    }
    else
    {
        rv.ErrNo = 0 == native_error ? -1 : native_error;
        rv.ErrMsg = (char*)message;
    }
    return rv;
}

RV GetStmtError(SQLHSTMT handle)
{
    RV rv;
    std::string res;
    SQLWCHAR     sqlstate[6], message[SQL_MAX_MESSAGE_LENGTH];
    SQLINTEGER  native_error;
    SQLSMALLINT length;
    SQLRETURN   drc;
    drc  =  SQLGetDiagRecW(SQL_HANDLE_STMT,  handle,  1,  sqlstate,
&native_error, message, SQL_MAX_MESSAGE_LENGTH - 1, &length);
    if((drc != SQL_SUCCESS && drc != SQL_SUCCESS_WITH_INFO))
    {
        rv.ErrNo = -1;
        rv.ErrMsg = "Fail to get ODBC's error message.";
    }
    else
    {
        rv.ErrNo = 0 == native_error ? -1 : native_error;
        rv.ErrMsg = (char*)message;
    }
    return rv;
}
```

main.cpp

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

main.cpp

```
<string.h>
#include <vector>
#include <sql.h>
#include <sqlext.h>
#include

"odbc_err.h"

std::string connURL="dsn=test;CACHE_INSERT_VALUES=1";
SQLHSTMT hstmt, hstmt2;
SQLHDBC hdbc, hdbc2;
SQLHENV henv;

#define

PARAMSET_SIZE 200

SQLRETURN InitStmtHandle()
{
    SQLRETURN rc;
    RV

rv;
    std::string errMsg;

    rc = SQLAllocHandle

(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
    if((rc != SQL_SUCCESS && rc

!= SQL_SUCCESS_WITH_INFO))
    {
        rv.ErrNo = -1;
        rv.ErrMsg =

"Fail to malloc SQL_HANDLE_ENV.";
        std::cout << rv.ErrMsg <<

std::endl;
        return rc;
    }
```

main.cpp

```
    else
    {
        rc =

SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);


  if((rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO))
        {


    rv.ErrNo = -1;
            rv.ErrMsg = "Fail o set

SQL_ATTR_ODBC_VERSION to SQL_OV_ODBC3.";
            std::cout <<

rv.ErrMsg << std::endl;
            return rc;
        }
    }

    rc =

SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
    if((rc !=

SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO))
    {
        rv.ErrNo = -1;


      rv.ErrMsg = "Fail to malloc SQL_HANDLE_DBC.";
        std::cout <<

rv.ErrMsg << std::endl;
        return rc;
    }
    else
    {
```

main.cpp

```
rc = SQLSetConnectAttr(hdbc,SQL_AUTOCOMMIT,(SQLPOINTER)

SQL_AUTOCOMMIT_OFF,SQL_IS_INTEGER);
        if((rc != SQL_SUCCESS &&

rc != SQL_SUCCESS_WITH_INFO))
        {
            rv = GetDbcError

(hdbc);
            std::cout << rv.ErrMsg << std::endl;


return rc;
        }
        rc = SQLDriverConnect(hdbc, NULL, (SQLCHAR*)

connURL.c_str(), SQL_NTS, NULL, 0, NULL, 0);
        if((rc !=

SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO))
        {
            rv =

GetDbcError(hdbc);
            std::cout << rv.ErrMsg << std::endl;


     return rc;
        }
    }
    rc = SQLAllocHandle(SQL_HANDLE_DBC,

henv, &hdbc2);
    if((rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO))


 {
        rv.ErrNo = -1;
        rv.ErrMsg = "Fail to malloc
```

main.cpp

```
SQL_HANDLE_DBC.";
        std::cout << rv.ErrMsg << std::endl;


return rc;
    }
    else
    {
        rc = SQLSetConnectAttr

(hdbc2,SQL_AUTOCOMMIT,(SQLPOINTER)SQL_AUTOCOMMIT_OFF,SQL_IS_INTEGER)
;


if((rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO))
        {


  rv = GetDbcError(hdbc2);
            std::cout << rv.ErrMsg <<

std::endl;
            return rc;
        }
        rc = SQLDriverConnect

(hdbc2, NULL, (SQLCHAR*)connURL.c_str(), SQL_NTS, NULL, 0, NULL, 0);


   if((rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO))
        {


     rv = GetDbcError(hdbc2);
            std::cout << rv.ErrMsg <<

std::endl;
            return rc;
        }
    }
```

main.cpp

```
    SQLAllocHandle

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

hdbc2, &hstmt2);

    return rc;
}

void FreeStmtHandle()
{


SQLFreeStmt(hstmt, SQL_CLOSE);
    SQLFreeStmt(hstmt2, SQL_CLOSE);


SQLDisconnect(hdbc);
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);


SQLFreeHandle(SQL_HANDLE_STMT, hstmt2);
    SQLFreeHandle(SQL_HANDLE_DBC,

hdbc);
}
SQLRETURN TestCacheInsertValues()
{
    SQLRETURN rc;
    RV rv;


SQLSMALLINT colCount = 0;
    SQLUINTEGER*  NumRowsFetched = new

SQLUINTEGER;

    rc = SQLExecDirect(hstmt, (SQLCHAR*)"select * from t",

SQL_NTS);
```

main.cpp

```
    if(!SQL_SUCCEEDED(rc))
    {
        rv =

GetStmtError(hstmt);
        std::cout << rv.ErrMsg << std::endl;


return rc;
    }
    rc = SQLNumResultCols(hstmt, &colCount);


if(!SQL_SUCCEEDED(rc))
    {
        rv = GetStmtError(hstmt);


std::cout << rv.ErrMsg << std::endl;
        return rc;
    }
    if

(colCount <= 0)
    {
        std::cout << "column num is 0" <<

std::endl;
        return rc;
    }

    std::vector<SQLPOINTER>

vecColBuf(colCount, 0);
    std::vector<SQLINTEGER*> vecColBufLen(colCount,

0);
    std::vector<SQLLEN*> vecColInd(colCount, 0);
    SQLUSMALLINT

RowStatusArray[PARAMSET_SIZE];
```

main.cpp

```
    std::string insertSql = "insert into t1

values(";

    SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_BIND_TYPE,

SQL_BIND_BY_COLUMN, 0);
    SQLSetStmtAttr(hstmt,

SQL_ATTR_ROW_ARRAY_SIZE, (SQLPOINTER)PARAMSET_SIZE, 0);


SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_STATUS_PTR, RowStatusArray, 0);


SQLSetStmtAttr(hstmt, SQL_ATTR_ROWS_FETCHED_PTR, &NumRowsFetched, 0);


SQLSetStmtAttr(hstmt2, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)

PARAMSET_SIZE, 0);
    for (SQLSMALLINT idx = 0; idx < colCount; +

+idx)
    {
        SQLULEN colSize = 0;
        SQLSMALLINT

colType = 0;
        rc = SQLDescribeCol(hstmt, idx + 1, NULL, 0,

NULL, &colType, &colSize, NULL, NULL);
        if(!SQL_SUCCEEDED(rc))


{
            rv = GetStmtError(hstmt);


std::cout << rv.ErrMsg << std::endl;
            break;
```

main.cpp

```
}
        vecColBufLen[idx] = new SQLINTEGER[PARAMSET_SIZE];


vecColInd[idx] = new SQLLEN[PARAMSET_SIZE];
        vecColBuf[idx] =

new SQLCHAR[PARAMSET_SIZE * (colSize + 1) * sizeof(SQLCHAR)];


for(int i = 0; i < PARAMSET_SIZE; ++i)
        {


vecColBufLen[idx][i] = (colSize + 1) * sizeof(SQLCHAR);
        }


memset(vecColInd[idx], 0, (PARAMSET_SIZE * sizeof(SQLLEN)));


memset(vecColBuf[idx], 0, (PARAMSET_SIZE * (colSize + 1)));

        rc = SQLBindCol(hstmt, idx + 1, SQL_C_CHAR, vecColBuf

[idx], *vecColBufLen[idx], vecColInd[idx]);
        if(!

SQL_SUCCEEDED(rc))
        {
            rv =

GetStmtError(hstmt);
            std::cout << rv.ErrMsg <<

std::endl;
            break;
        }
```

main.cpp

```
        rc =

SQLBindParameter(hstmt2, idx + 1, SQL_PARAM_INPUT, SQL_C_CHAR, -8, 0,
0,

vecColBuf[idx], (colSize + 1) * sizeof(SQLCHAR), vecColInd[idx]);


if(!SQL_SUCCEEDED(rc))
        {
            rv =

GetStmtError(hstmt2);
            std::cout << rv.ErrMsg <<

std::endl;
            break;
        }


insertSql.append("?,");
    }
    insertSql.erase(--insertSql.end());


insertSql.append(")");

    rc = SQLPrepare(hstmt2, (SQLCHAR*)

insertSql.c_str(), insertSql.length());
    if(!SQL_SUCCEEDED(rc))
    {


rv = GetStmtError(hstmt2);
        std::cout << rv.ErrMsg << std::endl;


     return rc;
    }
    if(SQL_SUCCEEDED(rc))
```

main.cpp

```
    {
        while

(SQL_SUCCEEDED(rc = SQLFetch(hstmt)) || SQL_NO_DATA == rc)


{
            if(SQL_NO_DATA == rc)
            {


        rc = SQLTransact(henv, hdbc2, SQL_COMMIT);


    if(!SQL_SUCCEEDED(rc))
                {


        rv = GetDbcError(hdbc);


std::cout << rv.ErrMsg << std::endl;
                }


        break;
            }


SQLSetStmtAttr(hstmt2, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)

NumRowsFetched, 0);
            if(!SQL_SUCCEEDED(SQLExecute

(hstmt2)))
            {
                rv =

GetStmtError(hstmt2);
                std::cout << rv.ErrMsg
```

main.cpp

```
<< std::endl;
                break;
            }


}
    }

    for(SQLSMALLINT idx = 0; idx < colCount; ++idx)
    {


if(vecColBuf[idx])
        {
            delete []

vecColBuf[idx];
        }
        if(vecColBufLen[idx])


delete [] vecColBufLen[idx];
        if(vecColInd[idx])


delete [] vecColInd[idx];
    }
    delete NumRowsFetched;
    return

rc;
}

int main()
{
    InitStmtHandle();
    TestCacheInsertValues();


FreeStmtHandle();
    return 0;
```

```
main.cpp
}
```

7.5 GBase 8a ODBC 常见问题

7.5.1 支持动态游标

GBase 8a ODBC 8.3 除了支持只向前和静态游标类型还支持动态游标类型。由于性能问题，驱动程序缺省是不支持这个特征。用户可以通过指明连接选项标识 DYNAMIC_CURSOR＝1 或从 DSN 配置中选中使用动态游标选项来使用这个特征。

7.5.2 使用 unixODBC 访问 GBase 数据库时出现段错误

这是 unixODBC 的问题，通过在/etc/odbcinst.ini 文件中增加 DontDLClose=1 关键字可以解决该问题。

7.5.3 多线程使用 unixODBC 访问 GBase 数据库的配置

在多线程调用 unixODBC 访问 GBase 数据库时，可以在/etc/odbcinst.ini 文件中增加 threading=0 来提高并发。

7.5.4 python 语言调用 GBase 8a ODBC 驱动

使用 python 语言调用 GBase 8a ODBC 时,需要在下载安装 python 和 pyodbc 包。以 python2.7 为例

1， 从 http://www.python.org/ftp/python/2.7.3/Python-2.7.3.tgz 下载 python2.7 的源码包。

2， 从 http://pyodbc.googlecode.com/files/pyodbc-3.0.6.zip 下载

pyodbc-3.0.6 的安装包。

3， 安装 python2.7，先解压 Python-2.7.3.tgz，然后执行如下命令：

```
./configure --prefix=/opt/python2.7
make
make install
```

4， 安装 pyodbc-3.0.6，先解压 pyodbc-3.0.6.zip，然后执行如下命令：

```
/opt/python2.7/bin/python setup.py build install
```

这样 python2.7 和 pyodbc-3.0.6 就安装成功。可以进行 python 控制台进行验证，如下所示：

```
[root@GBase pyodbc-3.0.6]# /opt/python2.7/bin/python
Python 2.7.3 (default, Jul 16 2012, 18:45:35)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-50)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyodbc
>>>
```

使用 import pyodbc 引用 pyodbc 时没有报错，说明 python 安装成功。

接下来是创建 ODBC 数据源，创建 ODBC 数据源的方法请参考 3.2.2。如果涉及到中文，那么一定要设置 GBase 8a ODBC 驱动的字符集（该字符集需要与集群安装时默认的字符集保持一致)。此用例中 GBase 8a ODBC 字符集设置为 UTF8。

创建 ODBC 数据源成功后就可以使用如下 python 脚本测试：

```
#! /opt/python2.7/bin/python
#coding:utf8
import pyodbc
```

```
conn_str = 'DSN=gbase;'
conn = pyodbc.connect(conn_str)
cursor = conn.cursor()
cursor.execute("drop table if exists t_pyodbc")
cursor.execute("create table t_pyodbc(a varchar(100), b int)")
cursor.execute("insert into t_pyodbc(a , b) values(?, ?)",'pyodbc',
10)
cursor.execute("insert into t_pyodbc(a , b) values(?, ?)", '南大
通用', 11)
cursor.commit()
cursor.execute("select * from t_pyodbc")
row = cursor.fetchall()
for r in row:
    print "%s %d" % (r[0].encode('utf8'), r[1])
```

7.5.5 perl 语言调用 GBase 8a ODBC 驱动

使用 python 语言调用 GBase 8a ODBC 时,需要在下载安装 perl 和 DBD::ODBC 包。以 perl-5.14 为例:

1, 从 http://downloads.activestate.com/ActivePerl/releases/5.14.2.1402/ActivePerl-5.14.2.1402-x86_64-linux-glibc-2.3.5-295342.tar.gz 下载 perl-5.14 的安装包。

2, 从 http://mirrors.sohu.com/CPAN/authors/id/M/MJ/MJEVANS/DBD-ODBC-1.39.tar.gz 下载 DBD-ODBC-1.39。

3, 安装 perl-5.14。perl 默认安装在/opt/ActivePerl-5.14 目录下。

4, 安装 DBD-ODBC-1.39, 先解压 DBD-ODBC-1.39.tar.gz, 然后执行如下

命令安装：

```
/opt/ActivePerl-5.14/bin/perl Makefile.PL

make

make install
```

5， 接下来是创建 ODBC 数据源，创建 ODBC 数据源的方法请参考 3.2.2。如果涉及到中文，那么一定要设置 GBase 8a ODBC 驱动的字符集（该字符集需要与集群安装时默认的字符集保持一致）。此用例中 GBase 8a ODBC 字符集设置为 UTF8。

```
#!/opt/ActivePerl-5.14/bin/perl
use DBI;
use encoding 'utf-8';

$dbh=DBI->connect('dbi:ODBC:gbase8a', 'gbase', 'gbase20110531');
my $sth=$dbh->prepare("drop table if exists t_plodbc");
$sth->execute();
my $sth=$dbh->prepare("create table t_plodbc(a varchar(100), b
int)");
$sth->execute();
my $sth=$dbh->prepare("insert into t_plodbc values('南大通用',
10)");
$sth->execute();
my $sth=$dbh->prepare("select * from t_plodbc");
$sth->execute();
while(@data=$sth->fetchrow_array())
{
    print "$data[0]\n";
}
```

7.5.6 php 语言调用 GBase 8a ODBC 驱动

```
<?
$conn_str = "DRIVER=GBase 8a ODBC 8.3
Driver;SERVER=192.168.9.173;UID=gbase;PWD=gbase20110531;database
=test;charset=gbk;LOG_QUERY=1";
$sql_drop = "drop table if exists t_php";
$sql_create = "create table t_php (col1 int, col2 varchar(100))";
$sql_insert1 = "insert into t_php values (1, 'abc')";
$sql_insert2 = "insert into t_php values (2, '南大通用')";
$sql_select = "select * from t_php";
$conn = odbc_connect($conn_str, "", "");
if(!($conn)){
    echo "get connection: " . odbc_errormsg($conn);
    echo "<br>";
}
odbc_exec($conn,$sql_drop);
if (odbc_error()) {
    echo "drop table: " . odbc_errormsg($conn);
    echo "<br>";
}
odbc_exec($conn,$sql_create);
if (odbc_error()) {
    echo "create table: " . odbc_errormsg($conn);
    echo "<br>";
}

/*insert data*/
odbc_exec($conn,$sql_insert1);
if (odbc_error()) {
    echo "insert" . odbc_errormsg($conn);
    echo "<br>";
}
```

```
    odbc_exec($conn,$sql_insert2);
    if (odbc_error()) {
        echo "insert" . odbc_errormsg($conn);
        echo "<br>";
    }
    /*select data*/
    $rs = odbc_exec($conn,$sql_select);
    if (odbc_error()) {
        echo "select" . odbc_errormsg($conn);
        echo "<br>";
    }
    while(odbc_fetch_row($rs)){
        echo "\n" . odbc_result($rs, "col1") . "\t" . odbc_result($rs,
"col2") . "\n";
        echo "<br>";
    }
    odbc_exec($conn,$sql_drop);
    if (odbc_error()) {
        echo odbc_errormsg($conn);
    }
    odbc_close($conn);
    ?>
```

7.5.7 GBase ODBC 驱动重复执行能够返回结果的 SQL 问题

如果应用像如下示例调用 ODBC 接口，那么 GBase ODBC 会在执行 SQLNumResultCols 时先在 Server 端执行一次 selectSql，然后在执行 SQLExecute 在 Server 端再执行一次 selectSql。

```
SQLPrepare(hstmt, selectSql, SQL_NTS);
```

```
SQLNumResultCols(hstmt, &tnum);

...

SQLExecute(hstmt);
```

第一次执行 selectSql 是为了获取结果集的 MetaData，第二次执行 selectSql 时才会真正获取数据。

7.5.8 获取存储过程的结果集

使用 SQLPrepare 接口获取存储过程的结果集必须在调用 SQLExecute 后获取结果集信息，否则获取不到结果集信息。

7.5.9 调用 SQLBindCol 时报错 Invalid descriptor index

正常情况下传给 SQLBindCol 的第二个参数 ColumnNumber 为 0 或者大于结果集的列数时会报 Invalid descriptor index 错误。

当传入的 ColumnNumber 参数正确时，由于某种原因 gclusterd 主动 断开了连接，也会导致 SQLBindCol 报错 Invalid descriptor index。比如当结果集比较大时，ODBC 客户端应用不能及时从 ODBC 读取结果集，导致 gclusterd 向 ODBC 发送数据时写超时。这种情况在 gclusterd 的参数 "gcluster_send_client_data_timeout"默认值 30 时比较常见。这时需要修改该参数值为一个较 大值或者直接修改为 0。

7.5.10 特殊场景下屏蔽 ODBC 负载均衡方式

GBase 8a Cluster ODBC_8.4_build2.0 支持屏蔽 ODBC 负载均衡，使用环境

变量 GSODBC_USE_BALANCE 来控制 gsodbc 驱动是否使用 balance 后台检测。 默认使用 ODBC 负载均衡，如需屏蔽 ODBC 负载均衡功能，方法如下：

应用程序中在创建连接前执行：

```
system( “export GSODBC_USE_BALANCE” );
```

之后该程序创建的连接均屏蔽 ODBC 负载均衡功能，屏蔽后会使用 CAPI 的 balance 实现，也就是说会随机选取一个节点执行 SQL。

注：

建议使用中兴的库时屏蔽 ODBC 负载均衡。

GBASE

南大通用数据技术股份有限公司
General Data Technology Co., Ltd.

微博二维码　微信二维码

技术支持热线：400-013-9696