



## GBase 8s ODBC Driver 程序员指南



## GBase 8s ODBC Driver 程序员指南，南大通用数据技术股份有限公司

GBase 版权所有©2004-2021，保留所有权利

### 版权声明

本文档所涉及的软件著作权及其他知识产权已依法进行了相关注册、登记，由南大通用数据技术股份有限公司合法拥有，受《中华人民共和国著作权法》、《计算机软件保护条例》、《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

### 免责声明

本文档包含的南大通用数据技术股份有限公司的版权信息由南大通用数据技术股份有限公司合法拥有，受法律的保护，南大通用数据技术股份有限公司对本文档可能涉及到的非南大通用数据技术股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅，并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经南大通用数据技术股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将视为侵权，南大通用数据技术股份有限公司具有依法追究其责任的权利。

本文档中包含的信息如有更新，恕不另行通知。您对本文档的任何问题，可直接向南大通用数据技术股份有限公司告知或查询。

### 通讯方式

南大通用数据技术股份有限公司

天津市高新区开华道22号普天创新产业园东塔20-23层

电话：400-013-9696

邮箱：[info@gbase.cn](mailto:info@gbase.cn)

### 商标声明

**GBASE<sup>®</sup>** 是南大通用数据技术股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由南大通用数据技术股份有限公司合法拥有，受法律保护。未经南大通用数据技术股份有限公司书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯南大通用数据技术股份有限公司商标权的，南大通用数据技术股份有限公司将依法追究其法律责任。

## 目 录

1 简介 .....	1
1.1 关于本出版物.....	1
1.2 符合行业标准.....	1
1.3 演示数据库.....	1
1.4 示例代码约定.....	2
1.5 语法图.....	2
1.5.1 如何阅读命令行语法图.....	2
1.5.2 关键字和标点.....	3
1.5.3 标识和名称.....	4
2 GBase 8s ODBC Driver 概述.....	4
2.1 什么是 GBase 8s ODBC Driver? .....	4
2.1.1 GBase 8s ODBC Driver 特性.....	5
2.1.2 某些 ODBC 函数参数的附加的值.....	6
2.2 ODBC 组件概述.....	6
2.2.1 带有驱动程序管理器的 GBase 8s ODBC Driver .....	7
2.2.2 不带有驱动程序管理器的 GBase 8s ODBC Driver (UNIX™) .....	7
2.3 带有 DMR 的 GBase 8s ODBC Driver.....	8
2.4 GBase 8s ODBC Driver 组件.....	8
2.4.1 环境变量.....	9
2.4.2 标头文件.....	10
2.4.3 数据类型.....	10
2.4.4 库.....	11
2.5 GBase 8s ODBC Driver API.....	12
2.5.1 环境、连接和语句句柄.....	13
2.5.2 缓冲区.....	14
2.5.3 SQLGetInfo 尝试实现.....	15
2.6 全球语言支持.....	17
2.7 X/Open 标准接口 .....	19
2.8 外部认证.....	20
2.8.1 UNIX 和 Linux 上的 Pluggable Authentication Module (PAM) .....	20
2.8.2 Windows 上的 LDAP Authentication .....	20
2.8.3 带有认证的 SQLSetConnectAttr() 函数.....	20
2.9 绕过 ODBC 解析.....	23

2.10	SQLGetDiagRecW 的字符中的 BufferLength .....	24
2.11	SQLGetDiagRec 中的 GBase 8s 和 ISAM 错误描述.....	25
2.12	提高单线程应用程序的性能.....	25
2.13	部分支持的和不支持的 ODBC 特性.....	25
2.13.1	事务处理.....	25
2.13.2	ODBC 游标.....	26
2.13.3	ODBC 书签.....	26
2.13.4	SQLBulkOperations.....	27
2.13.5	SQLDescribeParam.....	27
2.13.6	不支持的 Microsoft ODBC 驱动程序特性 .....	28
3	配置数据源.....	28
3.1	在 UNIX 上配置 DSN.....	28
3.1.1	odbcinst.ini 文件 .....	29
3.1.2	odbc.ini 文件.....	30
3.1.3	ODBC 部分.....	37
3.1.4	设置 \$ODBCINI 环境变量 .....	38
3.1.5	.netrc 文件.....	38
3.2	在 Windows 中配置 DSN .....	38
3.2.1	配置新的用户 DSN 或系统 DSN.....	40
3.2.2	移除 DSN.....	44
3.2.3	重新配置现有的 DSN.....	44
3.2.4	配置文件 DSN.....	44
3.2.5	创建调用驱动程序的日志.....	45
3.3	制作连接的连接字符串关键字.....	45
3.4	DSN 迁移工具.....	46
3.4.1	设置和使用 DSN 迁移工具.....	47
3.4.2	DSN 迁移工具示例.....	47
4	数据类型.....	48
4.1	数据类型.....	48
4.2	SQL 数据类型 .....	48
4.2.1	标准 SQL 数据类型 .....	48
4.2.2	GLS 的附加 SQL 数据类型.....	51
4.2.3	GBase 8s 的附加 SQL 数据类型 .....	52
4.2.4	精度、小数位数、长度和显示大小 .....	52

4.3 C 数据类型 .....	58
4.3.1 C 间隔结构 .....	59
4.3.2 传输数据 .....	61
4.4 报告标准 ODBC 类型 .....	61
4.4.1 SQL_INFX_ATTR_ODBC_TYPES_ONLY .....	61
4.4.2 SQL_INFX_ATTR_LO_AUTOMATIC .....	62
4.4.3 SQL_INFX_ATTR_DEFAULT_UDT_FETCH_TYPE .....	62
4.4.4 报告宽字符列 .....	63
4.4.5 报告标准 ODBC 数据类型的 DSN 设置 .....	63
4.5 转换数据 .....	64
4.5.1 标准转换 .....	64
4.5.2 GLS 的附加转换 .....	68
4.5.3 GBase 8s 的附加转换 .....	69
4.5.4 将数据由 SQL 转换为 C .....	70
4.5.5 由 C 转换为 SQL .....	79
5 智能大对象 .....	87
5.1 智能大对象的数据结构 .....	87
5.1.1 使用智能大对象数据结构 .....	88
5.2 智能大对象的存储 .....	88
5.2.1 磁盘存储信息 .....	89
5.2.2 创建时刻标志 .....	89
5.2.3 继承层级 .....	90
5.3 创建智能大对象的示例 .....	92
5.4 转移智能大对象 .....	103
5.5 访问智能大对象 .....	104
5.5.1 智能大对象自动化 .....	104
5.5.2 ifx_lo 函数 .....	105
5.6 检索智能大对象的状态 .....	120
5.7 从文件读取智能大对象，或将它写至文件 .....	132
6 行和集合 .....	133
6.1 分配和绑定行或集合缓冲区 .....	133
6.1.1 固定型缓冲区和非固定型缓冲区 .....	133
6.1.2 缓冲区和内存分配 .....	134
6.1.3 SQL 数据 .....	134

6.1.4 执行本地访存.....	135
6.1.5 从数据库检索行和集合的示例.....	135
6.2 在客户端创建行和列表的示例.....	146
6.3 修改行或集合.....	155
6.4 检索行或集合的信息.....	156
7 客户端函数.....	157
7.1 调用客户端函数.....	157
7.1.1 SQL 语法.....	157
7.1.2 函数语法.....	157
7.1.3 输入和输出参数.....	158
7.1.4 SQL_BIGINT 数据类型.....	158
7.1.5 返回码.....	158
7.2 智能大对象函数.....	159
7.2.1 ifx_lo_alter() 函数.....	159
7.2.2 ifx_lo_close() 函数.....	159
7.2.3 ifx_lo_col_info() 函数.....	160
7.2.4 ifx_lo_create() 函数.....	160
7.2.5 ifx_lo_def_create_spec() 函数.....	161
7.2.6 ifx_lo_open() 函数.....	162
7.2.7 ifx_lo_read() 函数.....	163
7.2.8 ifx_lo_readwithseek() 函数.....	163
7.2.9 ifx_lo_seek() 函数.....	164
7.2.10 ifx_lo_specget_estbytes() 函数.....	165
7.2.11 ifx_lo_specget_extsz() 函数.....	165
7.2.12 ifx_lo_specget_flags() 函数.....	166
7.2.13 ifx_lo_specget_maxbytes() 函数.....	166
7.2.14 ifx_lo_specget_sbspace() 函数.....	167
7.2.15 ifx_lo_specset_estbytes() 函数.....	167
7.2.16 ifx_lo_specset_extsz() 函数.....	168
7.2.17 ifx_lo_specset_flags() 函数.....	169
7.2.18 ifx_lo_specset_maxbytes() 函数.....	170
7.2.19 ifx_lo_specset_sbspace() 函数.....	170
7.2.20 ifx_lo_stat() 函数.....	171
7.2.21 ifx_lo_stat_atime() 函数.....	171

7.2.22 ifx_lo_stat_cspec() 函数 .....	172
7.2.23 ifx_lo_stat_ctime() 函数 .....	172
7.2.24 ifx_lo_stat_refcnt() 函数 .....	173
7.2.25 ifx_lo_stat_size() 函数 .....	173
7.2.26 ifx_lo_tell() 函数 .....	174
7.2.27 ifx_lo_truncate() 函数 .....	174
7.2.28 ifx_lo_write() 函数 .....	175
7.2.29 ifx_lo_writewithseek() 函数 .....	176
7.3 行和集合的函数 .....	177
7.3.1 ifx_rc_count() 函数 .....	177
7.3.2 ifx_rc_create() 函数 .....	177
7.3.3 ifx_rc_delete() 函数 .....	179
7.3.4 ifx_rc_describe() 函数 .....	179
7.3.5 ifx_rc_fetch() 函数 .....	180
7.3.6 ifx_rc_free() 函数 .....	181
7.3.7 ifx_rc_insert() 函数 .....	182
7.3.8 ifx_rc_isnull() 函数 .....	183
7.3.9 ifx_rc_setnull() 函数 .....	183
7.3.10 ifx_rc_typespec() 函数 .....	184
7.3.11 ifx_rc_update() 函数 .....	184
8 提高应用程序性能 .....	186
8.1 在数据传输过程中进行错误检查 .....	186
8.2 在 ODBC 中启用分隔标识符 .....	186
8.3 连接级别优化 .....	187
8.4 优化查询执行 .....	187
8.5 插入多行 .....	188
8.6 自动释放游标 .....	188
8.6.1 启用 AUTOFREE 功能 .....	188
8.6.2 AUTOFREE 功能 .....	189
8.7 延迟执行 SQL PREPARE 语句 .....	189
8.8 设置简单大对象访存数组大小 .....	190
8.9 SPL 输出参数功能 .....	191
8.10 OUT 和 INOUT 参数 .....	191
8.11 异步执行 .....	195

8.12 使用定位更新和删除更新数据 .....	195
8.13 BIGINT 和 BIGSERIAL 数据类型 .....	197
8.14 消息传输优化.....	197
8.14.1 消息链接限制.....	197
8.14.2 禁用消息链接.....	197
8.14.3 优化消息传输的错误.....	198
9 错误消息.....	199
9.1 诊断 SQLSTATE 值.....	200
9.2 将 SQLSTATE 值映射到 GBase 8s 错误消息.....	200
9.3 将 GBase 8s 错误消息映射到 SQLSTATE 值 .....	207
9.3.1 已弃用的和新的 GBase 8s ODBC Driver API.....	207
9.3.2 SQLAllocConnect （仅限核心级别） .....	208
9.3.3 SQLAllocEnv （仅限核心级别） .....	208
9.3.4 SQLAllocStmt （仅限核心级别） .....	209
9.3.5 SQLBindCol （仅限核心级别） .....	209
9.3.6 SQLBindParameter （仅限一级） .....	210
9.3.7 SQLBrowseConnect （仅限二级） .....	210
9.3.8 SQLCancel （仅限核心级别） .....	212
9.3.9 SQLColAttributes （仅限核心级别） .....	212
9.3.10 SQLColumnPrivileges （仅限二级） .....	213
9.3.11 SQLColumns （仅限一级） .....	213
9.3.12 SQLConnect （仅限核心级别） .....	214
9.3.13 SQLDataSources （仅限二级） .....	215
9.3.14 SQLDescribeCol （仅限核心级别） .....	215
9.3.15 SQLDisconnect .....	216
9.3.16 SQLDriverConnect （仅限一级） .....	217
9.3.17 SQLDrivers （仅限一级） .....	218
9.3.18 SQLError （仅限核心级别） .....	218
9.3.19 SQLExecDirect （仅限核心级别） .....	219
9.3.20 SQLExecute （仅限核心级别） .....	220
9.3.21 SQLExtendedFetch （仅限二级） .....	221
9.3.22 SQLFetch （仅限核心级别） .....	223
9.3.23 SQLForeignKeys （仅限二级） .....	224
9.3.24 SQLFreeConnect （仅限核心级别） .....	225



9.3.25 SQLFreeEnv (仅限核心级别)	225
9.3.26 SQLFreeStmt (仅限核心级别)	225
9.3.27 SQLGetConnectOption (仅限一级)	226
9.3.28 SQLGetCursorName (仅限核心级别)	226
9.3.29 SQLGetData (仅限一级)	226
9.3.30 SQLGetFunctions (仅限一级)	227
9.3.31 SQLGetInfo (仅限一级)	228
9.3.32 SQLGetStmtOption (仅限一级)	228
9.3.33 SQLGetTypeInfo (仅限一级)	229
9.3.34 SQLMoreResults (仅限二级)	229
9.3.35 SQLNativeSql (仅限二级)	230
9.3.36 SQLNumParams (仅限二级)	230
9.3.37 SQLNumResultCols (仅限核心级别)	231
9.3.38 SQLParamData (仅限一级)	231
9.4 SQLParamOptions (仅限核心和二级)	232
9.4.1 SQLPrepare	232
9.4.2 SQLPrimaryKeys (仅限二级)	233
9.4.3 SQLProcedureColumns (仅限二级)	234
9.4.4 SQLProcedures (仅限二级)	235
9.4.5 SQLPutData (仅限一级)	236
9.4.6 SQLRowCount (仅限核心级别)	236
9.4.7 SQLSetConnectOption (仅限一级)	237
9.4.8 SQLSetCursorName (仅限核心级别)	237
9.4.9 SQLSetStmtOption (仅限一级)	238
9.4.10 SQLSpecialColumns (仅限一级)	238
9.4.11 SQLStatistics (仅限一级)	239
9.4.12 SQLTablePrivileges (仅限二级)	240
9.4.13 SQLTables (仅限一级)	241
9.4.14 SQLTransact (仅限核心级别)	242
10 Unicode	242
10.1 Unicode 概述	242
10.1.1 Unicode 版本	242
10.2 ODBC 应用程序中的 Unicode	243
10.3 ODBC 应用程序中的 Unicode	244

10.3.1 配置.....	244
10.4 支持的 Unicode 函数 .....	245

# 1 简介

## 1.1 关于本出版物

本简介概述了本出版物中的信息，并描述了所使用的约定。

本手册作为 GBase 8s ODBC Driver 的用户指南和参考，这是 Microsoft™“开放数据库互连”（ODBC）接口 Version 3.0 的 GBase 8s 实现。

本手册说明如何使用 GBase 8s ODBC Driver 应用程序编程接口（API），来访问 GBase 8s 数据库，并与 GBase 8s 数据库服务器交互。

本文档是为 C 编程人员编写的，他们使用 GBase 8s ODBC Driver 来访问 GBase 8s 数据库。

这些主题假定您具有下列背景：

- 使用计算机、操作系统和操作系统提供的实用程序的知识
- 使用关系型或对象-关系型数据库的一定经验，或接触过关系数据库概念
- C 编程语言

要了解关于软件兼容性的信息，请参阅 GBase 8s Client SDK 发版说明。

这些主题出自《GBase 8s ODBC Driver 程序员指南》。

## 1.2 符合行业标准

GBase 8s 产品符合各种标准。

基于 SQL 的 GBase 8s 产品完全兼容 SQL-92 入门级（发布为 ANSI X3.135-1992），这与 ISO 9075:1992 完全相同。另外，GBase 8s 数据库服务器的许多功能都遵守 SQL-92 中级和完全级别以及 X/Open SQL 公共应用程序环境 (CAE) 标准。

## 1.3 演示数据库

DB-Access 实用程序随 GBase 8s 数据库服务器产品一起提供，它包括一个或多个以下演示数据库：

- **stores\_demo** 数据库以一家虚构的体育用品批发商的有关信息举例说明了关系模式。GBase 8s 出版物中的许多示例均基于 **stores\_demo** 数据库。
- **superstores\_demo** 数据库举例说明了对象关系模式。**superstores\_demo** 数据库包含扩展数据类型、类型和表继承以及用户定义的例程的示例。

有关如何创建和填充演示数据库的信息，请参阅《GBase 8s DB-Access 用户指南》。有关数据库及其内容的描述，请参阅《GBase 8s SQL 指南：参考》。

用于安装演示数据库的脚本位于 UNIX™ 平台上的 \$GBS\_HOME/bin 目录和 Windows™ 环境中的 %GBS\_HOME%\bin 目录中。

## 1.4 示例代码约定

SQL 代码的示例在整个出版物中出现。除非另有说明，代码不特定于任何单个的 GBase 8s 应用程序开发工具。

如果示例中仅列出 SQL 语句，那么它们将不用分号定界。例如：您可能看到以下示例中的代码：

```
CONNECT TO stores_demo
...
DELETE FROM customer
      WHERE customer_num = 121
...
COMMIT WORK
DISCONNECT CURRENT
```

要将此 SQL 代码用于特定产品，必须应用该产品的语法规则。例如，如果使用的是 SQL API，那么必须在每条语句的开头使用 EXEC SQL，并在每条语句的结尾使用分号（或其他合适的定界符）。如果使用的是 DB - Access，那么必须用分号将多条语句隔开。

**提示：** 代码示例中的省略点表示在整个应用程序中将添加更多的代码，但是不必显示它以描述正在讨论的概念。

有关使用特定应用程序开发工具或 SQL API 的 SQL 语句的详细指导，请参阅您的产品文档。

## 1.5 语法图

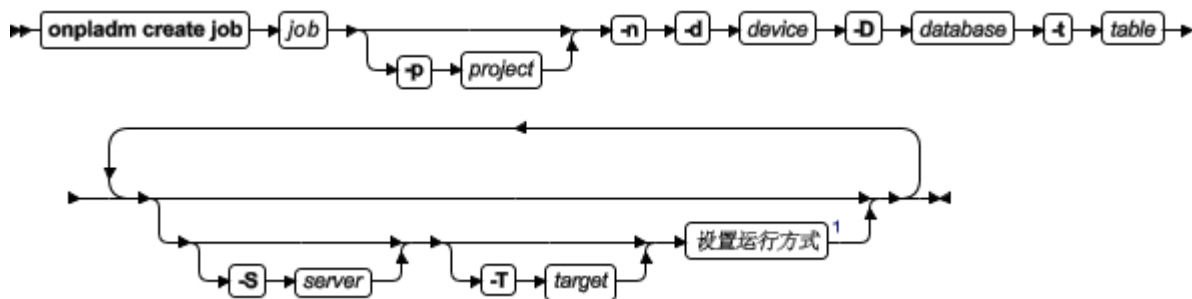
语法图使用特殊组件描述语句和命令的语法。

### 1.5.1 如何阅读命令行语法图

命令行语法图使用类似于其他语法图元素的元素。

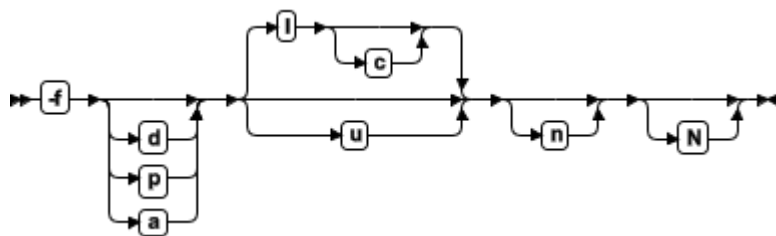
某些元素列于语法图中的表中。

**创建非转换作业**



此图中有一个名为“设置运行方式”的段，根据图脚注，这个段在第 Z-1 页上。如果这是真正的交叉引用，那么您可以在附录 Z 的第一页上找到此段。但在此处，此段显示在以下段图表中。请注意：该图使用段开头和结束部分。

### 设置运行方式



要了解如何正确构造命令，请从主图的左上角开始。遵循右边的图表，包括想要的元素。此图中的元素区分大小写，因为它们说明实用程序的语法。其他类型的语法（例如 SQL）则不区分大小写。

“创建非转换作业”图表说明了以下步骤：

1. 输入 `onpladm create job`，然后输入作业的名称。
2. 或者，输入 `-p`，然后输入项目的名称。
3. 输入以下所需的元素：
  - `-n`
  - `-d` 和设备的名称
  - `-D` 和数据库的名称
  - `-t` 和表的名称
4. 或者，可以选择一个或多个以下元素并重复它们任意次：
  - `-S` 和服务名称
  - `-T` 和目标服务器名称
  - 运行方式。要设置运行方式，请遵循“设置运行方式”段图表来输入 `-f`，或者输入 `d`、`p` 或 `a`，然后可选择输入 `l` 或 `u`。
5. 遵循图表直至终止符。

### 1.5.2 关键字和标点

关键字是为语句和除了系统级别命令的所有命令保留的词语。

当关键字出现在语法图表中时，它以大写字母显示。在命令中使用关键字时，可用大写或小写字母写关键字，但是必须严格按照语法图表中所显示的来拼写关键字。

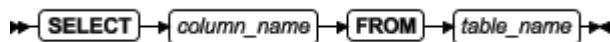
还必须严格按照语法图表中所显示的在语句和命令中使用标点。

### 1.5.3 标识和名称

变量作为语法图表和示例中标识符和名称的占位符。

根据上下文，可用任意名称、标识符或文字替换变量。变量也用来代表附加语法图表中扩展的复杂语法元素。当变量出现在语法图表、示例或文本中时，它以斜体小写字母显示。

下列语法图使用变量来说明简单 `SELECT` 语句的一般格式。



当编写此格式的 `SELECT` 语句时，请使用特定的列和表名称来替换 `column_name` 和 `table_name` 变量。

## 2 GBase 8s ODBC Driver 概述

这些主题介绍 GBase 8s ODBC Driver，并描述它的优势和架构。这些主题还描述符合性、隔离和锁级别、库，以及环境变量。

### 2.1 什么是 GBase 8s ODBC Driver?

“开放数据库互连”（ODBC）是数据库应用程序编程接口（API）的一种规范。

Microsoft<sup>™</sup> ODBC Version 3.0 基于 X/Open 和 International Standards

Organization/International Electromechanical Commission (ISO/IEC) 的 Call Level Interface 规范。ODBC 支持带有 C 函数库的 SQL 语句。应用系统调用这些函数来实现 ODBC 功能。

ODBC 应用程序使得您能够执行下列操作：

- 连接至数据源，或从数据源断开连接
- 检索关于数据源的信息
- 检索关于 GBase 8s ODBC Driver 的信息
- 设置和检索 GBase 8s ODBC Driver 选项
- 准备和发送 SQL 语句
- 检索 SQL 结果，并动态地处理结果
- 检索关于 SQL 结果的信息，并动态地处理该信息

在结果可用之前或之后，ODBC 都允许您为结果分配存储。此特性允许您确定结果和要采取的行动，而不受预定义的数据结构强加的限制。

ODBC 不需要预处理器来编译应用程序。

ODBC 支持“安全套接层”（SSL）。

### 2.1.1 GBase 8s ODBC Driver 特性

GBase 8s ODBC Driver 实现 Microsoft™ “开放数据库互连”（ODBC）Version 3.0 标准。

GBase 8s ODBC Driver 产品还提供下列特性和功能：

- Data Source Name (DSN) 迁移
- Driver Manager Replacement Module, 支持 ODBC 2.x 应用程序与 ODBC 驱动程序 Version 3.00 之间的兼容性。
- Microsoft Transaction Server (MTS), 允许您开发、运行和管理可伸缩的、基于组件的 Internet 和 intranet 服务器应用程序的一种环境。MTS 执行下列任务：
  - 管理系统资源, 包括进程、线程和数据库连接, 以便于应用程序可缩放满足许多并发用户
  - 管理服务器组件创建、执行和删除
  - 自动地开始和控制事务, 实现应用程序的可靠性
  - 实现安全性, 以便于未获授权的用户不可访问应用程序
  - 为配置、管理和部署提供工具

**重要：** 如果想要以 GBase 8s ODBC Driver 来使用由 MTS 管理的分布式事务, 则必须启用连接池。

- 扩展的数据类型, 包括 row 和 collection
- 长标识符
- 对书签的有限支持
- GLS 数据类型
- 广泛的错误检测
- Unicode 支持
- XA 支持
- 对互联网 128 位协议的 Internet Protocol Version 6 支持（要获取更多信息, 请参阅《GBase 8s 管理员指南》。）

#### 支持扩展的数据类型

GBase 8s ODBC Driver 支持扩展的数据类型。

GBase 8s ODBC Driver 支持下列扩展的数据类型：

- 集合（LIST、MULTISET、SET）
- DISTINCT
- OPAQUE（固定的、未命名的）
- row（命名的, 未命名的）
- 智能大对象（BLOB、CLOB）
- 支持某些扩展的数据类型的客户机函数

#### 支持 GLS 数据类型

GBase 8s ODBC Driver 支持 GLS 数据类型。

GBase 8s ODBC Driver 支持下列 GLS 数据类型：

- NCHAR

- NVARCHAR

#### 扩展的错误检测

GBase 8s ODBC Driver 检测 XA 类型错误。

### 2.1.2 某些 ODBC 函数参数的附加的值

GBase 8s ODBC Driver 支持某些 ODBC 参数的附加的值。

某些 ODBC 函数参数的这些附加的值包括：

- SQLColAttributes 的 *fDescType* 值
  - SQL\_INFX\_ATTR\_FLAGS
  - SQL\_INFX\_ATTR\_EXTENDED\_TYPE\_ALIGNMENT
  - SQL\_INFX\_ATTR\_EXTENDED\_TYPE\_CODE
  - SQL\_INFX\_ATTR\_EXTENDED\_TYPE\_NAME
  - SQL\_INFX\_ATTR\_EXTENDED\_TYPE\_OWNER
  - SQL\_INFX\_ATTR\_SOURCE\_TYPE\_CODE
- SQLGetInfo 的 *fInfoType* 返回值
  - SQL\_INFX\_LO\_PTR\_LENGTH
  - SQL\_INFX\_LO\_SPEC\_LENGTH
- SQL\_INFX\_LO\_STAT\_LENGTH
- SQLGetConnectOption 和 SQLSetConnectOption 的 *fOption* 值：  
SQL\_INFX\_OPT\_LONGID
- SQLGetConnectOption 和 SQLSetConnectOption 的 *fOption* 值：  
SQL\_ATTR\_ENLIST\_IN\_DTC

## 2.2 ODBC 组件概述

随同 GBase 8s ODBC Driver 的 ODBC 包括几个组件。

随同 GBase 8s ODBC Driver 的 ODBC 可包括下列组件：

- Driver manager

可链接至驱动程序管理器的应用程序，其链接至数据源指定的驱动程序。驱动程序管理程序还检查参数和事务。在大多数 UNIX™ 平台上，都可从第三方供应商购买 ODBC Driver Manager。

在 Microsoft™ Windows™ 平台上，ODBC Driver Manager 是 OS 的一部分。

- GBase 8s ODBC Driver

驱动程序提供至 GBase 8s 数据库服务器的接口。应用程序可在下列配置中使用驱动程序：

- 要链接至 ODBC 驱动程序管理器
- 要链接至 Driver Manager Replacement & 驱动程序
- 要直接链接至驱动程序



- 数据源

驱动程序提供至下列数据源的访问：

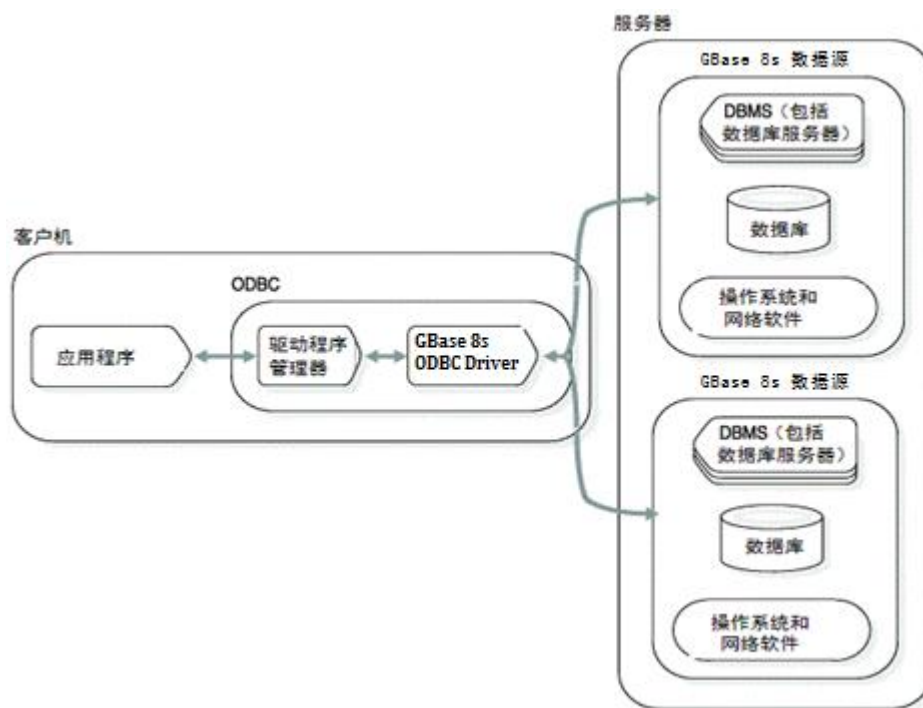
- 包括数据库服务器的数据库管理系统（DBMS）
- 数据库
- 访问数据库所需的操作系统和网络软件

### 2.2.1 带有驱动程序管理器的 GBase 8s ODBC Driver

当在系统中包括驱动程序管理器时，有一种软件体系结构。

当在系统中包括驱动程序管理器时，下图展示软件体系结构。在这样的系统中，驱动程序和驱动程序管理器就像处理函数调用的单个单元一样行动。

图：带有驱动程序管理器的 GBase 8s ODBC Driver

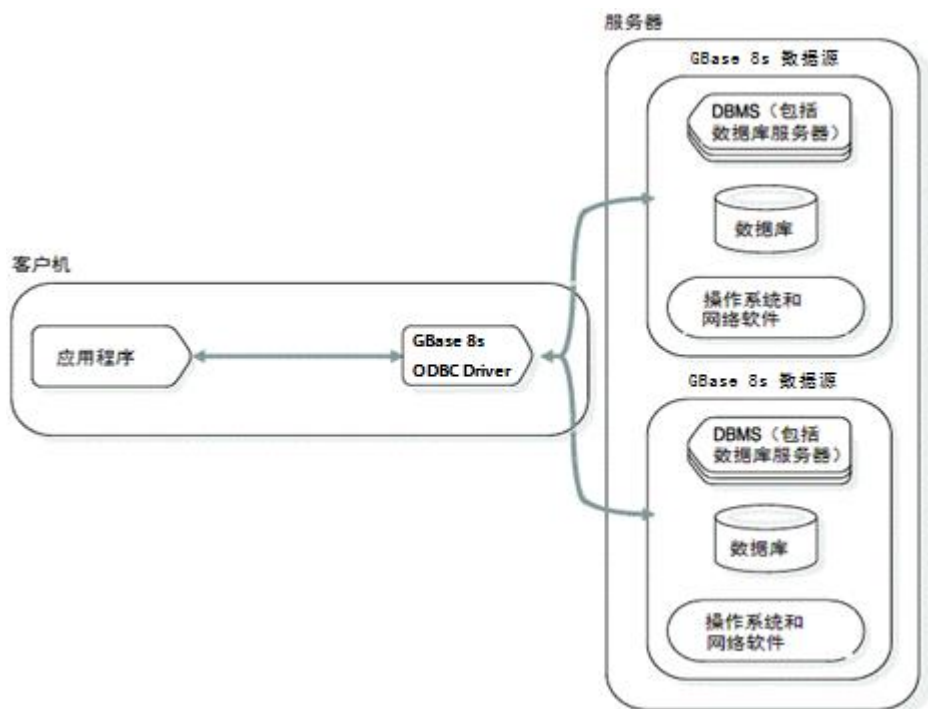


### 2.2.2 不带有驱动程序管理器的 GBase 8s ODBC Driver (UNIX™)

当系统中不包括驱动程序管理器时，有一种软件体系结构。

下图展示使用不带有驱动程序管理器的 GBase 8s ODBC Driver 的应用程序。在此情况下，该应用程序必须链接至 GBase 8s ODBC Driver 库。

图：不带有驱动程序管理器的 GBase 8s ODBC Driver

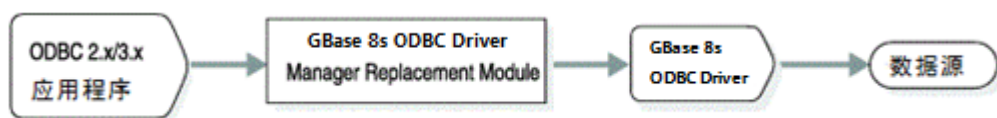


## 2.3 带有 DMR 的 GBase 8s ODBC Driver

GBase 8s ODBC Driver 包括一个“驱动程序管理器替代”（DMR）库。该 DMR 替代驱动程序管理器不可用的平台上的驱动程序管理器。

下图展示带有 DMR 的 ODBC 配置。

图：驱动程序管理器替代模块的体系结构



直接链接至 ODBC Version 4.10 驱动程序和 DMR 的应用程序不需要 ODBC Driver Manager。

除了支持 ODBC Version 4.10 特性之外，DMR 还支持 ODBC 2.x 应用程序与 GBase 8s ODBC Driver Version 3.00 之间的兼容性。要与 ODBC 2.x 应用程序相兼容，应用程序必须通过 DMR 或通过 ODBC Version 4.10 驱动程序管理器，来连接至 GBase 8s ODBC Driver Version 3.00。

不可使用 GBase 8s DMR 来连接至非 GBase 8s 数据源。DMR 不支持连接池。DMR 不在 Unicode 与 ANSI API 之间映射。

## 2.4 GBase 8s ODBC Driver 组件

GBase 8s ODBC Driver 包括四个组件。

GBase 8s ODBC Driver 包括下列组件：

- 环境变量
- 标头文件
- 数据类型
- 库

### 2.4.1 环境变量

您必须为驱动程序设置的四个环境变量。

以下列表描述您必须为驱动程序设置的环境变量。要获取关于环境变量的更多信息，请参阅《GBase 8s SQL 指南：参考》。

#### GBS\_HOME

安装 GBase 8s Client Software Development Kit 处的目录的完全路径。

在 Windows™ 平台上，GBS\_HOME 是注册设置，而不是环境变量。在安装期间设置它。

#### PATH

搜索可执行程序的路径。PATH 的设置必须包括至 \$GBS\_HOME/bin 目录的路径。

#### DBCENTURY（可选的）

控制年值的设置。当用户发出包含仅指定年份的最后两个数字的 date 或 datetime 字符串的语句时，DBCENTURY 影响客户机程序。例如：

```
insert into datetable (datecol) values ("01/01/01");
```

数据库服务器将此语句中指定的日期存储为 01-01-1901 或 01-01-2001，这取决于客户机上的 DBCENTURY 值。

#### GL\_DATE（可选的）

GL\_DATE 控制日期的解释。例如，您可指定日期格式为 mm-dd-yyyy 或 yyyy-mm-dd。

### 在 UNIX™ 上设置环境变量

如果您在命令行处设置环境变量，则每当登录至系统时，都必须重置它们。如果在文件中设置环境变量，则当登录到系统时，会自动设置它们。

GBase 8s ODBC Driver 在 \$GBS\_HOME/etc 中提供一个名为 setup.odbc 的样例设置文件。您可使用此文件，来为驱动程序设置环境变量。下面列表描述 setup.odbc 中的环境变量。

#### GBS\_HOME

安装 GBase 8s Client Software Development Kit 处的目录的完全路径。

#### GBASEDBTSQLHOSTS

此值是可选的。它指定包含 sqlhosts 的目录。在缺省情况下，sqlhosts 在 \$GBS\_DATA/conf 中。

#### ODBCINI

此值是可选的。您可使用它来为 odbc.ini 文件指定替代的位置。缺省位置为您的

home 目录。

## 在 Windows 中设置环境变量

如果在命令行处设置环境变量，则必须在登录至 Windows™ 环境时重置它们。然而，如果在 Windows 注册表中设置它们，则在登录时会自动设置。

GBase 8s ODBC Driver 在 Windows 注册表中的下列位置中存储环境变量：

`\HKEY_CURRENT_USERS\Software\Gbasedbt\Environment`

在 Windows 环境中，您必须使用 `setnet32.exe` 或直接更新注册表的工具，来设置诸如 `iclit09b.dll` 这样的 GBase 8s 动态链接库（DLL）使用的环境变量。Setnet 实用程序只能用于设置 GBase 8s 环境变量。

可根据开发环境的需要来使用环境变量。例如，编译器需要知道在哪里找到 `include` 文件。要指定 `include` 文件的位置，请设置环境变量 `GBS_HOME`（或某其他环境变量），然后将 `include` 路径设置为 `GBS_HOME\incl\cli`。

设置环境变量的选项有下列优先顺序：

1. Setnet 实用程序
2. 命令行
3. Windows 注册表

## 2.4.2 标头文件

您可使用 `sql.h` 和 `sqlext.h` 标头文件，来运行 GBase 8s ODBC Driver，其为 Microsoft™ 编译器的一部分。

要运行 GBase 8s 扩展，请包括安装在 `GBS_HOME/incl/cli` 中的 `infxcli.h` 文件。此文件定义 GBase 8s ODBC Driver 常量和类型，并为 GBase 8s ODBC Driver 环境提供函数 `prototype`。如果包括 `infxcli.h` 文件，则它自动地包括 `sql.h` 和 `sqlext.h` 文件。

`sql.h` 和 `sqlext.h` 标头文件包含 C 数据类型的定义。

请在 XA ODBC 应用程序中包括 `xa.h` 标头文件。Windows™ 上的 ODBC 应用程序需要 GBase 8s Client Software Development Kit 来编译。在重新编译它们之前，ODBC 驱动程序的应用程序可能需要在 `PATH` 环境变量中包括 Client SDK 的位置。

## 2.4.3 数据类型

存储在数据源上的一列数据有一个 SQL 数据类型。

GBase 8s ODBC Driver 将特定于 GBase 8s 的 SQL 数据类型映射为以 ODBC SQL 语法定义的 ODBC SQL 数据类型。（驱动程序通过 `SQLGetTypeInfo` 返回这些映射。它还使用 ODBC SQL 数据类型来描述 `SQLColAttributes` 和 `SQLDescribeCol` 中的列和参数的数据类型）。

每一 SQL 数据类型对应于一个 ODBC C 数据类型。在缺省情况下，驱动程序假设存储位置的 C 数据类型对应于该位置绑定至其的列或参数的 SQL 数据类型。如果存储位置的 C 数据类型不是缺省的 C 数据类型，则应用程序可以使用 SQLBindCol 的 *TargetType*、SQLGetData 的 *fCType* 参数，以及 SQLBindParameter 中的 *ValueType* 参数来指定正确的 C 数据类型。在驱动程序从数据源返回数据之前，它将数据转换为指定的 C 数据类型。在驱动程序将数据发送至数据源之前，它将数据从指定的 C 数据类型转换为 SQL 数据类型。

GBase 8s 数据类型名称不同于 Microsoft<sup>™</sup> ODBC 数据类型名称。要获取关于这些差异的信息，请参阅《GBase 8s ODBC Driver 程序员指南》。

## 2.4.4 库

对于 UNIX<sup>™</sup> 和 Windows<sup>™</sup>，有一个安装库的安装过程。

### UNIX

该安装过程将下列库安装至 GBS\_HOME/lib/cli 内。在 odbc.ini 文件中的每一数据源规范部分中，将驱动程序值指定为下列库文件名称之一的完整路径。

- libifcli.a 或 libcli.a  
    单个（非线程的）库的静态版本
- libifcli.so 或 iclis09b.so  
    单个（非线程的）库的共享版本
- libthcli.a  
    多线程库的静态版本
- libthcli.so 或 iclit09b.so  
    多线程库的共享版本
- libifdrm.so 或 idmrs09a.so  
    SMR（线程安全的）共享库

如果未使用驱动程序管理器，则应用程序需要连接至 GBase 8s ODBC Driver 库的静态或共享版本。

下列编译命令将应用程序链接至 GBase 8s ODBC Driver 库的线程安全版本：

```
cc ... -L$GBS_HOME/lib/cli -lifdmr -lthcli
```

### Windows

安装过程将下列库安装至 GBS\_HOME\lib 内。

- iclit09b.lib  
    直接启用至驱动程序的链接，而不使用驱动程序管理器
- iregt07b.lib  
    允许直接链接至 iregt07b.dll

下列编译命令将应用程序链接至 GBase 8s ODBC Driver 库的线程安全版本：

```
cl ... -L$GBS_HOME/lib/cli iclit09b.lib
```

如果使用驱动程序管理器，则必须将应用程序仅链接至驱动程序管理器库，如下例所示：

```
cl odbc32.lib
```

GBase 8s ODBC Driver 需要 Version 3.0 驱动程序管理器。

## 2.5 GBase 8s ODBC Driver API

应用程序使用 GBase 8s ODBC Driver API 建立与数据源的连接，将 SQL 语句发送至数据源，动态地处理结果数据，并终止连接。

驱动程序允许应用程序执行下列步骤：

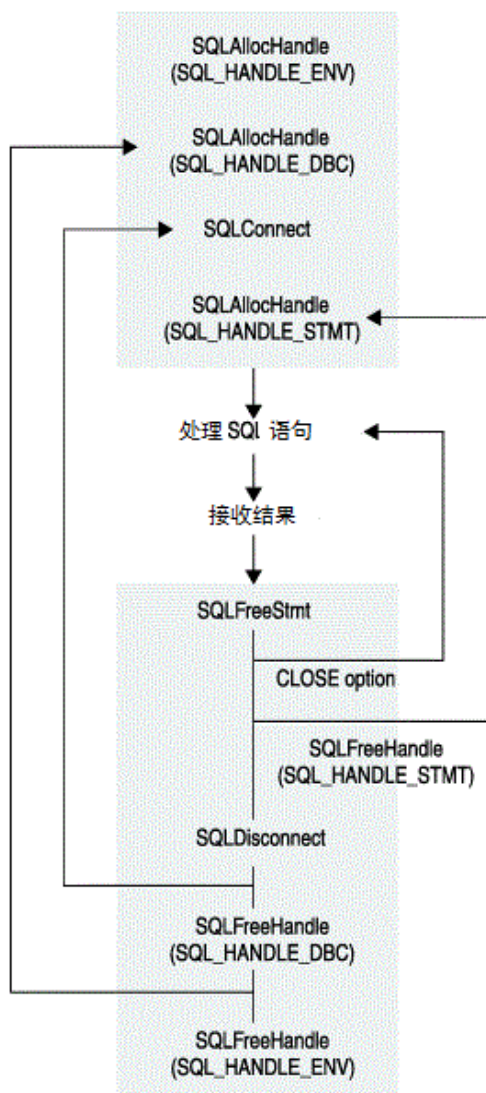
1. 连接至数据源。  
可以通过 DSN 连接来连接数据源，也可以使用 DSN-less 连接字符串。指定数据源的名称和完成连接所需的任何附加信息。
2. 处理一个或多个 SQL 语句：
  - a. 将 SQL 文本字符串置于缓冲区中。如果该语句包括参数标记，则设置参数值。
  - b. 如果该语句返回结果集，则为该语句分配一个游标名称，或让驱动程序分配一个。
  - c. 准备该语句，或者提交立即执行。
  - d. 如果该语句创建结果集，则可查询结果集的属性，诸如列数，以及特定列的名称和类型。对于结果集中每一列，请指定存储和访存结果。
  - e. 如果该语句导致错误，则从驱动程序检索错误信息，并采取恰当的措施。
3. 通过提交或回滚来结束任何事务。
4. 当应用程序结束与数据源的交互时，终止连接。

每个 GBase 8s ODBC Driver 函数名称都以前缀 SQL 开始。每一函数接受一个或多个参数。定义参数作为（至驱动程序的）输入或（来自驱动程序的）输出。

下图展示了应用程序进行的基本函数调用，即使应用程序通常也调用其他函数。

图: GBase 8s ODBC Driver 应用程序进行的函数调用的示例列表





### 2.5.1 环境、连接和语句句柄

当应用程序请求它时，驱动程序和驱动程序管理器为关于环境、每一连接和每一 SQL 语句的信息分配存储。

对于这些分配，驱动程序为每一应用程序返回一个句柄，在每一函数调用中，其使用一个或多个句柄。

GBase 8s ODBC Driver API 使用下列句柄类型：

#### 环境句柄

环境句柄为全局信息标识内存存储，包括有效的连接句柄和当前活动的连接句柄。环境句柄为 *henv* 变量类型。一个应用程序使用一个环境句柄。在它连接至数据源之前，它必须请求此句柄。

#### 连接句柄

连接句柄为关于特殊连接的信息标识内存存储。连接句柄为 *hdbc* 变量类型。应用程序必须在连接至数据源之前，必须请求连接句柄。每个连接句柄都与环境句柄相关联。但是，环境句柄可以与多个连接句柄相关联。

#### 语句句柄

语句句柄为关于 SQL 语句的信息标识内存存储。语句句柄为 *hstmt* 变量类型。在

应用程序提交 SQL 请求之前，它必须请求一语句句柄。每一语句句柄仅与一个连接句柄相关联。但是，每一连接句柄可与多个语句句柄相关联。

## 2.5.2 缓冲区

应用程序在输入缓冲区中将数据传给驱动程序。驱动程序在输出缓冲区中将数据返回给应用程序。

应用程序必须同时为输入和输出缓冲区分配内存。如果应用程序使用缓冲区来检索字符串数据，则该缓冲区必须包含空终止字节的空间。

有些函数接受指向缓冲区的指针，稍后由其他函数来使用。应用程序必须确保这些指针保持有效，直到所有适用的函数都已使用了它们。例如，SQLBindCol中的参数 *rgbValue* 指向 SQLFetch 返回列的数据的输出缓冲区。

### 输入缓冲区

应用程序将输入缓冲区的地址和长度传给驱动程序。

缓冲区的长度必须为下列值之一：

- 大于或等于零的长度  
此值为输入缓冲区中数据的实际长度。对于字符数据，长度零指示该数据为空字符串（零长度）。零长度不同于空指针。如果应用程序指定字符数据的长度，则字符数据不需要以空字符结尾。
- SQL\_NTS  
此值指定字符串数据值为空终止的。
- SQL\_NULL\_DATA  
此值告诉驱动程序忽略输入缓冲区中的值，并替代使用 NULL 数据值。仅当输入缓冲区提供 SQL 语句中参数的值时，它才是有效的。

对于包含嵌入的空字符的字符数据，GBase 8s ODBC Driver函数的操作是未定义的；出于最大可操作性的考虑，最好不要使用它们。GBase 8s 数据库服务器将空字符处理为字符串结束标志，或作为不再存在数据的指示符。

除非在函数描述中禁止它，否则，输入缓冲区的地址可为空指针。在此情况下，对应的缓冲区长度参数的值将被忽略。

### 输出缓冲区

应用程序将参数传给驱动程序，以便于驱动程序可在输出缓冲区中返回数据。

这些参数为：

- 输出缓冲区的地址，驱动程序将数据返回这里  
除非在函数描述中禁止它，否则，输出缓冲区的地址可为空指针。在此情况下，驱动程序不在缓冲区中返回任何内容，在没有错误时，返回 SQL\_SUCCESS。



如有必要，在返回它之前，驱动程序转换数据。在返回它之前，驱动程序始终空终止字符数据。

- 缓冲区的长度

如果返回的数据有固定的 C 长度，比如整数、实数或日期结构，则驱动程序忽略此值。

- 驱动程序返回数据长度的变量地址（长度缓冲区）

如果在结果集中数据为空值，则返回的数据长度为 SQL\_NULL\_DATA。否则，返回的数据长度为可用于返回的数据的字节数。如果驱动程序转换该数据，则返回的数据长度为转换之后保留的字节数；对于字符数据，它不包括驱动程序添加的空终止字节。

如果输出缓冲区太小，则驱动程序尝试截断数据。如果截断未导致有效数据的丢失，则驱动程序在输出缓冲区中返回截断的数据，在长度缓冲区中返回可用数据的长度（与截断的数据的长度相反），并返回 SQL\_SUCCESS\_WITH\_INFO。如果截断导致有效数据的丢失，则驱动程序保持输出和长度缓冲区不变，并返回 SQL\_ERROR。应用程序调用 SQLGetDiagRec 来检索关于截断或错误的信息。

2.5.3 SQLGetInfo 尝试实现

GBase 8s 实现 GBase 8s ODBC Driver 的 SQLGetInfo 参数。

下表描述 GBase 8s ODBC Driver 的 SQLGetInfo 参数的 GBase 8s 实现。

参数名称	GBase 8s 实现
SQL_ACTIVE_ENVIRONMENTS	GBase 8s 驱动程序没有对活动环境数的限制。始终返回零。
SQLAggregate_FUNCTIONS	GBase 8s 驱动程序返回数据库服务器支持的所有聚集函数。
SQL_ASYNC_MODE	GBase 8s 驱动程序返回 SQL_AM_NONE。
SQL_ATTR_METADATA_ID	GetInfo 和 PutInfo 支持
SQL_BATCH_ROW_COUNT	GBase 8s 驱动程序返回位掩码零。
SQL_BATCH_SUPPORT	GBase 8s 驱动程序返回位掩码零。
SQL_CA1_POS_DELETE	支持对 SQLSetPos 调用中的操作参数
SQL_CA1_POS_POSITION	支持对 SQLSetPos 调用中的操作参数
SQL_CA1_POS_REFRESH	支持对 SQLSetPos 调用中的操作参数
SQL_CA1_POS_UPDATE	支持对 SQLSetPos 调用中的操作参数

参数名称	GBase 8s 实现
<b>SQL_CA1_POSITIONED_DELETE</b>	当游标为 forward-only 游标时, 支持 DELETE WHERE CURRENT OF SQL 语句。
<b>SQL_CA1_POSITIONED_UPDATE</b>	当游标为 static-only 游标时, 支持 UPDATE WHERE CURRENT OF SQL 语句。(符合 SQL-92 入门级的驱动程序始终急哦根据支持返回此选项。)
<b>SQL_CA1_LOCK_NO_CHANGE</b>	当游标为 static-only 游标时, 在对 SQLSetPos 的调用中支持 SQL_LOCK_NO_CHANGE 的 LockType 参数。
<b>SQL_CA1_SELECT_FOR_UPDATE</b>	当游标为 forward-only 游标时, 支持 SELECT FOR UPDATE SQL 语句。(符合 SQL-92 入门级的驱动程序始终急哦根据支持返回此选项。)
<b>SQL_CATALOG_NAME</b>	GBase 8s 驱动程序返回 'Y'
<b>SQL_COLLATION_SEQ</b>	INTERSOLV DataDirect ODBC Driver返回 InfoValuePtr (未更改的)
<b>SQL_DDL_INDEX</b>	返回位掩码 SQL_DL_CREATE_INDEX   SQL_DL_DROP_INDEX
<b>SQL_DESCRIBE_PARAMETER</b>	返回 'N'; 不可描述参数。(这是因为最新的 GBase 8s 数据库服务器支持函数重载, 以便于带有相同该名称的多个函数可接受不同的参数类型。)
<b>SQL_DIAG_DYNAMIC_FUNCTION</b>	返回空字符串
<b>SQL_DROP_TABLE</b>	返回位掩码 SQL_DT_DROP_TABLE   SQL_DT_CASCADE   SQL_DT_RESTRICT
<b>SQL_DROP_VIEW</b>	返回位掩码 SQL_DV_DROP_TABLE   SQL_DV_CASCADE   SQL_DV_RESTRICT
<b>SQL_INDEX_KEYWORDS_</b>	SQL_LK_ASC   SQL_LK_DESC
<b>SQL_INSERT_STATEMENT</b>	返回位掩码 SQL_IS_INSERT_LITERALS   SQL_INSERT_SEARCHED   SQL_IS_SELECT_INTRO

参数名称	GBase 8s 实现
SQL_MAX_DRIVER_CONNECTIONS	返回零
SQL_MAX_IDENTIFIER_LEN	返回不同的值，这依赖于数据库服务器容量
SQL_ODBC_INTERFACE_CONFORMANCE	返回 SQL_OIC_CORE
SQL_PARAM_ARRAY_ROW_COUNTS	返回 SQL_PARC_NO_BATCH
SQL_PARAM_ARRAY_SELECTS	返回 SQL_PAS_NO_SELECT
SQL_SQL_CONFORMANCE	返回 SQL_OSC_CORE
SQL_SQL92_FOREIGN_KEY_DELETE_RULE	返回位掩码零
SQL_SQL92_FOREIGN_KEY_UPDATE_RULE	返回位掩码零
SQL_SQL92_GRANT	返回位掩码零
SQL_SQL92_NUMERIC_VALUE_FUNCTIONS	返回位掩码零
SQL_SQL92_PREDICATES	返回位掩码零
SQL_SQL92_RELATIONAL_JOIN_OPERATORS	返回位掩码零
SQL_SQL92_REVOKE	SQL_SR_CASCADE   SQL_SR_RESTRICT
SQL_SQL92_ROW_VALUE_CONSTRUCTOR	返回位掩码零
SQL_SQL92_STRING_FUNCTIONS	返回位掩码零
SQL_SQL92_VALUE_EXPRESSIONS	返回位掩码零
SQL_STANDARD_CLI_CONFORMANCE	返回位掩码 SQL_SCC_XOPEN_CLI_VERSION1   SQL_SCC_ISO92_CLI
SQL_STATIC_CURSOR_ATTRIBUTES1	仅可滚动的
SQL_STATIC_CURSOR_ATTRIBUTES2	仅可滚动的
SQL_XOPEN_CLI_YEAR	返回字符串 “1995”

## 2.6 全球语言支持

GBase 8s 产品可支持多种语言、文化的代码集。“全球语言支持”（GLS）提供对所有特定于语言和特定于文化的信息的支持。

下表描述如何根据您的平台来设置 GLS 选项。

平台	如何设置 GLS 选项
UNIX™	在 odbc.ini 文件中指定 GLS 选项。
Windows™	在 GBase 8s ODBC Driver DSN Setup 对话框中指定 GLS 选项。

下表描述 GBase 8s ODBC Driver 的 GLS 选项。

GLS 选项	描述
客户机语言环境	<p>描述： 应用程序在其中运行的语言环境和代码集</p> <p>格式： locale.codeset@modifier</p> <p>UNIX 的 odbc.ini 字段： CLIENT_LOCALE</p> <p>UNIX 的缺省值： <b>en_us.8859-1</b></p> <p>Windows 的缺省值： <b>en_us.1252</b></p> <p><b>重要：</b> GBase 8s ODBC Driver 忽略在操作系统环境中和在 Setnet32 中的 CLIENT_LOCALE 环境变量的设置。要更改客户机语言环境，必须使用此 GLS 选项。</p>
数据库语言环境	<p>描述： 在其中创建了数据库的语言环境和代码集</p> <p>格式： locale.codeset@modifier</p> <p>UNIX 的 odbc.ini 字段： DB_LOCALE</p> <p>UNIX 的缺省值： <b>en_us.8859-1</b></p> <p>Windows 的缺省值： <b>en_us.1252</b></p> <p><b>重要：</b> GBase 8s ODBC Driver 忽略操作系统环境中在 Setnet32 中的 DB_LOCALE 环境变量的设置。要更改数据库环境变量，则必须使用此 GLS 选项。</p>
事务库	<p>描述： 指定代码集转换</p> <p>格式： 至库文件的路径。事务 DLL 必须遵循事务库的 ODBC 标准。要获取更多信息，请参阅 <i>GBase 8s ODBC Driver 程序员指南</i>。</p> <p>UNIX 的 odbc.ini 字段： TRANSLATIONDLL</p> <p>UNIX 的缺省值： \$GBS_HOME/lib/esql/igo4a304.xx 在此，xx 是共享库的特定于平台的扩展</p> <p>Windows 的缺省值： igo4n304.dll</p>
事务选项	<p>描述： 非 GBase 8s 事务库的选项</p> <p>格式： 由供应商确定</p> <p>UNIX 的 odbc.ini 字段 TRANSLATION_OPTION</p> <p>Windows 的缺省值： 由供应商确定</p> <p><b>限制：</b> 请不要为 GBase 8s 事务库设置此值。GBase 8s 事务库基于客户机语言</p>

GLS 选项	描述
	环境和数据库语言环境值来确定事务选项。
VMB 字符	<p>描述：</p> <p>对于代码集转换的数据，当 <i>rgbValue</i>（输出区）不够大时，改变指定如何设置 <i>pcbValue</i> 的多字节字符长度报告选项。可能的值为：</p> <p><b>Estimate</b></p> <p>GBase 8s ODBC Driver 对返回数据所需的存储空间做最坏估算。</p> <p><b>Exact</b></p> <p>GBase 8s ODBC Driver将代码集转换的数据写至磁盘，直到转换所有数据为止。由于此选项可能降低性能，因此建议您不要使用此选项，除非应用程序不适用于 <b>Estimate</b>。</p> <p>对于数据库或客户机语言环境，当您使用多字节代码集（其中的字符从 1 至 4 字节长度不等）时，数据库语言环境中的字符串或简单大对象（TEXT）的长度不指示将它转换为客户机语言环境之后的字符串长度。</p> <p>UNIX 的可能值：</p> <p>0 = <b>Estimate</b></p> <p>1 = <b>Exact</b></p> <p>Windows 的可能值：</p> <p><b>Estimate</b></p> <p><b>Exact</b></p> <p>UNIX 的 odbc.ini 字段：</p> <p>VMBCHARLENEXACT</p> <p>UNIX 的缺省值：</p> <p><b>Estimate</b></p> <p>Windows 的缺省值：</p> <p><b>Estimate</b></p>

要获取关于 GLS 和语言环境的更多信息，请参阅《GBase 8s GLS 用户指南》。

## 2.7 X/Open 标准接口

除了标准 ODBC 函数之外，GBase 8s ODBC Driver 还支持附加的函数。

GBase 8s ODBC Driver 支持下列函数

**fninfx\_xa\_switch**

获得 GBase Enterprise Records 定义的 xa\_switch 结构的函数

**IFMX\_SQLGetXaHenv**

取得与 XA Connection 相关联的环境句柄的函数

**IFMX\_SQLGetXaHdbc**

取得与 XA Connection 相关联的数据库句柄的函数

**xa\_open**

采用 `xa_info` 参数的函数。GBase 8s ODBC Driver 使用此 `xa_info` 来建立 XA connection

`xa_info` 的格式如下：

```
<applicationtoken>|<DSN name>
```

应用程序令牌是应用程序为每一 `xa_open` 请求生成的唯一编号。它必须使用同一应用程序令牌作为 `IFMX_SQLGetXaHenv` 和 `IFMX_SQLGetXaHdbc` 的参数，来获得相关联的环境句柄和数据库句柄。

## 2.8 外部认证

对于 GBase 8s，您可通过 GBase 8s ODBC Driver 来实现外部认证。

有两个外部认证模块可用于 GBase 8s ODBC Driver，Pluggable Authentication Module（PAM）适用于 UNIX™ 和 Linux™ 服务器。在 Microsoft™ Windows™ 操作系统上支持 LDAP Authentication。

### 2.8.1 UNIX 和 Linux 上的 Pluggable Authentication Module（PAM）

在支持 PAM 的 UNIX™ 和 Linux™ 操作系统上，可随同 GBase 8s ODBC Driver 使用 Pluggable Authentication Module（PAM）。

PAM 使得系统管理员能够为不同的应用程序实现不同的认证机制。例如，像 UNIX 登录程序这样的系统的需求，可能与访问数据库敏感信息的应用程序不同。PAM 允许在单个机器中有多个这样的场景，因为认证服务是附加在应用程序级的。

### 2.8.2 Windows 上的 LDAP Authentication

在 Windows™ 操作系统上，可随同 GBase 8s ODBC Driver 使用 LDAP Authentication。LDAP Authentication 类似于 Pluggable Authentication Module。

当您想要使用 LDAP 服务器来认证系统用户时，请使用 LDAP Authentication 支持模块。该模块包含您可为特定的 LDAP Authentication 支持模块修改的源代码。要获取关于安装和定制 LDAP Authentication 支持模块的信息，请参阅《GBase 8s 安全指南》。

### 2.8.3 带有认证的 SQLSetConnectAttr() 函数

使用 `SQLSetConnectAttr()` 函数来指定服务器使用的回调函数。

`SQLSetConnectAttr()` 也用于指定回调函数使用的参数。参数属性按照为驱动程序指定的方式传回回调函数。

下列属性是特定于 GBase 8s 的对 ODBC 标准的扩展：

参数	类型	描述
SQL_INFX_ATTR_PAM_FUNCTION	void *	指向回调函数的指针。
SQL_INFX_ATTR_PAM_RESPONSE_BUF	void *	指向包含对认证质询的响应的缓冲区的通用指针。
SQL_INFX_ATTR_PAM_RESPONSE_LEN	int	响应缓冲区的长度，以字节计。
SQL_INFX_ATTR_PAM_RESPONSE_LEN_PTR	int *	存储响应中的字节数的地址。
SQL_INFX_ATTR_PAM_CHALLENGE_BUF	void *	指向包含认证质询的缓冲区的通用指针。驱动程序将从服务器收到的任何质询都存储在此缓冲区内。如果该缓冲区未大到足以包含该质询，则截断质询。通过将缓冲区长度与质询中的字节数相对比，回调函数可检测到此质询。由应用程序开发人员负责检测此情况，并正确地处理它。
SQL_INFX_ATTR_PAM_CHALLENGE_BUF_LEN	int	质询缓冲区的长度，以字节计。
SQL_INFX_ATTR_PAM_CHALLENGE_LEN_PTR	int *	存储质询中字节数的地址。

质询和响应缓冲区指针可为空。如果认证服务器要求存储缓冲区中的信息，则由于认证失败导致连接失败。无论连接是否成功，都会返回质询长度信息。如果消息类型不需要响应，则响应缓冲区可能为空（缺省的），或它可能包含空字符串。

可在任何时刻，以任何顺序来设置前面表中的属性。然而，它们仅对以驱动程序的连接函数之一的后续调用建立的连接有效。

通过使用下列连接属性之一，使用 `SQLSetConnectAttr()` API 设置隔离级别：

- `SQL_TXN_READ_UNCOMMITTED` = Read Uncommitted
- `SQL_TXN_READ_COMMITTED` = Read Committed
- `SQL_TXN_SERIALIZABLE` = Serializable
- `SQL_TXN_REPEATABLE_READ` = Repeatable Read
- `SQL_TXN_LAST_COMMITTED` = Last Committed
- `SQL_TXN_TRANSACTION` = Transaction

如果随同 `SQLSetConnectAttr()` API 使用 `SQL_TXN_LAST_COMMITTED` 或 `SQL_TXN_TRANSACTION` 属性，则您的应用程序必须直接链接至 GBase 8s ODBC Driver，而不连接至 ODBC Driver Manager。然而，如果在 `odbc.ini` 文件或 Data Source Administrator 中指定该属性，则应用程序可以使用 ODBC Driver Manager 连接。

如果使用 `SQL_TXN_TRANSACTION` 属性，则将在 DTC 应用程序中设置的隔离级别传播至服务器。仅应在 Windows™ DTC 应用程序中使用此选项。



ODBC 驱动程序的默认行为是从 VARCHAR 列结果的结尾除去空字符。要保留结尾的空白，请设置 SQL\_INFX\_ATTR\_LEAVE\_TRAILING\_SPACES 属性：

```
SQLSetConnectAttr( hdbc, SQL_INFX_ATTR_LEAVE_TRAILING_SPACES,  
(SQLPOINTER)SQL_TRUE, SQL_IS_INTEGER );
```

要除去结尾的空白，请将 SQL\_TRUE 更改为 SQL\_FALSE。

将该行为限定至连接。

### 连接池和认证

在 ODBC 中，驱动程序管理器控制连接池。

当使用认证时，应用程序编程人员必须意识到连接池的影响。驱动程序管理器不控制其连接何时置于池中，或者何时从池中拉出。如果应用程序不了解用户就连接或断开连接，则连接池的性能优势得以保持，用户不会收到任何意外的认证质询。如果应用程序未使用户意识到他们正在重建连接，则仍没有认证问题，因为驱动程序管理器与服务器之间的连接从未关闭。

### 连接函数

可随同认证模块使用任何建立连接的 ODBC 函数，SQLConnect()、SQLDriverConnect() 或 SQLBrowseConnect()。

当使用这些函数时，请考虑下列事项：

#### SQLConnect() 函数

SQLConnect() 函数的 DriverCompletion 参数可采用下列值

- SQL\_DRIVER\_PROMPT
- SQL\_DRIVER\_COMPLETE
- SQL\_DRIVER\_COMPLETE\_REQUIRED
- SQL\_DRIVER\_NOPROMPT

如果预料到有认证挑战，则推荐您使用 SQL\_DRIVER\_NOPROMPT。使用其他值可能导致用户面对认证信息的多个请求。

#### SQLBrowseConnect() 函数

设计 SQLBrowseConnect() 函数旨在反复使用，驱动程序为应用程序提供关于如何完成连接字符串的指导，且应用程序提示用户所需的值。这可导致用户在连接字符串完成与认证之间面对多个提示。

此外，作为连接字符串完成进程的一部分，驱动程序通常向应用程序提供数据库的选择。然而，在用户通过身份验证之后，驱动程序才能为数据库的列表查询服务器。根据应用程序逻辑，它是在原始的连接字符串中提供数据库名称，还是打算从认证服务器接受质询，当服务器使用认证时，可能无法使用 SQLBrowseConnect()。



### 第三方应用程序或中间代码

当使用认证时，应由应用程序处理来自认证服务器的任何质询。

要处理质询，应用程序编程人员必须能够向驱动程序注册回调函数。由于在 ODBC 标准中未定义用来完成此任务的属性，因此，使用的属性为 GBase 8s 扩展。

以 Microsoft<sup>™</sup> 的 ADO 层编写的许多应用程序从开发人员抽取 ODBC 调用。大多数 Visual Basic 应用程序都用 ADO 层编写。这些应用程序和第三方应用程序通常不知道 GBase 8s 扩展，且不能处理身份验证质询。

Windows<sup>™</sup> 上的 ODBC Data Source Administrator 也属于第三方应用程序这一类。当配置 UNIX<sup>™</sup> 数据源时，不是所有特性都可用。例如，如果收到质询，则 Apply and Test Connection 按钮和 User Server Database Locale 切换不起作用，因为那些特性需要连接到服务器的能力。

## 2.9 绕过 ODBC 解析

通过使用几个选项，可绕过 ODBC 解析。

有时，您可能想要通过绕过 ODBC 解析来提升性能。如果存在下列情况，请不要绕过 ODBC 解析：

- 打算在查询中使用 ODBC 转义序列。
- 在运行 SQL 查询之后，打算调用任何目录函数（例如，SQLColumns、SQLProcedureColumns 或 SQLTables）

可以使用下列方式来绕过 ODBC 解析：

- 在连接字符串中将 SKIPPARSING 设置为 1。在 SQLDriverConnect 调用中使用该连接字符串。例如：

```
connString="DB=xxx;UID=xxx;....;SKIPPARSING=1;"
```

- 在 SQLSetConnectAttr 调用中包括 SQL\_INFX\_ATTR\_SKIP\_PARSING，例如：

```
SQLSetConnectAttr(hdbc, SQL_INFX_ATTR_SKIP_PARSING,  
(SQLPOINTER)SQL_TRUE, SQL_IS_USMALLINT);
```

请在连接完成之后使用此调用。要恢复 ODBC 解析，请将 SQL\_TRUE 更改为 SQL\_FALSE。在连接级启用此值之后，以该连接分配的所有语句句柄都继承此属性。

- 在 SQLSetStmtAttr 调用中，包括 SQL\_TRUE。要恢复 ODBC 解析，请将 SQL\_TRUE 更改为 SQL\_FALSE。

```
SQLSetStmtAttr(hstmt, SQL_INFX_ATTR_SKIP_PARSING,  
(SQLPOINTER)SQL_TRUE, SQL_IS_USMALLINT);
```

- 在 UNIX<sup>™</sup> 系统上，在 odbc.ini 中设置 SKIPPARSING=1。要恢复 ODBC 解析，请将该值重置为 SKIPPARSING=0。

绕过 ODBC 解析的优先顺序如下：

- 如果在 `odbc.ini` 文件（在 UNIX 系统上）以及具有 `SQLDriverConnect`、`SQLSetConnectAttr` 或 `SQLSetStmtAttr` API 的应用程序中绕过或重置 ODBC 解析，而且也在带中，则 API 设置优先。
- 如果使用 `SQLDriverConnect` API 以及 `SQLSetConnectAttr` 或 `SQLSetStmtAttr` API 在应用程序中绕过或重置 ODBC 解析，则后者优先。

## 2.10 SQLGetDiagRecW 的字符中的 BufferLength

`SQLGetDiagRecW` API 在输出缓冲区中返回诊断信息，在此，`BufferLength` 参数是分配的缓冲区的长度。

`BufferLength` 的缺省值是分配的字节数。将

`SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW` 属性设置为 `TRUE` 之后，处理 `BufferLength` 为特定的字符数。作为 Widechar API，一个字符 = `sizeof(SQLWCHAR)` 字节。

以下列方式设置该属性：

- `SQLSetEnvAttr (henv, SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW, (SQLPOINTER)SQL_TRUE, SQL_IS_INTEGER);`
- `SQLSetConnectAttr (hdbc, SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW, (SQLPOINTER)SQL_TRUE, SQL_IS_INTEGER);`
- `SQLSetStmtAttr (hstmt, SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW, (SQLPOINTER)SQL_TRUE, SQL_IS_INTEGER);`
- 在连接字符串中设置 `LENGTHINCHARFORDIAGRECW=1`。
- 在 UNIX<sup>™</sup> 系统上，在 `odbc.ini` 中设置 `LENGTHINCHARFORDIAGRECW=1`

设置 `SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW` 的优先顺序为：

- `SQLSetEnvAttr` 属性的设置反映 `henv`、`hdbc` 和 `hstmt` 句柄。
- 重置 `hdbc` 和 `hstmt` 句柄，通过
  - 设置 `SQLSetConnectAttr`
  - 在连接字符串中传递属性
  - 在 DSN 中启用 `Length in Chars for SQLGetDiagRecW` 选项
- 如果以前面提及的方法设置或未设置 `hstmt` 句柄，则设置 `SQLSetStmtAttr` 会重置它。

## 2.11 SQLGetDiagRec 中的 GBase 8s 和 ISAM 错误描述

SQLGetDiagRec API 在输出缓冲区中返回诊断信息,在此,错误描述特定于 GBase 8s 错误消息的。

当 GBase 8s 服务器遇到错误时,它返回 GBase 8s 错误代码以及相关联的错误描述。有一个附加的错误代码,ISAM 错误代码,其提供理解导致该 GBase 8s 错误代码的环境的必要信息。

如果未为 SQLSetConnectAttr API 设置属性,则 SQLGetDiagRec API 返回 GBase 8s 错误消息。

如果为 SQLGetDiagField API 设置了 SQL\_DIAG\_ISAM\_ERROR 属性,则 SQLGetDiagField API 返回 ISAM 错误代码。

如果为 SQLSetConnectAttr API 设置了 SQL\_INFX\_ATTR\_IDSISAMERRMSG 属性,则 SQLGetDiagRec API 同时返回 GBase 8s 错误消息和 ISAM 错误消息。

以下列方式设置 SQL\_INFX\_ATTR\_IDSISAMERRMSG 属性:

```
SQLSetConnectAttr(hdbc, SQL_INFX_ATTR_IDSISAMERRMSG,  
(SQLPOINTER)SQL_TRUE, SQL_IS_INTEGER);
```

## 2.12 提高单线程应用程序的性能

可以通过使用 SINGLETHREADED 连接参数,提升单线程应用程序的性能。缺省情况下,该值为 off。

请不要在 XA/MSDTC 环境中使用此参数。可在连接字符串中设置 SINGLETHREADED 连接参数,如下列示例所示:

```
DSN=xxx;Uid=xxx;Pwd=xxx;SINGLETHREADED=1;"
```

## 2.13 部分支持的和不支持的 ODBC 特性

GBase 8s ODBC Driver 支持几个 ODBC 特性的部分实现。

这些 ODBC 特性为

- 事务处理
- ODBC 游标
- ODBC 书签
- **SQLBulkOperations**

### 2.13.1 事务处理

事务隔离级别和事务模式的 GBase 8s ODBC Driver 实现与这些特性的 Microsoft™ ODBC 实现略有不同。

下列主题描述 GBase 8s ODBC Driver 中事务隔离级别和事务模式的实现。

事务隔离级别

GBase 8s ODBC Driver 支持 GBase 8s 数据库服务器的三个事务隔离级别。

对于 GBase 8s 数据库服务器，下表罗列 GBase 8s ODBC Driver 支持的事务隔离级别。

数据库服务器	事务隔离级别
GBase 8s	<ul style="list-style-type: none"><li>• SQL_TXN_READ_COMMITTED</li><li>• SQL_TXN_READ_UNCOMMITTED</li><li>• SQL_TXN_SERIALIZABLE</li></ul>

缺省的事务隔离级别为 SQL\_TXN\_READ\_COMMITTED。要更改事务隔离级别，请以 SQL\_TXN\_ISOLATION 的 *fOption* 值来调用 SQLSetConnectOption()。

要获取关于事务隔离级别的更多信息，请参阅《GBase 8s ODBC Driver 程序员指南》中的 SQL\_DEFAULT\_TXN\_ISOLATION 和 SQL\_TXN\_ISOLATION\_OPTION 描述。

更改事务模式

您可将事务模式从它缺省的 auto-commit 更改为 manual commit。

要将事务模式更改为 manual commit，请：

1. 为数据库服务器启用事务日志记录。  
要获取关于事务日志记录的信息，请参阅 *GBase 8s 管理员指南*。
2. 调用 SQLSetConnectOption()，将 SQL\_AUTOCOMMIT 设置为 SQL\_AUTOCOMMIT\_OFF。

2. 13. 2 ODBC 游标

GBase 8s ODBC Driver 支持 static 和 forward 游标，但不支持 dynamic 和 keyset-driven 游标。

2. 13. 3 ODBC 书签

书签是标识一行数据的值。

GBase 8s ODBC Driver 以 SQLFetchScroll 和 SQLExtendedFetch 支持书签，不以 SQLBulkOperations 支持它们。GBase 8s ODBC Driver 支持书签到下列程度：

- 仅使用变长书签。
- 为书签列将 SQL\_DESC\_OCTET\_LENGTH 设置为 4。
- 书签是行集内包含行编号的一个整数，从 1 开始。

- 只有游标保持打开，书签才能保持。
- 完全支持 **SQLFetchScroll**，使用 **SQL\_FETCH\_BOOKMARK** 来访问 **orientation** 参数。
- **SQLBulkOperations** 不更新 **SQL\_ADD** 的书签列。

#### 2. 13. 4 SQLBulkOperations

GBase 8s ODBC Driver 仅支持 **SQLBulkOperations** 的 **SQL\_ADD** 参数。

#### 2. 13. 5 SQLDescribeParam

**SQLDescribeParam** 是一个 ODBC API，它返回查询参数的元数据。

在较早版本的 GBase 8s ODBC Driver 中，如果调用 API 来取得关于嵌入在另一例程内的表达式值或参数的信息，**SQLDescribeParam** API 返回 **SQL\_UNKNOWN**。对于 **BOOLEAN**、**LVARCHAR**，或由其他 UDR 中下列表达式返回的内置的非 opaque GBase 8s 数据类型，此限制不再适用：

- 二进制算术表达式
  - 加 (+)
  - 减 (-)
  - 乘 (\*)
  - 除 (/)
- 关系运算符表达式
  - 小于 (<)
  - 小于或等于 (<=)
  - 等于 (=、==)
  - 大于或等于 (>=)
  - 大于 (>)
  - 不等于 (<>、!=)
- 下列字符串运算
  - 串联 (||)
  - MATCHES
  - LIKE
- BETWEEN ... AND 条件表达式

例如，如果列 **tab1.c1** 是 **INT** 数据类型，则 **SQLDescribeParam()** 为下列查询的输入主变量返回类型 **int**：

```
select c1, c2 from tab1 where ABS(c1) > ?;
```

来自表达式另一侧的 UDR 可以是列表表达式或内置的例程，但它不可以是用户定义的例程。在较早的版本中，在下列情况下，**SQLDescribeParam** API 为嵌入在另一过程中的表达式值和参数返回 **SQL\_UNKNOWN**：

- 表达式的另一侧的值为用户定义的例程。
- 同一表达式的另一运算对象是用户定义的例程。

- 表达式的任何运算对象的数据类型不是 BOOLEAN、LVARCHAR 或内置的非 opaque 数据类型。

## 2.13.6 不支持的 Microsoft ODBC 驱动程序特性

GBase 8s ODBC Driver 不支持某些 Microsoft™ ODBC 驱动程序特性的实现。

不支持的 Microsoft ODBC 驱动程序特性是：

- 异步通讯模式
- 并发检查
  - SQL\_CA2\_OPT\_ROWVER\_CONCURRENCY
  - SQL\_CA2\_OPT\_VALUES\_CONCURRENCY
- CONVERT 标量函数
- 游标模拟特性：
  - SQL\_CA2\_CRC\_APPROXIMATE
  - SQL\_CA2\_CRC\_EXACT
  - SQL\_CA2\_SIMULATE\_NON\_UNIQUE
  - SQL\_CA2\_SIMULATE\_TRY\_UNIQUE
  - SQL\_CA2\_SIMULATES\_UNIQUE
- 动态游标属性
- Installer DLL

## 3 配置数据源

这些主题说明如何为 GBase 8s ODBC Driver 在 UNIX™ 和 Windows™ 上配置数据源（DSN）。

安装驱动程序之后，在可连接到它之前，您必须配置 DSN。

### 3.1 在 UNIX 上配置 DSN

配置文件提供驱动程序用来连接至 DSN 的信息，诸如驱动程序属性。

本部分提供关于 UNIX™ 上的驱动程序规范和 DSN 规范的信息，并描述下列 DSN 配置文件：

- sqlhosts
- odbcinst.ini
- odbc.ini

要修改这些文件，请使用文本编辑器。该部分还提供驱动程序和 DSN 规范的示例。

如果您正在启用单点登录（SSO），请参阅《GBase 8s 安全指南》中的“为 SSO 配置 ESQ/L/C 和 ODBC Driver”的内容。

### 3.1.1 odbcinst.ini 文件

对于计算机上所有安装的驱动程序，odbcinst.ini 文件都有条目。

安装了的 ODBC 驱动程序使用 odbcinst.ini 样例文件，其位于 \$GBS\_HOME/etc/odbcinst.ini 中。要创建您的 odbcinst.ini 文件，请将 odbcinst.ini 样例文件复制至您的目录作为 \$HOME/.odbcinst.ini（请注意文件名称开始处添加的点）。当您安装新的驱动程序或驱动程序的新版本时，请更新此文件。下表描述 \$HOME/.odbcinst.ini 文件中的部分项。

部分	描述	状态
ODBC 驱动程序	所有安装的 ODBC 驱动程序的名称列表	可选的
ODBC 驱动程序规范	驱动程序属性和值的列表	可选的

#### ODBC 驱动程序

使用示例来获得关于 ODBC 驱动程序的信息。

下列示例说明关于驱动程序的信息：

```
[ODBC Drivers]
    driver_name1=Installed
    driver_name2=Installed
    :
```

下列示例说明关于安装了的驱动程序的信息：

```
[ODBC Drivers]
    GBase 8s ODBC DRIVER=Installed
```

#### 驱动程序规范

每个安装了的驱动程序在驱动程序名称下方描述其属性。

下列示例说明驱动程序规范格式：

```
[driver name1]
    Driver=driver_library_path
    Setup=setup/driver_library_path
    APILevel=api_level_supported
    ConnectFunctions=connectfunctions
    DriverODBCVer=odbc_version
    FileUsage=file_usage
    SQLLevel=sql_level
    :
```

下列示例说明关于驱动程序规范的信息：

```
[GBase 8s ODBC DRIVER]
```

```
Driver=/vobs/tristarm/odbc/iclis09b.so
Setup=/vobs/tristarm/odbc/iclis09b.so
APILevel=1
ConnectFunctions=YYY
DriverODBCVer=03.50
FileUsage=0
SQLLevel=1
```

下表描述驱动程序规范部分中的关键字。

关键字	描述	状态
API Level	ODBC 接口符合驱动程序支持的级别 <ul style="list-style-type: none"> <li>0= 无</li> <li>1= 支持级别 1</li> <li>2= 支持级别 2</li> </ul>	必需的
ConnectFunctions	指示驱动程序是否支持 SQLConnect、SQLDriverConnect 和 SQLBrowseConnect 的三字符字符串	必需的
DriverODBCVer	带有驱动程序支持的 ODBC 版本的字符串	必需的
Driver	驱动程序库路径	必需的
FileUsage	指示基于文件的驱动程序如何直接处理 DSN 中文件的数值	必需的
Setup	设置库	必需的
SQLLevel	指示驱动程序支持的 SQL-92 语法的数值	必需的

### 3.1.2 odbc.ini 文件

odbc.ini 文件是样例数据源配置信息文件。

要了解 odbc.ini 文件的位置，请参阅发布说明。要创建此文件，请将 odbc.ini 复制至您的 home 目录作为 \$HOME/.odbc.ini（请注意文件名称开始处添加的点）。您的应用程序连接至的每个 DSN 都必须在此文件中有一个条目。下表描述 \$HOME/.odbc.ini 中的部分。

部分	描述	状态
ODBC 数据源	此部分罗列 DSN 并将它们与驱动程序的名称相关联。如果您使用来自第三方供应商的 ODBC 驱动程序，则仅需要提供此部分。	必需的
数据源规范	“ODBC 数据源”部分罗列的每一 DSN 都有一个描述该 DSN 的“数据源规范”部分。	必需的
ODBC	此部分罗列 ODBC 跟踪选项。	可选的

请遵循下列规则，以在 UNIX™ 系统上的 odbc.ini 文件中包括注释：



- 在第一行的第一个位置，以分号 (;) 或数字符号 (#) 来开始注释。
- 如果注释包括多行，则可以空格或标签字符 (\t) 来开始下面的注释行。
- 可在注释中包括空行。

## ODBC 数据源

“ODBC 数据源”部分中的每一条目都罗列 DSN 和驱动程序名称。

*data\_source\_name* 值是您选择的任何名称。它就像包含关于 DSN 的所有相关链接信息的信封。

下列示例说明 ODBC data-source 格式：

```
[ODBC Data Sources]
    data_source_name=GBase 8s ODBC DRIVER
    :
```

下列示例定义两个名为 EmpInfo 和 CustInfo 的 DSN：

```
[ODBC Data Sources]
    EmpInfo=GBase 8s ODBC DRIVER
    CustInfo=GBase 8s ODBC DRIVER
```

## 数据源规范

数据源部分中的每一 DSN 都有一个数据源规范部分。

下列示例说明数据源规范格式：

```
[data_source_name]
    Driver=driver_path
    Description=data_source_description
    Database=database_name
    LogonID=user_id
    pwd=user_password
    Server=database_server
    CLIENT_LOCALE=application_locale
    DB_LOCALE=database_locale
    TRANSLATIONDLL=translation_path
    CURSORBEHAVIOR=cursor_behavior
    DefaultUDTFetchType=default_UDT_Fetch_type
    ENABLESCROLLABLECURSORS=enable_scroll_cursors
    ENABLEINSERTCURSORS=enable_insert_cursors
    OPTIMIZEAUTOCOMMIT=optimize_auto_commit
    NEEDODBCTYPESONLY=need_odbc_types_only
    OPTOFC=open_fetch_close_optimization
    REPORTKEYSETCURSORS=report_keyset_cursors
    FETCHBUFFERSIZE=fetchbuffer_size
    DESCRIBEDECIMALFLOATPOINT=describe_decimal_as_float
    USESERVERDBLOCALE=use_server_dblocale
```

```

DONOTUSELVARCHAR=do_not_use_lvvarchar
REPORTCHARCOLASWIDECHARCOL=char_col_as_widechar_col
[ODBC]
UNICODE=unicode_type

```

```

LENGTHINCHARFORDIAGRECW=bufferlength_as_number_of_characters
LEAVE_TRAILING_SPACES=leave_trailing_spaces

```

下表描述数据源规范部分中的关键字，以及它们在每一部分中出现的顺序。

关键字	描述	状态
<b>data_source_name</b>	在“数据源”部分指定的数据源	必需的
<b>Driver</b>	驱动程序的路径  将此值设置为驱动程序库的完整路径名称。要获取关于库目录和文件名称的更多信息，请参阅发版说明。	必需的
<b>Description</b>	DSN 的描述  为单个用户，或为系统用户配置。	可选的
<b>Database</b>	在缺省情况下 DSN 连接至的数据库	必需的
<b>LogonID</b>	访问 DSN 的用户标识或账户名称	可选的
<b>pwd</b>	访问 DSN 的口令	可选的
<b>Server</b>	<i>database_name</i> 所在的 GBase 8s 数据库服务器	必需的
<b>CLIENT_LOCALE</b> （仅限于 GLS）	客户机语言环境。缺省值： <b>en_us.8859-1</b>	可选的
<b>DB_LOCALE</b> （仅限于 GLS）	数据库语言环境。缺省值： <b>en_us.8859-1</b>	可选的
<b>TRANSLATIONDLL</b> （仅限于 GLS）	执行代码集转换的 DLL；缺省值： \$GBS_HOME/lib/esql/ig04a304. <i>xx</i> 在此， <i>xx</i> 表示特定于平台的文件扩展名	可选的
<b>CURSORBEHAVIOR</b>	当调用提交或回滚事务时，游标行为的标志。 可能的值为： <ul style="list-style-type: none"> <li>0= 关闭游标</li> <li>1= 保持游标</li> </ul> 缺省值：0	可选的
<b>DefaultUDTFetchType</b>	缺省的 UDT 访存类型。  缺省值：SQL_C_BINARY  可能的值为： <ul style="list-style-type: none"> <li>SQL_C_BINARY</li> </ul>	可选的

关键字	描述	状态
	<ul style="list-style-type: none"> <li>SQL_C_CHAR</li> </ul>	
ENABLESCROLLABLECURSORS	<p>如果激活此选项，则 GBase 8s ODBC Driver 仅支持 scrollable、static 游标。仅作为连接选项可用：</p> <p>SQL_INFX_ATTR_ENABLE_SCROLL_CURSORS</p> <p>或作为连接属性字符串：</p> <p>EnableScrollableCursors</p> <p>缺省值为：0（禁用的）</p>	可选的
ENABLEINSERTCURSORS	<p>通过缓存与参数数组和插入语句一起使用的插入行，减少发送至服务器和来自服务器的网络消息数。此选项可以提高批量插入操作的性能。同时作为连接和语句选项可用：</p> <p>SQL_INFX_ATTR_ENABLE_INSERT_CURSORS</p> <p>或作为连接属性字符串：</p> <p>EnableInsertCursors</p> <p>缺省值为：0</p>	可选的
OPTIMIZEAUTOCOMMIT	<p>在游标保持打开时，延迟自动提交操作。当应用程序正在使用非 ANSI 日志记录数据库时，此选项可减少数据库通讯。作为连接选项可用：</p> <p>SQL_INFX_ATTR_OPTIMIZE_AUTOCOMMIT</p> <p>或作为连接属性字符串：</p> <p>OptimizeAutoCommit</p> <p>缺省值为：1（禁用的）</p>	可选的
OPTOFC	<p>导致驱动程序将 open、fetch 和 close 游标消息缓冲至服务器。当您使用 SQLPrepare、SQLExecute 和 SQLFetch 语句来访问带有游标的数据时，此选项可去除一个或多个消息循环。仅作为连接选项可用：</p> <p>SQL_INFX_ATTR_OPTOFC</p> <p>或作为连接属性字符串：</p> <p>OPTOFC</p> <p>缺省值为：0（禁用的）</p>	可选的
REPORTKEYSETCURSORS	<p>导致驱动程序（通过SQLGetInfo）来报告，支持 forward-only、static 和</p>	可选的

关键字	描述	状态
	<p>keyset-driver 游标，即使驱动程序仅支持 forward-only 和 static 游标。使用此选项来启用 dynaset 类型函数，诸如 Microsoft™ Visual Basic，其要求驱动程序支持 keyset-driven 游标。</p> <p>也作为连接选项可用：</p> <p>SQL_INFX_ATTR_REPORT_KEYSET_CURSORS</p> <p>或作为连接属性字符串：</p> <p>ReportKeysetCursors</p> <p>缺省值为：0（禁用的）</p>	
FETCHBUFFERSIZE	<p>以字节计的访存缓冲区的大小。</p> <p>作为连接属性字符串可用：</p> <p>FETCHBUFFERSIZE</p> <p>访存缓冲区的最大大小为 2 GB。</p> <p>缺省值为：32767</p>	可选的
DESCRIBEDECIMALFLOATPOINT	<p>描述所有浮点小数列为：</p> <ul style="list-style-type: none"> <li>Float(SQL_REAL) 或</li> <li>Float(SQL_DOUBLE)</li> </ul> <p>浮点小数列是不带有范围创建了的列，例如 DECIMAL(12)。某些诸如 Visual Basic 这样的打包应用程序无法正确地格式化没有固定范围的 Decimal 列。要使用这些应用程序，您必须启用此选项或以固定的范围来重新定义该列。</p> <p>启用此选项的缺点是，SQL_REAL 和 SQL_DOUBLE 为近似数值数据类型，SQL_DECIMAL 是精确的数值数据类型。将精度为 8 或更低的 SQL_DECIMAL 描述为 SQL_REAL。将精度大于 8 的描述为 SQL_DOUBLE。</p> <p>作为连接属性字符串可用：</p> <p>DESCRIBEDECIMALFLOATPOINT</p> <p>缺省值为：0（禁用的）</p>	可选的
USESERVERDBLOCALE	<p>用户服务器数据库语言环境。</p> <p>作为连接属性字符串可用：</p> <p>USESERVERDBLOCALE</p>	可选的

关键字	描述	状态
	缺省值为：0（禁用的）	
DONOTUSELVARCHAR	<p>如果启用，则 SQLGetTypeInfo 不报告 LVARCHAR 作为 SQL_VARCHAR 的支持的类型（DATA_TYPE）。一些应用程序使用 LVARCHAR 而不是 VARCHAR，即使在小于 256 字节的列中。为 LVARCHAR 传输的最小字节数高于 VARCHAR。许多 LVARCHAR 列可导致 rowset 大小超出最大值。</p> <p><b>重要：</b> 仅当 SQL_VARCHAR 列小于 256 字节时，才启用此选项。</p> <p>作为连接属性字符串可用：</p> <p>DONOTUSELVARCHAR</p> <p>缺省值为：0（禁用的）</p>	可选的
REPORTCHARCOLASWIDECHARCOL	<p>导致 SQLDescribeCol 报告字符列作为宽字符列，如下：</p> <ul style="list-style-type: none"> <li>• 报告 SQL_CHAR 作为 SQL_WCHAR</li> <li>• 报告 SQL_VARCHAR 作为 SQL_WVARCHAR</li> <li>• 报告 SQL_LONGVARCHAR 作为 SQL_WLONGVARCHAR</li> </ul> <p>作为连接属性字符串可用：</p> <p>REPORTCHARCOLASWIDECHARCOL</p> <p>缺省值为：0（禁用的）</p>	可选的
UNICODE	<p>指示应用程序使用的 Unicode 的类型。此属性仅适用于 UNIX™ 应用程序，并在 odbc.ini 文件的 ODBC 部分中设置。请考虑下列注意事项：</p> <ul style="list-style-type: none"> <li>• 未链接至 Data Direct ODBC 驱动程序管理器的 UNIX 上的应用程序应设置此为 UCS-4</li> <li>• 版本低于 GBase AIX®5L 上的应用程序应设置此属性为 UCS-2。</li> <li>• 使用 Data Direct 驱动程序管理器的应用程序不需要设置此属性。</li> </ul> <p>缺省值为：UTF-8</p> <p>要获取关于在 ODBC 应用程序中使用 Unicode 的更多信息，请参阅 Unicode。</p>	必需的
LENGTHINCHARFORDIAGRECW	<p>如果启用，则 SQLGetDiagRecW API 处理</p>	

关键字	描述	状态
	BufferLength 参数作为字符数。  缺省值为：FALSE（禁用的）  要获取关于使用 BufferLength 参数的更多信息，请参阅 SQLGetDiagRecW 的字符中的 BufferLength。	
LEAVE_TRAILING_SPACES	如果启用，则驱动程序在 VARCHAR 列结果的结尾保留空字符。 可能的值为： <ul style="list-style-type: none"><li>• 0（除去结尾空格）</li><li>• 1（保留结尾空格）</li></ul> 缺省值为：0	

下列示例展示名为 EmpInfo 的 DSN 的配置：

```
[EmpInfo]
    Driver=/gbasedbt/lib/cli/iclis09b.so
    Description=Demo data source
    Database=odbc_demo
    LogonID=admin
    pwd=tiger
    Server=ifmx_91
    CLIENT_LOCALE=en_us.8859-1
    DB_LOCALE=en_us.8859-1
    TRANSLATIONDLL=/opt/gbasedbt/lib/esql/igo4a304.so
```

下列示例展示名为 GBase 8s 9 的 DSN 的配置：

```
[GBase 8s 9]
    Driver=/work/gbasedbt/lib/cli/iclis09b.so
    Description=GBase 8s 9.x ODBC Driver
    LogonID=user1
    pwd=tigress4
    Database=odbc_demo
    ServerName=my_server
```

如果指定空 LogonID 或 pwd，则发生下列错误：

Insufficient connect information supplied

**提示：**如果本地 GBase 8s 数据库服务器与客户机在的同一计算机上，则您可以 LogonID 和 pwd 的空值来建立至 DSN 的连接。在此情况下，将当前用户视为可信的用户。

服务器与客户机位于同一计算机上，不带有 LogonID 和 pwd 的样例数据源可能像下列示例一样：

```
Driver=/work/gbasedbt/lib/cli/iclis09b.so
Description=GBase 8s 9.x ODBC Driver
LogonID=
pwd=tiger
Database=odbc_demo
ServerName=ifmx_server
```

设置隔离级别（仅限于 UNIX）

通过使用 ISOLATIONLEVEL 和 SQL\_TXN\_LAST\_COMMITTED 关键字，来在 odbc.ini 文件中设置隔离级别。

要在 odbc.ini 文件中指定隔离级别，请使用下列关键字和值：

- ISOLATIONLEVEL = level
- SQL\_TXN\_LAST\_COMMITTED = last committed

在此，level 是从 0 至 5 的数值：

- 0 = 基于数据库类型自动地考虑缺省值
- 1 = Read Uncommitted
- 2 = Read Committed（非 ANSI 数据库的缺省值）
- 3 = Repeatable Read（ANSI 数据库的缺省值）
- 4 = Serializable
- 5 = Last Committed

如果应用程序调用带有 SQL\_ATTR\_TXN\_ISOLATION 属性的 SQLSetConnectAttr，并在连接之前设置该值，然后又在连接字符串中设置 ISOLATIONLEVEL 或 ISOLVL，则连接字符串是要使用的最终值。

在 UNIX™ 平台上不支持 SQL\_TXN\_TRANSACTION 隔离级别。

### 3.1.3 ODBC 部分

odbc.ini 的 ODBC 部分中的值指定 ODBC 跟踪选项。

使用跟踪，您可发现调用产生的日志，还有每一调用的返回代码。在 Windows™ 上，通过 ODBC Data Source Administrator 对话框的 Tracing 标签来设置这些选项。

下表描述 ODBC 部分中的跟踪选项：

表 1. odbc.ini 的 ODBC 部分的跟踪选项

选项	细节
TRACE=1	启用跟踪
TRACE=0	禁用跟踪
TRACEFILE	设置为你想要驱动程序写入调用日志的位置。
TRACEDLL	始终为 idmrs09a.so



下列示例说明 ODBC 部分规范格式：

```
[ODBC]
TRACE=1
TRACEFILE=/WORK/ODBC/ODBC.LOG
TRACEDLL=idmrs09a.so
UNICODE=UCS-4
```

您必须将 `TRACEFILE` 设置为想要驱动程序写入所有调用日志的位置。请记住，`TRACE=1` 意味着启用跟踪。`TRACE=0` 禁用跟踪选项。

### 3.1.4 设置 \$ODBCINI 环境变量

设置 `$ODBCINI` 环境变量来由系统用户提供对 DSN 的访问。

在缺省情况下，GBase 8s ODBC Driver 使用在 `$HOME/.odbc.ini` 文件中的配置信息。如果您想要由系统用户提供对您的 DSN 的访问，请修改 `$ODBCINI` 环境变量中的路径，使其指向另一个包含 `$HOME/.odbc.ini` 文件中的配置信息的配置文件。然后更改配置文件许可，以允许系统用户进行读访问。请不要更改对 `$HOME/.odbc.ini` 文件的权限。

在下列示例中，配置文件名称为 `myodbc.ini`：

```
setenv ODBCINI /work/myodbc.ini
```

### 3.1.5 .netrc 文件

`.netrc` 文件包含通过网络登录至远程数据库服务器的数据。

在客户机计算机初始化连接处的 `home` 目录中，创建 `.netrc` 文件。将用户的 `.netrc` 文件许可设置为拒绝由组和其他用户读访问。

要连接至远程数据库服务器，请在 `.netrc` 文件中为需要自动连接至数据源的 `LogonID` 和 `pwd` 创建条目。要建立至远程数据源的连接，ODBC 驱动程序首先从 `$HOME/.odbc.ini` 文件中的数据源条目读取 `LogonID` 和 `pwd`。如果 `$HOME/.odbc.ini` 文件未指定 `LogonID` 和 `pwd`，则 ODBC 驱动程序搜索 `$HOME/.netrc` 文件。

例如，通过使用带有口令 `mypassword` 的登录名称 `log8in`，要允许自动登录至名为 **ray** 的计算机，您的 `.netrc` 文件要包含下列行：

```
machine ray login log8in password mypassword
```

要获取关于 `.netrc` 文件的信息，请参阅 UNIX™ 资料。

## 3.2 在 Windows 中配置 DSN

在 Windows™ 环境中，GBase 8s ODBC Driver 提供 GUI 来配置 DSN。

要配置 DSN，请：

- 选择一个过程来修改 DSN：
  - 选择“用户 DSN”选项来限制一个用户的访问。
  - 选择“系统 DSN”选项来限制系统用户的访问。
  - 选择“文件 DSN”选项以允许网络上所有用户的访问。
- 键入“DSN 配置”值，来创建 DSN，诸如数据源名称、数据库服务器名称和数据库语言环境。

对于值的描述，请参阅后面的两个表。以它们在每一部分中出现的顺序显示值。还可使用 Microsoft<sup>™</sup> ODBC Version 2.5或后来的版本，来配置 DSN。

**提示：** 要查明有什么 DSN，请点击 关于 标签，并阅读 描述 文本框的内容。

**重要：** 要在 Windows 64 位平台上配置 DSN，您必须使用 32 位 ODBC Data Source Administrator：

C:\WINDOWS\SysWOW64\odbcad32.exe

您必须为 SSO 指定用户和口令或 CSM 设置。必须指定用户和口令。 如果正在启用单点登录（SSO），则附加的步骤位于《GBase 8s 安全指南》中的“为 SSO 配置 ESQ/C 和 ODBC Driver”中。

表 1. 必需的 DSN 值

必需的值	描述
数据源名称	要访问的 DSN  此值为您选择的任意名称。 <b>数据源名称</b> 就像一个包含关于 DSN 的所有相关连接信息的信封一样。
数据库名称	在缺省情况下，DSN 连接到的数据库名称。
主机名称	<b>服务器</b> 所在的计算机
协议	用来与 <b>服务器</b> 通讯的协议  在添加了 DSN 之后，该菜单键显示可用的选项。
服务器名称	<b>服务器</b> 所在的 GBase 8s 数据库服务器
服务	运行在 <b>主机</b> 上的 GBase 8s 数据库服务器进程  请与您的系统管理员或数据库管理员确认该服务名称。

表 2. 可选的 DSN 值

可选的值	描述
客户机语言环境	缺省值：en_us.1252
数据库语言环境	缺省值：en_us.1252
描述	任何信息，诸如版本号和服务
选项	通用信息，诸如口令设置  要获取关于此值的更多信息，请参阅《GBase 8s 管理员指南》中的 sqlhosts 信息。

必需的值	描述
口令	访问 DSN 的口令
事务库	指定代码集转换的“动态链接库”（DLL）；缺省值： \$GBS_HOME\bin\ig04n304.dll
用户 ID	访问 DSN 的用户标识或账户名称
事务选项	非 GBase 8s 事务库的选项  当 <i>rgbValue</i> （输出区域）对于代码集转换的数据不够大时，指定如何设置 <i>pcbValue</i> 的变化的多字符长度报告选项  可能的值： <ul style="list-style-type: none"><li>• 0= 估计的</li><li>• 1= 精确地</li></ul> 缺省值：0
游标行为	当调用提交或回滚事务时，游标行为的标志 可能的值为： <ul style="list-style-type: none"><li>• 0= 关闭游标</li><li>• 1= 保持游标</li></ul> 缺省值：0

在完成这些步骤之后，您将连接至 DSN。

3.2.1 配置新的用户 DSN 或系统 DSN

访问 ODBC Data Source Administrator 对话框，来配置新的用户 DSN 或系统 DSN。

要配置新的用户 DSN 或系统 DSN，请：

- 1. 选择**开始 > 设置 > 控制面板**。
- 2. 双击 **ODBC** 来打开 ODBC Data Source Administrator 对话框。
  - 要配置用户 DSN，请前进至步骤 3。
  - 要配置系统 DSN，请点击系统 DSN 标签，并前进至步骤 3。

所有后续步骤都与配置用户 DSN 或系统 DSN 相同。

- 3. 点击**添加**。  
  
“创建新的数据源”对话框打开。
- 4. 在“创建新的数据源”向导上双击 **GBase 8s ODBC 驱动程序**。  
  
“GBase 8s ODBC Driver 设置”对话框的通用页面打开。
- 5. 在通用页面中键入值，如下列示例所示：
  - “数据源名称”：odbc33int
  - “描述”：file DSN 3.81 on turbo

**限制：** 在此页面上键入值之后，请不要点击确定。如果在键入所有值之前点击确定，则会得到错误消息。

6. 点击**连接**标签来显示连接页面，并键入值，如下列示例所示：
- “服务器名称”：ol\_clipper（或者使用菜单来选择 sqlhosts 注册表上的服务器。如果使用菜单，则 ODBC 应用程序设置“主机名称”、“服务”、“协议”和“选项”值。）
  - “主机名称”：clipper
  - “服务”：turbo
  - “协议”：onsoctcp（或者使用菜单来选择协议）
  - “选项”：csm=(SPWDCSM)
  - “数据库名称”：odbc\_demo（或者使用菜单来找到数据库名称）
  - “用户 ID”：myname
  - “口令”：\*\*\*\*\*

要保存您选择的值并验证 DSN 连接成功，请点击“应用并测试连接”。“ODBC 消息”对话框打开。该框告诉您连接是否成功，如果不成功，则告诉您哪个“连接标签”值不正确。

7. 点击**环境**标签来显示环境页面，并键入值，如下列示例所示：
- “客户机语言环境”：en\_US.CP1252
  - “数据库语言环境”：en\_US.CP1252
  - “使用服务器数据库语言环境”：如果选中该勾选框，则将数据库语言环境设置为服务器语言环境。如果清除该勾选框，则将数据库语言环境设置为缺省的语言环境 en\_US.CP1252。
  - “事务库”：GBS\_HOME\lib\esql\ig04n304.dll
  - “事务选项”：0
  - “游标行为”：0 - 关闭
  - “VMB 字符”：0 - 估计的
  - “访存缓冲区大小”：4096
  - “隔离级别”：0 - 认为是缺省的 Read Committed（非 ANSI 数据库）或 Repeatable Read（ANSI 数据库）

8. 点击**高级**标签来显示高级页面，并点击所有适当的框。

选项	描述
自动提交优化	在游标保持打开时，此选项延迟自动提交操作，且当应用程序正在使用非 ANSI 日志记录的数据库时，此选项减少数据库通讯。此选项仅作为连接选项可用：  SQL_INFX_ATTR_OPTIMIZE_AUTOCOMMIT 或作为连接属性字符串：“OptimizeAutoCommit”  缺省值为：1（启用的）。

选项	描述
“打开-访存-关闭”优化	<p>此选项导致驱动程序将 open、fetch 和 close 游标消息缓冲至服务器。此外，当您使用 SQLPrepare、SQLExecute 和 SQLFetch 语句来访问带游标的数据时，此选项消除一个或多个消息往复。此选项仅作为连接选项可用：</p> <p>SQL_INFX_ATTR_OPTOFC</p> <p>或者作为连接属性字符串：“OPTOFC”</p> <p>缺省值为：0（禁用的）</p>
插入游标	<p>通过缓存与参数数组和插入语句一起使用的插入行，此选项减少发送至服务器或来自服务器的网络消息数。此选项通常可极大地提升批量插入操作的性能，并同时作为连接和语句选项可用：</p> <p>SQL_INFX_ATTR_ENABLE_INSERT_CURSORS.</p> <p>或者作为连接属性字符串：“EnableInsertCursors”</p> <p>缺省值为：0（禁用的）。</p>
可滚动游标	<p>如果激活此选项，则 GBase 8s ODBC Driver Version 2.90 和后来版本仅支持 scrollable、static 游标。此选项仅作为连接选项可用：</p> <p>SQL_INFX_ATTR_ENABLE_SCROLL_CURSORS</p> <p>或者作为连接属性字符串：“EnableScrollableCursors”</p> <p>缺省值为：0（禁用的）。</p>
报告 KeySet 游标	<p>此选项导致驱动程序（通过 SQLGetInfo）来报告它支持 forward-only、static 和 keyset-driven 游标类型，尽管驱动程序仅支持 forward-only 和 static 游标。当设置此选项时，驱动程序启用 dynaset-type 函数，诸如 Microsoft™ Visual Basic 的函数。这些函数需要驱动程序支持 keyset-driven 游标类型。此选项还作为连接属性可用：</p> <p>SQL_INFX_ATTR_REPORT_KEYSET_CURSORS</p> <p>或者作为连接属性字符串：“ReportKeysetCursors”</p> <p>缺省值为：0（禁用的）。</p>
仅报告标准 ODBC 类型	<p>如果激活此特性，则驱动程序会导致 SQLGetTypeInfo 按如下方式映射所有用户定义的类型（UDT）：</p> <p>Blob SQL_LONGVARBINARY</p> <p>Clob SQL_LONGVARBINARY</p> <p>Multiset SQL_C_CHAR/SQL_C_BINARY</p> <p>Set</p>

选项	描述
	<p><b>SQL_C_CHAR/SQL_C_BINARY</b></p> <p>List</p> <p><b>SQL_C_CHAR/SQL_C_BINARY</b></p> <p>Row</p> <p><b>SQL_C_CHAR/SQL_C_BINARY</b></p> <p>驱动程序将 multiset、set、row 和 list 数据类型映射为 <b>SQL_C_CHAR</b> 或 <b>SQL_C_BINARY</b>，这是缺省的 UDT FetchType 到 <b>SQL_C_CHAR</b> 功能。</p> <p>缺省值为：0（禁用的）。</p>
描述小数浮点为 SQL_REAL / SQL_DOUBLE	<p>此选项描述所有浮点小数列为 Float（SQL_REAL 或 SQL_DOUBLE）。浮点小数列是未带有范围创建的列，例如：DECIMAL(12)。诸如 Visual Basic 这样的一些预先打包的应用程序无法恰当地格式化那些没有固定范围的 Decimal 列。要使用这些应用程序，您必须启用此选项，或以固定的范围来重新定义该列。</p> <p>然而，启用此选项有一个缺点，SQL_DECIMAL 是精确的数值数据类型，而 SQL_REAL 和 SQL_DOUBLE 是近似的数值数据类型。将精度为 8 或更小的 SQL_DECIMAL 描述为 SQL_REAL，精度大于 8 的是 SQL_DOUBLE。</p> <p>缺省值为：0（禁用的）。</p>
不使用 LVARCHAR	<p>导致 SQLGetTypeInfo 不报告 LVARCHAR 作为 SQL_VARCHAR 的支持的 DATA_TYPE 类型。</p> <p>诸如 MS Access97 这样的一些应用程序使用 LVARCHAR，而不是 VARCHAR，即使对于长度小于 256 字节的列也是如此。为 LVARCHAR 传输的最小字节数高于 VARCHAR，且许多 LVARCHAR 列可能导致 rowset 大小超出最大值。仅当 SQL_VARCHAR 列的长度小于 256 字节时，才启用此选项。</p> <p>缺省值为：0（禁用的）。</p>
报告 CHAR 列作为宽 CHAR 列	<p>导致 SQLDescribeCol 报告 char 列为宽 char 列。报告 SQL_CHAR 列为 SQL_WCHAR，SQL_VARCHAR 为 SQL_WVARCHAR，SQL_LONGVARCHAR 列为 SQL_WLONGVARCHAR。</p> <p>缺省值为：0（禁用的）。</p>
SQLGetDiagRecW 的 Char 长度	<p>如果启用，则 SQLGetDiagRecW API 将 BufferLength Parameter 处理为字符数。</p> <p>缺省值为：FALSE（禁用的）。</p>
保留结尾的空格	<p>如果启用，则驱动程序保留 VARCHAR 列结果结尾处的空字符。</p> <p>缺省值为：0（禁用的）。</p>

9. 要检查至数据库服务器的连接，请点击测试连接。
10. 点击确定，以返回至“ODBC Data Source Administrator”对话框，并在恰当的文件中更新 DSN 信息。

当您的应用程序连接至此 DSN 时，您键入了的值是 DSN 连接的缺省条目。

### 3.2.2 移除 DSN

访问“ODBC Data Source Administrator”对话框来移除 DSN。

要移除 DSN，请：

1. 遵循来自 配置新的用户 DSN 或系统 DSN 的步骤 1 和 2。
2. 在“ODBC Data Source Administrator”对话框中点击移除。  
“32 位 ODBC Administrator”对话框打开。
3. 点击是，来移除 DSN 并返回至“ODBC Data Source Administrator”对话框。

### 3.2.3 重新配置现有的 DSN

访问“ODBC Data Source Administrator”对话框，来重新配置现有的用户 DSN。

要重新配置现有的 DSN，请：

1. 遵循来自 配置新的用户 DSN 或系统 DSN 的步骤 1 和 2。
2. 点击配置来显示“GBase 8s ODBC Driver 设置”对话框。  
在对应的文本框中键入新的配置值，并点击确定以返回“ODBC Data Source Administrator”对话框。

在完成这些步骤之后，您会连接至该 DSN。

### 3.2.4 配置文件 DSN

访问“ODBC Data Source Administrator”对话框，来配置文件 DSN。

要配置文件 DSN，请：

1. 选择开始 > 设置 > 控制面板。
2. 双击“ODBC”图标来打开“ODBC Data Source Administrator”对话框。
3. 点击文件 DSN 标签，来显示文件 DSN 页面。  
选择“文件 DSN”选项，以允许网络上的所有用户访问该 DSN。
4. 点击添加。  
“创建新的数据源”助手打开。
5. 从驱动程序列表选择 GBase 8s ODBC Driver，并点击下一步来显示“Create New Data Source Setup”助手，其包含文件数据源文本框。
6. 如果您知道数据源文件的名称，则将该名称键入至文本框内，点击下一步来显示完成的“Create New Data Source”助手，并前进至步骤 9



如果不知道文件的名称，则点击浏览来显示“另存为”对话框，并输入值，如下列示例所示：

- “文件名称”：File\_DSN
- “另存为类型”：ODBC File Data Sources

选择一个文件名称，或在 File\_name 文本框中输入文件名称。

- 7. 点击保存来显示“Create New Data Source”助手，其显示关于数据源名称的信息。
- 8. 点击下一步来显示完成的“Create New Data Source”助手。
- 9. 点击完成来显示“GBase 8s Connect”对话框。

要了解这些值的描述，请参阅 表 1 和 表 2。要了解高级标签值，请参阅 配置新的用户 DSN 或系统 DSN。

- 10. 点击确定来保存这些值，并显示“ODBC Data Source Administrator”对话框。

在文本框中显示您在步骤 6 中选择或输入的数据文件的名称。

在添加或更改 DSN 配置信息之后，驱动程序更新恰当的 Windows™ 注册表，来反映指定的值。要与其他 GBase 8s 连接性产品相兼容，驱动程序将 DSN 配置信息存储在 Windows 注册表中。

3.2.5 创建调用驱动程序的日志

访问跟踪页面，来创建调用驱动程序的日志。

要创建调用驱动程序的日志，请：

- 1. 点击跟踪标签，来显示跟踪页面。
- 2. 选择现在开始跟踪来开启跟踪。
- 3. 要输入现有的日志文件，请点击浏览来显示“选择 ODBC 日志文件”对话框。
- 4. 在 File\_name 文本框中键入文件名称，并点击保存来返回至跟踪页面。
- 5. 要选择定制跟踪动态链接库(DLL)，请点击选择 DLL 来显示“选择定制跟踪 dll”对话框，并键入值，如下列示例所示：
  - “文件名称”：test2\_dsn
  - “文件类型”：Dynamic link libraries (\*.dll)选择文件，或在 File\_name 文本框中输入文件名称。
- 6. 点击打开来显示“跟踪”页面。
- 7. 点击确定来保存更改。

3.3 制作连接的连接字符串关键字

请使用连接字符串关键字来制作带有或不带有 DSN 以及带有 DRIVER 关键字的连接。

下表罗列可在制作连接中使用的连接字符串关键字：

关键字	简短版本
-----	------

关键字	简短版本
CLIENT_LOCALE	CLOC
CONNECTDATABASE	CONDB
CURSORBEHAVIOR	CURB
DATABASE	DB
DB_LOCALE	DLOC
DESCRIBEDECIMALFLOATPOINT	DDFP
DESCRIPTION	DESC
DONOTUSELVARCHAR	DNL
DRIVER	DRIVER
DSN	DSN
ENABLEINSERTCURSORS	ICUR
ENABLESCROLLABLECURSORS	SCUR
EXCLUSIVE	XCL
FETCHBUFFERSIZE	FBC
FILEDSN	FILEDSN
HOST	HOST
NEEDODBCTYPESONLY	ODTYP
OPTIMIZEAUTOCOMMIT	OAC
OPTIONS	OPT
OPTOFC	OPTOFC
PWD	PWD
REPORTCHARCOLASWIDECHARCOL	RCWC
REPORTKEYSETCURSORS	RKC
SAVEFILE	SAVEFILE
SERVER	SRVR
SERVICE	SERV
SINGLETHREADED	SINGLETH
SKIPPARSING	SKIPP
TRANSLATIONDLL	TDLL
TRANSLATIONOPTION	TOPT
UID	UID

### 3.4 DSN 迁移工具

可通过创建带有 .ini 扩展名的文本文件，来使用 DSN 迁移工具。

要使用与 GBase 8s ODBC Driver 在一起的 DSN 迁移工具 `dsnmigrate.exe`，请创建扩展名为 .ini 的文本文件；然后输入您想要迁移或恢复的 DSN 的名称和值。迁移日志文件位于 `%GBS_HOME%\release\dsnMigr.log` 中。恢复信息位于 `%GBS_HOME%\release\dsnMigr.sav` 中。

下列限制适用：

- 仅可由创建用户 DSN 的用户使用或迁移该 DSN。
- 系统的所有用户都可使用系统 DSN。
- 文件 DSN 需要对该文件的写权限。

### 3.4.1 设置和使用 DSN 迁移工具

以文本编辑器创建文本文件，来设置和使用 DSN 迁移工具。

要设置和使用 DSN 迁移工具，请：

1. 打开文本编辑器，并以 .ini 扩展名来创建文本文件。
2. 在文件中，为要修改的每一 DSN 类型（用户、系统和文件）创建一个部分。
3. 在每一部分中独立的行上，通过使用下列格式来指定您的 DSN：

**DSNname=drivername**

*drivername* 必须为 GBase 8s ODBC DRIVER

4. 要运行 `dsnmigrate.exe`，请使用下列命令：

**dsnMigrate -f filename**

在此，*filename* 是在步骤 1 中创建的文本文件的名称

### 3.4.2 DSN 迁移工具示例

该 DSN 迁移工具示例说明迁移至 GBase 8s ODBC Driver 的各种 DSN。

在下列示例中，名为 **Test1** 的 DSN 迁移至 GBase 8s ODBC DRIVER，名为 **Test2** 的 DSN 迁移至 GBase 8s ODBC DRIVER。两个 DSN 都限定于创建它们的用户。

**[User DSN]**

Test1=GBase 8s ODBC DRIVER

Test2=GBase 8s ODBC DRIVER

在第二个示例中，名为 **Test3** 的 DSN 迁移至 GBase 8s ODBC DRIVER，名为 **Test4** 的 DSN 迁移至它的原始 DSN。系统的所有用户都可使用这两个 DSN。迁移这些系统 DSN 的用户必须拥有修改 ODBC 系统 DSN 注册表条目的许可。

**[System DSN]**

Test3=GBase 8s ODBC DRIVER

Test4=restore

在第三个示例中，名为 **test5.dsn** 和 **test6.dsn** 的两个文件 DSN 迁移至 GBase 8s ODBC DRIVER。

```
[File DSN]
C:\Program Files\ODBC\Data Sources\test5.dsn=GBase 8s ODBC DRIVER
C:\Program Files\ODBC\Data Sources\test6.dsn=GBase 8s ODBC DRIVER
```

## 4 数据类型

这些主题包含关于由 GBase 8s ODBC Driver 支持的数据类型的信息。

### 4.1 数据类型

GBase 8s ODBC Driver 支持五种不同的数据类型。

下表描述 GBase 8s ODBC Driver 支持的数据类型。

数据类型	描述	示例
GBase 8s SQL 数据类型	GBase 8s 数据库服务器使用的数据类型	CHAR( <i>n</i> )
GBase 8s ODBC Driver SQL 数据类型	对应于 GBase 8s SQL 数据类型的数据类型	SQL_CHAR
标准 C 数据类型	C 编译器定义的数据类型	unsigned char
GBase 8s ODBC Driver typedef	对应于标准 C 数据类型的 typedef	UCHAR
GBase 8s ODBC Driver C 数据类型	对应于标准 C 数据类型的数据类型	SQL_C_CHAR

### 4.2 SQL 数据类型

GBase 8s 数据库服务器使用 SQL 数据类型。

要获取关于 GBase 8s SQL 数据类型的详尽信息，请参阅《GBase 8s SQL 指南：参考》、《GBase 8s SQL 指南：教程》和《GBase 8s 用户定义的例程和数据类型开发者指南》。

#### 4.2.1 标准 SQL 数据类型

标准 GBase 8s SQL 数据类型有对应的 GBase 8s ODBC Driver 数据类型。

下表罗列标准 GBase 8s SQL 数据类型及其对应的 GBase 8s ODBC Driver 数据类型。

GBase 8s SQL 数据类型	GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	描述
-------------------	--	----

GBase 8s SQL 数据类型	GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	描述
BIGINT	SQL_INFX_BIGINT	精度 10、小数位数 0、取值范围 $n$ 的有符号数值值 $(-(263 - 1) \leq n \leq 263 - 1)$
BIGSERIAL	SQL_INFX_BIGINT	最大为 $(263 - 1)$ 的连续正整数
BOOLEAN	SQL_BIT	't' 或 'f'
BYTE	SQL_LONGVARBINARY	变长的二进制数据
CHAR( $n$ )、 CHARACTER( $n$ )	SQL_CHAR	定长 $n$ $(1 \leq n \leq 32,767)$ 的字符串
CHARACTER VARYING( $m, r$ )	SQL_VARCHAR	变长字符串, 最大长度 $m$ ( $1 \leq m \leq 255$ ), 最小保留空间量 $r$ ( $0 \leq r < m$ )
DATE	SQL_DATE	日历日期
DATETIME	SQL_TIMESTAMP	日历日期和当日时刻
DEC( $p,s$ )、 DECIMAL( $p, s$ )	SQL_DECIMAL	带符号的数值值, 精度 $p$ , 小数位数 $s$ : $(1 \leq p \leq 32; 0 \leq s \leq p)$  <b>重要:</b> <ul style="list-style-type: none"> <li>如果使用小数位数 <math>&gt; 14</math>, 则可能导致取整不一致。</li> <li>当 DECIMAL 列包含浮点数据时, GBase 8s ODBC 报告该列的小数位数为 255。这与定点数据不同, 其最大小数位数为 32。DataDirect ODBC Driver 始终返回小数位数零。</li> </ul>
DOUBLE PRECISION	SQL_DOUBLE	带符号的数值值, 与标准 C <b>double</b> 数据类型的特征相同
FLOAT	SQL_DOUBLE	带符号数值值, 其特征与标准 C <b>double</b> 数据类型相同
IDSSECURITYLABEL		内建的 DISTINCT OF VARCHAR(128) 数据类型; 仅限于基于标签的访问控制使用。
INT、INTEGER	SQL_INTEGER	带符号的数值值, 精度 10, 小数位数 0, 取值范围 $n$ $(-2,147,483,647 \leq n \leq 2,147,483,647)$

GBase 8s SQL 数据类型	GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	描述
INT8	SQL_BIGINT	带符号的数值值, 精度 10, 小数位数 0, 取值范围 $n$ $(- (263 - 1) \leq n \leq 263 - 1)$
INTERVAL MONTH( $p$ )	SQL_INTERVAL_MONTH	两个日期之间的月数; $p$ 为间隔前导精度。
INTERVAL YEAR( $p$ )	SQL_INTERVAL_YEAR	两个日期之间的年和月数; $p$ 是间隔前导精度。
INTERVAL YEAR( $p$ ) TO MONTH	SQL_INTERVAL_YEAR_T O_MONTH	两个日期之间的年和月数; $p$ 是间隔前导精度。
INTERVAL DAY( $p$ )	SQL_INTERVAL_DAY	两个日期之间的天数; $p$ 是间隔前导精度。
INTERVAL HOUR( $p$ )	SQL_INTERVAL_HOUR	两个日期时刻之间的小时数; $p$ 是间隔前导精度。
INTERVAL MINUTE( $p$ )	SQL_INTERVAL_MINUTE	两个日期/时刻之间的分钟数; $p$ 是间隔前导精度。
INTERVAL SECOND( $p, q$ )	SQL_INTERVAL_SECOND	两个日期/时刻之间的秒数; $p$ 是间隔前导精度, $q$ 是间隔第二精度。
INTERVAL DAY( $p$ ) TO HOUR	SQL_INTERVAL_DAY_TO_ HOUR	两个日期/时刻之间的天/小时数; $p$ 是间隔前导精度。
INTERVAL DAY( $p$ ) TO MINUTE	SQL_INTERVAL_DAY_TO_ MINUTE	两个日期/时刻之间的天/小时/分钟数; $p$ 是间隔前导精度。
INTERVAL DAY( $p$ ) TO SECOND( $q$ )	SQL_INTERVAL_DAY_TO_ SECOND	两个日期/时刻之间的天/小时/分钟/秒数; $p$ 是间隔前导精度, $q$ 是间隔第二精度。
INTERVAL HOUR ( $p$ ) TO MINUTE	SQL_INTERVAL_HOUR_T O_MINUTE	两个日期/时刻之间的小时/分钟数; $p$ 是间隔前导精度。
INTERVAL HOUR( $p$ ) TO SECOND( $q$ )	SQL_INTERVAL_HOUR_T O_SECOND	两个日期/时刻之间的小时/分钟/秒数; $p$ 是间隔前导精度, $q$ 是间隔第二精度。
INTERVAL MINUTE( $p$ ) TO SECOND( $q$ )	SQL_INTERVAL_MINUTE_ TO_SECOND	两个日期/时刻之间的分钟/秒数; $p$ 是间隔前导精度, $q$ 是间隔第二精度。
LVARCHAR	SQL_VARCHAR	变长字符串, 长度为 $l$ $(255 \leq l \leq 32,000)$
MONEY( $p, s$ )	SQL_DECIMAL	带符号的数值值, 精度 $p$ , 小数位数 $s$ $(1 \leq p \leq 32; 0 \leq s \leq p)$
NUMERIC	SQL_NUMERIC	带符号的精确数值值, 精度 $p$ , 小数

GBase 8s SQL 数据类型	GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	描述
		位数 $s$ $(1 \leq p \leq 15; 0 \leq s \leq p)$
REAL	SQL_REAL	带符号的数值值, 与标准 C float 数据类型有相同的特征
SERIAL	SQL_INTEGER	连续的 INTEGER
SERIAL8	SQL_BIGINT	连续的 INT8
SMALLFLOAT	SQL_REAL	带符号的数值值, 与标准 C float 数据类型有相同的特征
SMALLINT	SQL_SMALLINT	带符号的数值值, 精度 5, 小数位数 0, 取值范围 $n$ $(-32,767 \leq n \leq 32,767)$
TEXT	SQL_LONGVARCHAR	变长字符串
VARCHAR( $m, r$ )	SQL_VARCHAR	变长字符串, 最大长度 $m$ ( $1 \leq m \leq 255$ ), 最小保留空间量 $r$ ( $0 \leq r < m$ )

Visual Basic 客户机侧游标

当您使用 Visual Basic 客户机侧游标来执行与使用 CHAR 或 LVARCHAR 列有关的 rowset 更新操作时, 若这些列长度大于或等于 16,385, GBase 8s ODBC Driver 可能返回错误。

当长度大于或等于 16,385 时, Visual Basic 将该 SQL 数据类型发送至 SQLBindParameter, 作为 SQL\_LONGVARCHAR 而不是 SQL\_VARCHAR。GBase 8s ODBC Driver 将 SQL\_LONGVARCHAR 映射为 TEXT 数据类型。因此, 应用程序可能看到错误:

[GBase 8s][GBase 8s ODBC Driver]No cast from text to lvarchar

或

[GBase 8s][GBase 8s ODBC Driver]Illegal attempt to use Text/Byte host variable.

4.2.2 GLS 的附加 SQL 数据类型

GLS 的附加 SQL 数据类型有对应的 GBase 8s ODBC Driver 数据类型。

下表罗列 GLS 的附加 GBase 8s SQL 数据类型, 及其对应的 GBase 8s ODBC Driver 数据类型。GBase 8s ODBC 驱动程序不提供完全的 GLS 支持。要获取关于 GLS 的更多信息, 请参阅《GBase 8s GLS 用户指南》。

GBase 8s SQL 数据类型	GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	描述
-------------------	--	----

GBase 8s SQL 数据类型	GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	描述
NCHAR( <i>n</i> )	SQL_CHAR	定长 <i>n</i> 的字符串 ( $1 \leq n \leq 32,767$ )。排序方式依赖于语言环境。
NVARCHAR( <i>m</i> , <i>r</i> )	SQL_VARCHAR	变长字符串, 最大长度 <i>m</i> ( $1 \leq m \leq 255$ ), 最小保留空间量 <i>r</i> ( $0 \leq r < m$ )。排序方式依赖于语言环境。

4. 2. 3 GBase 8s 的附加 SQL 数据类型

GBase 8s 的附加 GBase 8s SQL 数据类型有对应的 GBase 8s ODBC Driver 数据类型。

下表罗列 GBase 8s 的附加 GBase 8s SQL 数据类型, 及其对应的 GBase 8s ODBC Driver 数据类型。要使用 GBase 8s 的 GBase 8s ODBC 驱动程序 SQL 数据类型, 请包括 infxcli.h。

GBase 8s SQL 数据类型	GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	描述
集合 (LIST、MULTISET、SET)	任何 GBase 8s ODBC 驱动程序 SQL 数据类型	由一个或多个元素组成的合成的值, 在此, 每一元素都有相同的数据类型。
DISTINCT	任何 GBase 8s ODBC 驱动程序 SQL 数据类型	存储方式与其源数据类型相同的 UDT, 但有不同的强制转型和函数
OPAQUE (固定的)	SQL_INFX_UDT_FIXED	带有内部结构的定长 UDT, 对于所有可能的值都有相同大小
OPAQUE (变化的)	SQL_INFX_UDT_VARYING	带有内部结构的变长 UDT, 对于每一不同的值, 可有不同的大小
row (命名的行、未命名的行)	任何 GBase 8s ODBC Driver SQL 数据类型	由一个或多个元素组成的合成的值, 在此, 每一元素可有不同的数据类型。
智能大对象 (BLOB 或 CLOB)	SQL_IFMX_UDT_BLOB SQL_IFMX_UDT_CLOB	存储在磁盘上 sbospace 中的大对象, 且是可恢复的。

4. 2. 4 精度、小数位数、长度和显示大小

为 SQL 值取得和设置精度、小数位数、长度和显示大小的函数对其输入参数有大小限制。因此, 这些值限定为 SDWORD 的大小, 其最大值为 2,147,483,647。下表描述这些值。

值	数值数据类型的描述	非数值数据类型的描述
---	-----------	------------



值	数值数据类型的描述	非数值数据类型的描述
精度	最大的数字个数。	或为最大长度,或为指定的长度。
小数位数	小数点之后数字的最大个数。对于浮点值,未定义小数位数,因为其小数点右边的数字个数不固定。	不适用。
长度	当将一个值转换为为其缺省的 C 数据类型时,函数返回的最大字节数。	当将一个值转换为为其缺省的 C 数据类型时,函数返回的最大字节数。该长度不包括 NULL 终止字节。
显示大小	需要以字符形式显示数据的最大字节数。	需要以字符形式显示数据的最大字节数。

标准 SQL 数据类型

对于标准 GBase 8s ODBC Driver SQL 数据类型,查看精度、小数位数、长度和显示大小的值。

下表描述标准 GBase 8s ODBC Driver SQL 数据类型的精度、小数位数、长度和显示大小。

GBase 8s ODBC 驱动程序 sql 数据类型 (fSqlType)	描述
SQL_BIGINT	<div>精度 19。对于此数据类型,SQLBindParameter 忽略 cbColDef 的值。</div> <div>小数位数 0。对于此数据类型,SQLBindParameter 忽略 ibScale 的值。</div> <div>长度 8 字节</div> <div>显示大小 20 位。一位表示符号。</div>
SQL_BIT	<div>精度 1。对于此数据类型,SQLBindParameter 忽略 cbColDef 的值。</div> <div>小数位数 0。对于此数据类型,SQLBindParameter 忽略 ibScale 的值。</div> <div>长度 1 字节</div> <div>显示大小 1 位</div>

GBase 8s ODBC 驱动程序 sql 数据类型 (fSqlType)	描述
SQL_CHAR	<p>精度 与长度相同</p> <p>小数位数 不适用。对于此数据类型，SQLBindParameter 忽略 <i>ibScale</i> 的值。</p> <p>长度 指定的长度。例如，CHAR(10) 的长度为 10 字节。</p> <p>显示大小 与长度相同。</p>
SQL_DATE	<p>精度 10。对于此数据类型，SQLBindParameter 忽略 <i>cbColDef</i> 的值。</p> <p>小数位数 不适用。对于此数据类型，SQLBindParameter 忽略 <i>ibScale</i> 的值。</p> <p>长度 6 字节</p> <p>显示大小 10 位，格式为 yyyy-mm-dd。</p>
SQL_DECIMAL	<p>精度 指定的精度。例如，DECIMAL(12, 3) 的精度是 12。</p> <p>小数位数 指定的小数位数。例如，DECIMAL(12, 3) 的小数位数是 3。</p> <p>长度 指定的精度加 2。例如，DECIMAL(12, 3) 的长度是 14 字节。两个额外的字节用于符号和小数点，因为函数作为字符串返回此数据类型。</p> <p>显示大小 与长度相同。</p>
SQL_DOUBLE	<p>精度。 15。对于此数据类型，SQLBindParameter 忽略 <i>cbColDef</i> 的值。</p> <p>小数位数 不适用。对于此数据类型，SQLBindParameter 忽略 <i>ibScale</i> 的值。</p> <p>长度 8 字节</p> <p>显示大小 22 位。这些位代表一个符号、15 个数值字符、一个小数点、字母 E，另一符号，以及另外两个数值字符。</p>

GBase 8s ODBC 驱动程序 sql 数据类型 (fSqlType)	描述
SQL_INTEGER	<p>精度 10。对于此数据类型, SQLBindParameter 忽略 <i>cbColDef</i> 的值。</p> <p>小数位数 0。对于此数据类型, SQLBindParameter 忽略 <i>ibScale</i> 的值。</p> <p>长度 4 字节</p> <p>显示大小 11 位。一位代表符号。</p>
SQL_LONGVARIABLE	<p>精度 与长度相同。</p> <p>小数位数 不适用。对于此数据类型, SQLBindParameter 忽略 <i>ibScale</i> 的值。</p> <p>长度 最大长度。如果函数无法确定最大长度, 则它返回 SQL_NO_TOTAL。</p> <p>显示大小 最大长度的 2 倍。如果函数无法确定最大长度, 则它返回 SQL_NO_TOTAL。</p>
SQL_LONGVARCHAR	<p>精度 与长度相同。</p> <p>小数位数 不适用。对于此数据类型, SQLBindParameter 忽略 <i>ibScale</i> 的值。</p> <p>长度 最大长度。如果函数无法确定最大长度, 则它返回 SQL_NO_TOTAL。</p> <p>显示大小 与长度相同。</p>
SQL_REAL	<p>精度 7。对于此数据类型, SQLBindParameter 忽略 <i>cbColDef</i> 的值。</p> <p>小数位数 不适用。对于此数据类型, SQLBindParameter 忽略 <i>ibScale</i> 的值。</p> <p>长度 4 字节</p> <p>显示大小 13 位。这些位表示一个符号、7 个数值字符、一个小数点、字母 E、另一个符号和另外 2 个数值字符。</p>

GBase 8s ODBC 驱动程序 sql 数据类型 (fSqlType)	描述
SQL_SMALLINT	精度 5。对于此数据类型, SQLBindParameter 忽略 <i>cbColDef</i> 的值。 小数位数 0。对于此数据类型, SQLBindParameter 忽略 <i>ibScale</i> 的值。 长度 2 字节 显示大小 6 位。一位表示符号。
SQL_TIMESTAMP	精度 8。对于此数据类型, SQLBindParameter 忽略 <i>cbColDef</i> 的值。 小数位数 FRACTION 域中的位数。 长度 16 字节 显示大小 19 位或更多: <ul style="list-style-type: none"><li>如果时间戳的小数位数为 0: 19 位, 格式为 yyyy-mm-dd hh:mm:ss。</li><li>如果时间戳的小数位数超出 0: 20 位加上 FRACTION 域中的数字, 格式为 yyyy-mm-dd hh:mm:ss.f...</li></ul>
SQL_VARCHAR	精度 与长度相同。 小数位数 不适用。对于此数据类型, SQLBindParameter 忽略 <i>ibScale</i> 的值。 长度 指定的长度。例如, VARCHAR(10) 的长度是 10 字节。 显示大小 与长度相同。

GBase 8s 的附加 SQL 数据类型

对于 GBase 8s ODBC Driver SQL 数据类型, 查看精度、小数位数、长度和显示大小的值。

对于 GBase 8s , 下表描述 GBase 8s ODBC Driver SQL 数据类型的精度、小数位数、长度和显示大小。

GBase 8s ODBC 驱动程序 sql 数据类型 (fSqlType)	描述
---	----

GBase 8s ODBC 驱动程序 sql 数据类型 (fSqlType)	描述
SQL_IFMX_UDT_BLOB	精度 变量值。要确定此值，请调用返回列的精确度的函数。 小数位数 不适用。对于此数据类型，返回列的小数位数的函数返回 -1。 长度 变量值。要确定此值，请调用返回列的长度的函数。 显示大小 变量值。要确定此值，请调用返回列的显示大小的函数。
SQL_IFMX_UDT_CLOB	精度 变量值。要确定此值，请调用返回列的精确度的函数。 小数位数 不适用。对于此数据类型，返回列的小数位数的函数返回 -1。 长度 变量值。要确定此值，请调用返回列的长度的函数。 显示大小 变量值。要确定此值，请调用返回列的显示大小的函数。
SQL_INFX_UDT_FIXED	精度 变量值。要确定此值，请调用返回列的精确度的函数。 小数位数 不适用。对于此数据类型，返回列的小数位数的函数返回 -1。 长度 变量值。要确定此值，请调用返回列的长度的函数。 显示大小 变量值。要确定此值，请调用返回列的显示大小的函数。
SQL_INFX_UDT_VARYING	精度 变量值。要确定此值，请调用返回列的精确度的函数。 小数位数 不适用。对于此数据类型，返回列的小数位数的函数返回 -1。 长度 变量值。要确定此值，请调用返回列的长度的函数。 显示大小 变量值。要确定此值，请调用返回列的显示大小的函数。

## 4.3 C 数据类型

GBase 8s ODBC Driver 应用程序使用 C 数据类型来存储该应用程序处理的值。

下表描述 GBase 8s ODBC Driver 提供的 C 数据类型。

**重要：** GBase 8s ODBC 驱动程序函数中的字符串参数是无符号的。因此，在使用它作为 GBase 8s ODBC 驱动程序函数中的参数之前，需要强制转型 CString 对象为无符号字符串。

值	GBase 8s ODBC 驱动程序 C 数据 类型 (fCType)	GBase 8s ODBC 驱动程序 typedef	标准 C 数据类型
二进制	SQL_C_BINARY	UCHAR FAR *	unsigned char FAR *
布尔	SQL_C_BIT	UCHAR	unsigned char
字符	SQL_C_CHAR	UCHAR FAR *	unsigned char FAR *
宽字符	SQL_C_WCHAR	WCHAR FAR *	wchar_t FAR *
日期	SQL_C_DATE	DATE_STRUCT	struct tagDATE_STRUCT{ SWORD year; UWORD month; UWORD day; }
间隔	SQL_C_INTERVAL_YEAR	SQL_INTERVAL_STRUCT	C 间隔结构
	SQL_C_INTERVAL_MONTH	SQL_INTERVAL_STRUCT	C 间隔结构
	SQL_C_INTERVAL_DAY	SQL_INTERVAL_STRUCT	C 间隔结构
	SQL_C_INTERVAL_HOUR	SQL_INTERVAL_STRUCT	C 间隔结构
	SQL_C_INTERVAL_MINUTE	SQL_INTERVAL_STRUCT	C 间隔结构
	SQL_C_INTERVAL_SECOND	SQL_INTERVAL_STRUCT	C 间隔结构
	SQL_C_INTERVAL_YEAR_TO_MONTH	SQL_INTERVAL_STRUCT	C 间隔结构
	SQL_C_INTERVAL_DAY_TO_HOUR	SQL_INTERVAL_STRUCT	C 间隔结构
	SQL_C_INTERVAL_DAY_TO_MINUTE	SQL_INTERVAL_STRUCT	C 间隔结构

值	GBase 8s ODBC 驱动程序 C 数据类型 (fCType)	GBase 8s ODBC 驱动程序 typedef	标准 C 数据类型
	<b>SQL_C_INTERVAL_DAY_TO_SECOND</b>	SQL_INTERVAL_STRUCT	C 间隔结构
	<b>SQL_C_INTERVAL_HOUR_TO_MINUTE</b>	SQL_INTERVAL_STRUCT	C 间隔结构
	<b>SQL_C_INTERVAL_HOUR_TO_SECOND</b>	SQL_INTERVAL_STRUCT	C 间隔结构
	<b>SQL_C_INTERVAL_MINUTE_TO_SECOND</b>	SQL_INTERVAL_STRUCT	C 间隔结构
数值	<b>SQL_C_DOUBLE</b>	SDOUBLE	signed double
	<b>SQL_C_FLOAT</b>	SFLOAT	signed float
	<b>SQL_C_LONG</b>	SDWORD	signed long int
	<b>SQL_C_NUMERIC</b>	SQL_NUMERIC_STRUCT	struct tag SQL_NUMERIC_STRUCT { SQLCHAR precision; SQLCHAR scale; SQLCHAR sign; SQLCHAR val [SQL_MAX_NUMERIC_LEN]; }SQL_NUMERIC_STRUCT;
	<b>SQL_C_SHORT</b>	SWORD	signed short int
	<b>SQL_C_SLONG</b>	SDWORD	signed long int
	<b>SQL_C_SSHORT</b>	SWORD	signed short int
	<b>SQL_C_STINYINT</b>	SCHAR	signed char
	<b>SQL_C_TINYINT</b>	SCHAR	signed char
	<b>SQL_C_ULONG</b>	UDWORD	unsigned long int
	<b>SQL_C_USHORT</b>	UWORD	unsigned short int
	<b>SQL_C_UTINYINT</b>	UCHAR	unsigned char
时间戳	<b>SQL_C_TIMESTAMP</b>	TIMESTAMP_STRUCT	struct tagTIMESTAMP_STRUCT { SWORD year; UWORD month; UWORD day; UWORD hour; UWORD minute; UWORD second; UDWORD fraction; }

#### 4.3.1 C 间隔结构

通过使用 C 间隔结构，来为 SQL 间隔数据类型指定 C 数据类型。

下列结构为 SQL 间隔数据类型指定 C 数据类型：

```
typedef struct tagSQL_INTERVAL_STRUCT
{
    SQLINTERVAL interval_type;
    SQLSMALLINT interval_sign;
    union
    {
        SQL_YEAR_MONTH_STRUCT year_month;
        SQL_DAY_SECOND_STRUCT day_second;
    } interval;
}SQLINTERVAL_STRUCT;

typedef enum
{
    SQL_IS_YEAR=1,
    SQL_IS_MONTH=2,
    SQL_IS_DAY=3,
    SQL_IS_HOUR=4,
    SQL_IS_MINUTE=5,
    SQL_IS_SECOND=6,
    SQL_IS_YEAR_TO_MONTH=7,
    SQL_IS_DAY_TO_HOUR=8,
    SQL_IS_DAY_TO_MINUTE=9,
    SQL_IS_DAY_TO_SECOND=10,
    SQL_IS_HOUR_TO_MINUTE=11,
    SQL_IS_HOUR_TO_SECOND=12,
    SQL_IS_MINUTE_TO_SECOND=13,
}SQLINTERVAL;

typedef struct tagSQL_YEAR_MONTH
{
    SQLINTEGER year;
    SQLINTEGER month;
}SQL_YEAR_MONTH_STRUCT;

typedef struct tagSQL_DAY_SECOND
{
    SQLINTEGER day;
    SQLINTEGER hour;
    SQLINTEGER minute;
    SQLINTEGER second
```



### 4.3.2 传输数据

在使用同一 DBMS 的数据源之中，可以安全地以 DBMS 使用的内部形式传输数据。

对于特定的数据，源数据源和目标数据源中的SQL 数据类型必须相同。C 数据类型为 **SQL\_C\_BINARY**。

当调用 **SQLFetch**、**SQLExtendedFetch** 或 **SQLGetData** 来从数据源检索此数据时，GBase 8s ODBC Driver 检索该数据，并不经转换，将它传输至 **SQL\_C\_BINARY**类型的存储位置。

当调用 **SQLExecute**、**SQLExecDirect** 或 **SQLPutData** 来将此数据发送至目标数据源时，GBase 8s ODBC Driver 从该存储位置检索数据，不经转换，将它传输至目标数据源。

**INT8**、**SERIAL8** 和 **BIGSERIAL** 数据类型的二进制表示是两个无符号长整型数组，后跟一个指示符号的短整数。符号字段为 1 表示正值，-1 表示负值，0 表示空值。

**重要：** 在 DBMS 之中，以此方式传输任何数据（二进制数据除外）的应用程序不可互操作。

## 4.4 报告标准 ODBC 类型

GBase 8s ODBC Driver 仅支持支持标准 ODBC 数据类型的现有应用程序。请勾选 DSN 选项**报告标准 ODBC 类型**，来开启此行为。

当应用程序设置此选项时，驱动程序设置下列行为：

- 对于定义了新数据类型的所有驱动程序，仅报告标准 ODBC 数据类型。
- 对于智能大对象（LO），可访问数据类型访问方法为 **SQL\_LONGVARCHAR** 和 **SQL\_LONGVARBINARY**。换句话说，**SQL\_LONGVARCHAR** 和 **SQL\_LONGVARBINARY** 行为就像简单大对象、byte 和 text 一样。
- 将 **defaultUDTfetchtype** 设置为 **SQL\_C\_CHAR**。

然而，可分别控制上述每一活动作为一个连接或语句级选项。请使用下列连接和语句级属性：

- **SQL\_INFX\_ATTR\_ODBC\_TYPES\_ONLY**
- **SQL\_INFX\_ATTR\_LO\_AUTOMATIC**
- **SQL\_INFX\_ATTR\_DEFAULT\_UDT\_FETCH\_TYPE**

应用程序可使用 **SQLSetConnectAttr** 和 **SQLSetStmtAttr** 来设置和复位这些值。（ODBC 2.x 应用程序可等同地使用 **SQLSetConnectOption** 与 **SQLSetStmtOption**。）

### 4.4.1 SQL\_INFX\_ATTR\_ODBC\_TYPES\_ONLY

应用程序可将 **SQL\_INFX\_ATTR\_ODBC\_TYPES\_ONLY** 属性设置为值 **SQL\_TRUE** 或 **SQL\_FALSE**。

可在连接级和语句级设置和复位此属性。在同一连接之下分配的所有语句都继承此值。或者，每一语句可更改此属性。在缺省情况下，设置此属性为 **SQL\_FALSE**。

通过使用 `SQLSetConnectAttr` 和 `SQLSetStmtAttr`（ODBC 2.x 中的 `SQLSetConnectOption` 和 `SQLSetStmtOption`），应用程序可更改此属性的值。通过使用 `SQLGetConnectAttr` 和 `SQLGetStmtAttr`（ODBC 2.x 中的 `SQLGetConnectOption` 和 `SQLGetStmtOption`），应用程序可检索这些值。

当将 `SQL_INFX_ATTR_LO_AUTOMATIC` 设置为 `SQL_FALSE` 时，不可将此属性设置为 `SQL_TRUE`。返回报告下列消息的错误消息：

Attribute cannot be set. LoAutomatic should be ON to set this value.

应用程序应先将 `SQL_INFX_ATTR_LO_AUTOMATIC` 属性设置为 `SQL_TRUE`，然后，将 `SQL_INFX_ATTR_ODBC_TYPES_ONLY` 属性设置为 `SQL_TRUE`。

#### 4. 4. 2 SQL\_INFX\_ATTR\_LO\_AUTOMATIC

应用程序可将 `SQL_INFX_ATTR_LO_AUTOMATIC` 属性设置为值 `SQL_TRUE` 或 `SQL_FALSE`。

可在连接级和语句级设置和复位此属性。在同一连接之下分配的所有语句都继承此值。或者，每一语句都可更改此属性。缺省情况下，设置此属性为 `SQL_FALSE`。

通过使用 `SQLSetConnectAttr` 和 `SQLSetStmtAttr`（ODBC 2.x 中的 `SQLSetConnectOption` 和 `SQLSetStmtOption`），应用程序可更改此属性的值。通过使用 `SQLGetConnectAttr` 和 `SQLGetStmtAttr`（ODBC 2.x 中的 `SQLGetConnectOption` 和 `SQLGetStmtOption`），应用程序可检索这些值。

当将 `SQL_INFX_ATTR_ODBC_TYPES_ONLY` 设置为 `SQL_TRUE` 时，不可将属性 `SQL_INFX_ATTR_LO_AUTOMATIC` 设置为 `SQL_FALSE`。返回报告下列消息的错误消息：

Attribute cannot be set. ODBC types only should be OFF to set this value.

应用程序应先将属性 `SQL_INFX_ODBC_TYPES_ONLY` 设置为 `SQL_FALSE`，然后将属性 `SQL_INFX_ATTR_LO_AUTOMATIC` 设置为 `SQL_FALSE`。

#### 4. 4. 3 SQL\_INFX\_ATTR\_DEFAULT\_UDT\_FETCH\_TYPE

应用程序可将 `SQL_INFX_ATTR_DEFAULT_UDT_FETCH_TYPE` 属性设置为 `SQL_C_CHAR` 或 `SQL_C_BINARY`，来设置 UDT 的缺省访存类型。

依赖于下列条件，设置此属性的缺省值：

- 如果报告标准 ODBC 类型的 DSN 设置为 ON，则将 `DefaultUDTFetchType` 的值设置为 `SQL_C_CHAR`。
- 如果报告标准 ODBC 类型的 DSN 设置为 OFF，则将 `DefaultUDTFetchType` 的值设置为 `SQL_C_BINARY`。
- 如果用户已设置了注册表键，只要未设置报告标准 ODBC 类型，则将 `DefaultUDTFetchType` 的值设置为注册表中的值。

通过使用 `SQLSetConnectAttr` 和 `SQLSetStmtAttr`（ODBC 2.x 中的 `SQLSetConnectOption` 和 `SQLSetStmtOption`），应用程序可更改此属性的值。通过使用 `SQLGetConnectAttr` 和 `SQLGetStmtAttr`（ODBC 2.x 中的 `SQLGetConnectOption` 和 `SQLGetStmtOption`），应用程序可检索这些值。

将报告标准 ODBC 类型设置为 ON，会始终将 `DefaultUDTFetchType` 重写为 `SQL_C_CHAR`。

4.4.4 报告宽字符列

GBase 8s 服务器不支持宽字符数据类型。

当应用程序设置 `Report Char Columns as Wide Char Columns` 选项时，驱动程序设置下列行为：

- `SQLDescribeCol` 报告 `char` 列为宽 `char` 列
- 报告 `SQL_CHAR` 列为 `SQL_WCHAR`
- 报告 `SQL_VARCHAR` 列为 `SQL_WVARCHAR`
- 报告 `SQL_LONGVARCHAR` 列为 `SQL_WLONGVARCHAR`
- 缺省值为：0（禁用的）

设置 `Report Char Columns as Wide Char Columns` 选项之后，请以有下列行为的 SQL 数据类型来调用 `SQLBindParameter`：

- 将 `SQL_WCHAR` 映射至 `SQL_CHAR`
- 将 `SQL_WVARCHAR` 映射至 `SQL_VARCHAR`
- 将 `SQL_WLONGVARCHAR` 映射至 `SQL_LONGVARCHAR`

4.4.5 报告标准 ODBC 数据类型的 DSN 设置

对于 UNIX<sup>™</sup> 和 Windows<sup>™</sup>，可添加新的 DSN 选项 `NeedODBCTypesOnly`。

对于 UNIX，在 `odbc.ini` 文件中的 DSN 设置之下，添加新的 DSN 选项 `NeedODBCTypesOnly`：

```
[GBase 8s]
    Driver=/gbasedbt/lib/cli/libthcli.so
    Description=GBase 8s ODBC Driver
    ...
    NeedODBCTypesOnly=1
```

对于 Windows，对于 GBase 8s Driver DSN [缺省值为 0]，请检查 ODBC Administration 的 **Advanced** 标签之下的此选项。

下表展示如何将 GBase 8s 数据类型映射至标准 ODBC 数据类型。

表 1. GBase 8s 和 ODBC 数据类型映射

GBase 8s	ODBC
----------	------

GBase 8s	ODBC
Bigint	SQL_BIGINT
Bigserial	SQL_BIGINT
Blob	SQL_LONGVARBINARY
Boolean	SQL_BIT
Clob	SQL_LONGVARCHAR
Int8	SQL_BIGINT
Lvarchar	SQL_VARCHAR
Serial8	SQL_BIGINT
Multiset	SQL_C_CHAR or SQL_C_BINARY
Set	SQL_C_CHAR or SQL_C_BINARY
List	SQL_C_CHAR or SQL_C_BINARY
Row	SQL_C_CHAR or SQL_C_BINARY

重要:

- 对于 multiset、set、row 和 list 数据类型，将该数据类型映射至 defaultUDTFetchType 属性集（SQL\_C\_CHAR 或 SQL\_C\_BINARY）。
- 要使得 SQL\_BIGINT 能够与 SQLBindCol 和 SQLBindParameter 正确工作，您必须使用 SQL\_C\_UBIGINT（其有 8 字节无符号整数的支持数据范围），而不使用 SQL\_C\_LONG（其有 4 字节整数的支持数据范围）。

4.5 转换数据

在本部分中使用的词语 *转换* 是广义的；它包括不经数据类型转换，将数据从一个存储位置转移至另一个。

4.5.1 标准转换

标准转换存在于 GBase 8s SQL 数据类型与 GBase 8s ODBC Driver C 数据类型之间。

仅 GBase 8s 可将数据转换为 SQL\_C\_BIT。

GBase 8s ODBC 驱动程序 C 数据类型 SQL\_C\_BINARY、SQL\_C\_CHAR 和 SQL\_C\_WCHAR 支持罗列在下表中的所有 GBase 8s SQL 数据类型之间转换。

下表展示在 GBase 8s SQL 数据类型与 GBase 8s ODBC Driver C 数据类型之间支持的转换。

表 1. 在 GBase 8s SQL 数据类型与 ODBC Driver C 数据类型之间支持的转换。  
展示在 GBase 8s SQL 数据类型与 ODBC Driver C 数据类型之间支持的转换的五列表。

SQL 数据类型	ODBC 驱动程序 C 数据类型（目标类型）			
	SQL_C_BIT	SQL_C_DATE	SQL_C_DOUBLE	SQL_C_FLOAT

SQL 数据类型	ODBC 驱动程序 C 数据类型（目标类型）			
	SQL_C_BIT	SQL_C_DATE	SQL_C_DOUBLE	SQL_C_FLOAT
BOOLEAN	是	否	否	否
CHAR、CHARACTER	是	否	是	是
CHARACTER VARYING	是	否	是	是
DATE	否	是	否	否
DATETIME	否	是	否	否
DEC、DECIMAL	是	否	是	是
DOUBLE PRECISION	否	否	是	是
FLOAT	否	否	是	是
INT、INTEGER	是	否	是	是
INT8	否	否	否	否
LVARCHAR	是	是	否	是
MONEY	否	是	是	是
NUMERIC	否	是	是	是
REAL	否	是	是	是
SERIAL	否	是	是	是
SMALLFLOAT	是	否	是	是
SMALLINT	是	否	是	是
TEXT	是	是	是	是
VARCHAR	是	是	是	是

表 2. 在 GBase 8s SQL 数据类型与 ODBC Driver C 数据类型之间支持的转换.  
展示在 GBase 8s SQL 数据类型与 ODBC Driver C 数据类型之间支持的转换的五列表。

SQL 数据类型	ODBC 驱动程序 C 数据类型（目标类型）			
	SQL_C_LONG	SQL_C_NUMERIC	SQL_C_SHORT	SQL_C_SLONG
BIGINT	是	是	否	是

SQL 数据类型	ODBC 驱动程序 C 数据类型（目标类型）			
	SQL_C_LONG	SQL_C_NUMERIC	SQL_C_SHORT	SQL_C_SLONG
BIGSERIAL	是	是	是	是
BYTE	否	否	否	否
CHAR、CHARACTER	是	是	是	是
CHARACTER VARYING	是	是	是	是
DEC、DECIMAL	是	是	是	是
DOUBLE PRECISION	是	是	是	是
FLOAT	是	是	是	是
INT、INTEGER	是	是	是	是
INT8	是	是	否	是
LVARCHAR	是	否	是	是
MONEY	是	是	是	是
NUMERIC	是	是	是	是
REAL	是	是	是	是
SERIAL	是	否	是	是
SERIAL8	是	是	是	是
SMALLFLOAT	是	是	是	是
SMALLINT	是	是	是	是
TEXT	是	是	是	是
VARCHAR	是	是	是	是

表 3. 在 GBase 8s SQL 数据类型与 ODBC Driver C 数据类型之间支持的转换。  
展示在 GBase 8s SQL 数据类型 与 ODBC Driver C 数据类型支持的展缓的五列表。

SQL 数据类型	ODBC 驱动程序 C 数据类型（目标类型）		
	SQL_C_SSHORT	SQL_C_STINYINT	SQL_C_TIMESTAMP

SQL 数据类型	ODBC 驱动程序 C 数据类型（目标类型）		
	SQL_C_SSHORT	SQL_C_STINYINT	SQL_C_TIMESTAMP
BIGINT	是	否	否
BIGSERIAL	是	否	否
CHAR、 CHARACTER	是	是	否
CHARACTER VARYING	是	是	否
DATE	否	否	是
DATETIME	否	否	是
DEC、DECIMAL	是	是	否
DOUBLE PRECISION	是	是	否
FLOAT	是	是	否
INT、INTEGER	是	是	否
INT8	是	否	否
LVARCHAR	是	是	是
MONEY	是	是	是
NUMERIC	是	是	是
REAL	是	是	是
SERIAL	是	是	是
SERIAL8	是	否	否
SMALLFLOAT	是	是	否
SMALLINT	是	是	否
TEXT	是	是	是
VARCHAR	是	是	是

ODBC 驱动程序 C 数据类型 SQL\_C\_ULONG 支持罗列在下表中的所有 SQL 数据类型之间的转换。

表 4. 在 GBase 8s SQL 数据类型与 ODBC Driver C 数据类型之间支持的转换。  
展示在 GBase 8s SQL 数据类型与 ODBC Driver C 数据类型之间支持的转换的五列表。

SQL 数据类型	ODBC 驱动程序 C 数据类型（目标类型）		
	SQL_C_TINYINT	SQL_C_USHORT	SQL_C_UTINYINT
BIGINT	否	否	否
BIGSERIAL	否	是	否
CHAR、 CHARACTER	是	是	是
CHARACTER VARYING	是	是	是
DEC、DECIMAL	是	是	是
DOUBLE PRECISION	是	是	是
FLOAT	是	是	是
INT、INTEGER	是	是	是
INT8	否	否	否
LVARCHAR	是	是	是
MONEY	是	是	是
NUMERIC	是	是	是
REAL	是	是	是
SERIAL	是	是	是
SERIAL8	否	是	否
SMALLFLOAT	是	是	是
SMALLINT	是	是	是
TEXT	是	是	是
VARCHAR	是	是	是

#### 4.5.2 GLS 的附加转换

在 GLS 的 GBase 8s SQL 数据类型与 GBase 8s ODBC Driver C 数据类型之间有支持的转换。

仅 GBase 8s 可将数据转换为 **SQL\_C\_BIT**。



GBase 8s NCHAR 和 NVARCHAR SQL 数据类型支持在下列 ODBC 驱动程序 C 数据类型（fCType）之间转换：

- **SQL\_C\_BINARY**
- **SQL\_C\_BIT**
- **SQL\_C\_CHAR**
- **SQL\_C\_DATE**
- **SQL\_C\_DOUBLE**
- **SQL\_C\_FLOAT**
- **SQL\_C\_LONG**
- **SQL\_C\_SHORT**
- **SQL\_C\_SLONG**
- **SQL\_C\_SSHORT**
- **SQL\_C\_STINYINT**
- **SQL\_C\_TIMESTAMP**
- **SQL\_C\_TINYINT**
- **SQL\_C\_ULONG**
- **SQL\_C\_USHORT**
- **SQL\_C\_UTINYINT**

#### 4.5.3 GBase 8s 的附加转换

在 GBase 8s 的附加 GBase 8s SQL 数据类型与 GBase 8s ODBC Driver C 数据类型之间有支持的转换。

GBase 8s SQL 数据类型 Collection、DISTINCT、Row 和智能大对象支持在下列 GBase 8s ODBC 驱动程序 C 数据类型（fCType）之间转换：

- **SQL\_C\_BINARY**
- **SQL\_C\_BIT**
- **SQL\_C\_CHAR**
- **SQL\_C\_DATE**
- **SQL\_C\_DOUBLE**
- **SQL\_C\_FLOAT**
- **SQL\_C\_LONG**
- **SQL\_C\_SHORT**
- **SQL\_C\_SLONG**
- **SQL\_C\_SSHORT**
- **SQL\_C\_STINYINT**
- **SQL\_C\_TIMESTAMP**
- **SQL\_C\_TINYINT**
- **SQL\_C\_ULONG**
- **SQL\_C\_USHORT**
- **SQL\_C\_UTINYINT**

GBase 8s SQL 数据类型 OPAQUE 支持在 **SQL\_C\_BINARY** 与 **SQL\_C\_CHAR** ODBC 驱动程序 C 数据类型 (fCType) 之间转换。请使用 **SQL\_C\_CHAR** 作为字符串来访问外部格式的 OPAQUE 值。请使用 **SQL\_C\_BINARY** 来访问内部二进制格式的 OPAQUE。

#### 4.5.4 将数据由 SQL 转换为 C

当调用 `SQLExtendedFetch`、`SQLFetch` 或 `SQLGetData` 时，GBase 8s ODBC Driver 从数据源检索数据。

如有必要，GBase 8s ODBC Driver 将数据由源数据类型转换为 `SQLBindCol` 中 *TargetType* 参数或 `SQLGetData` 中 *fCType* 参数指定的数据类型。最终，GBase 8s ODBC Driver 将该数据存储在由 `SQLBindCol` 或 `SQLGetData` 中 *rgbValue* 参数指向的位置中。

下面部分中的表描述 GBase 8s ODBC Driver 如何从数据源检索的数据转换它。对于给定的 GBase 8s ODBC Driver SQL 数据类型，表的第一列罗列 `SQLBindCol` 中 *TargetType* 参数和 `SQLGetData` 中 *fCType* 参数的合法输入值。第二列罗列测试的结果，通常通过使用 `SQLBindCol` 或 `SQLGetData` 中 *cbValueMax* 参数，GBase 8s ODBC Driver 执行其来确定它是否可转换该数据。对于每一结果，第三和第四列罗列在 GBase 8s ODBC Driver 试图转换该数据之后，在 `SQLBindCol` 或 `SQLGetData` 中指定的 *rgbValue* 和 *pcbValue* 参数的值。

对于每一结果，最后一列罗列由 `SQLExtendedFetch`、`SQLFetch` 或 `SQLGetData` 返回的 `SQLSTATE`。

如果 `SQLBindCol` 中的 *TargetType* 参数或 `SQLGetData` 中的 *fCType* 参数包含一个 GBase 8s ODBC Driver C 数据类型的值，而该数据类型未显示在给定的 GBase 8s ODBC Driver SQL 数据类型的表中，则 `SQLExtendedFetch`、`SQLFetch` 或 `SQLGetData` 返回 `SQLSTATE 07006`（受限制的数据类型属性违反）。如果 *fCType* 参数或 *TargetType* 参数包含一值，其指定由特定于驱动程序的 SQL 数据类型转换为 GBase 8s ODBC Driver C 数据类型，且 GBase 8s ODBC Driver 不支持此转换，则 `SQLExtendedFetch`、`SQLFetch` 或 `SQLGetData` 返回 `SQLSTATE S1C00`（驱动程序不支持）。

虽然本章节中的表未展示它，但当 SQL 数据值为空时，*pcbValue* 参数包含 `SQL_NULL_DATA`。当 GBase 8s ODBC Driver 将 SQL 数据转换为字符 C 数据时，*pcbValue* 中返回的字符计数不包括空终止字节。如果 *rgbValue* 是空指针，则 `SQLBindCol` 或 `SQLGetData` 返回 `SQLSTATE S1009`（无效的参数值）。

表中使用下列术语和惯例：

##### 数据的长度

无论数据在返回到应用程序之前是否被截断，都可以在 *rgbValue* 中返回的 C 数据的字节数。对于字符串数据，这不包括空终止字节。

##### 显示大小

以字符格式显示该数据所需要的总字节数。

##### 斜体词语

表示函数参数或 GBase 8s ODBC Driver SQL 语法的元素。

#### 缺省的 C 数据类型

可为不同的函数指定 **SQL\_C\_DEFAULT**，以便于 GBase 8s ODBC Driver 使用该 C 数据类型。

如果为 SQLBindCol 中的 *TargetType* 参数、SQLGetData 中的 *fCType* 参数或 SQLBindParameter 中的 *ValueType* 参数指定 **SQL\_C\_DEFAULT**，则 GBase 8s ODBC Driver使用输出或输入缓冲区的 C 数据类型作为缓冲区绑定的列或参数的 SQL 数据类型  
标准缺省 C 数据类型

对于每一 GBase 8s ODBC Driver SQL 数据类型，有缺省的 C 数据类型。

对于每一 GBase 8s ODBC Driver SQL 数据类型，下表展示缺省 C 数据类型。

GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	缺省的 GBase 8s ODBC 驱动程序 C 数据类型 (fCType)
SQL_BIGINT	SQL_C_CHAR
SQL_BIT	SQL_C_BITS
SQL_CHAR	SQL_C_CHAR
SQL_DATE	SQL_C_DATE
SQL_DECIMAL	SQL_C_CHAR
SQL_DOUBLE	SQL_C_DOUBLE
SQL_INTEGER	SQL_C_SLONG
SQL_LONGVARBINARY	SQL_C_BINARY
SQL_LONGVARCHAR	SQL_C_CHAR
SQL_NUMERIC	SQL_C_NUMERIC
SQL_REAL	SQL_C_FLOAT
SQL_SMALLINT	SQL_C_SSHORT
SQL_TIMESTAMP	SQL_C_TIMESTAMP
SQL_VARCHAR	SQL_C_CHARS

**GBase 8s 的附加缺省 C 数据类型**

对于每一附加的 GBase 8s ODBC Driver SQL 数据类型，有缺省 C 数据类型。

对于 GBase 8s 的每一附加的 GBase 8s ODBC Driver SQL 数据类型，下表展示缺省的 C 数据类型。

GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	缺省的 GBase 8s ODBC 驱动程序 C 数据类型 (fCType)
SQL_IFMX_UDT_BLOB	SQL_C_BINARY
SQL_IFMX_UDT_CLOB	SQL_C_BINARY

GBase 8s ODBC 驱动程序 SQL 数据类型 (fSqlType)	缺省的 GBase 8s ODBC 驱动程序 C 数据类型 (fCType)
SQL_INFX_UDT_FIXED	此 GBase 8s ODBC Driver SQL 数据类型没有缺省的 GBase 8s ODBC Driver C 数据类型。由于此 GBase 8s ODBC 驱动程序 SQL 数据类型可包含二进制数据或字符数据, 因此, 在访存对应的值之前, 您必须为此 GBase 8s ODBC 驱动程序 SQL 数据类型绑定一个变量。绑定的变量的数据类型为该值指定 C 数据类型。
SQL_INFX_UDT_VARYING	此 GBase 8s ODBC Driver SQL 数据类型没有缺省的 GBase 8s ODBC Driver C 数据类型。因为此 GBase 8s ODBC Driver SQL 数据类型包含二进制数据或字符数据, 因此在访存对应的值之前, 您必须为此 GBase 8s ODBC Driver SQL 数据类型绑定一个变量。绑定的变量的数据类型为该值指定 C 数据类型。

SQL 至 C: 二进制

二进制 GBase 8s ODBC Driver SQL 数据类型是 SQL\_LONGVARBINARY。

下表展示可将二进制 SQL 数据转换成的 GBase 8s ODBC Driver C 数据类型。

fCType	测试	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	数据的长度 $\leq cbValueMax$	数据	数据的长度	不适用
	数据的长度 $> cbValueMax$	截断的数据	数据的长度	01004
SQL_C_CHAR	(数据的长度) * 2 $< cbValueMax$	数据	数据的长度	不适用
	(数据的长度) * 2 $\geq cbValueMax$	截断的数据	数据的长度	01004

当 GBase 8s ODBC Driver 将二进制 SQL 数据转换为字符 C 数据时, 将源数据的每一字节 (8 位) 表示为两个 ASCII 字符。这些字符是以十六进制形式表示的数值的 ASCII 字符。例如, GBase 8s ODBC Driver 将二进制 00000001 转换为 "01", 将二进制 11111111 转换为 "FF"。

GBase 8s ODBC Driver将单独的字节转换为十六进制数字对, 并以空字节终止该字符串。由于此惯例, 如果 *cbValueMax* 是偶数, 并小于被转换的数据的长度, 则不使用 *rgbValue* 缓冲区的最后一个字节。(转换的数据需要偶数字节, 邻近最后一个字节的是空字节, 且不可使用最后的字节。)

SQL 至 C: 布尔

布尔 GBase 8s ODBC Driver SQL 数据类型为 SQL\_BIT。

下表展示可将布尔 SQL 数据转换为哪些 GBase 8s ODBC Driver C 数据类型。当 GBase 8s ODBC Driver 将布尔 SQL 数据转换为字符 C 数据时, 可能的值为 0 和 1。

可将布尔 SQL 数据转换成的 GBase 8s ODBC Driver C 数据类型

fCType	测试	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	$cbValueMax \leq 1$	数据	1	不适用
	$cbValueMax < 1$	未受影响	未受影响	22003
SQL_C_BIT	对于此转换， GBase 8s ODBC Driver 忽略 $cbValueMax$ 的值。对于该 C 数据类型的大小， GBase 8s ODBC Driver 使用 $rgbValue$ 的大小。	数据	1 (这是对应的 C 数据类型的大小。)	不适用
SQL_C_CHAR	$cbValueMax > 1$	数据	1	不适用
	$cbValueMax \leq 1$	未受影响	未受影响	22003

### SQL 至 C: 字符

字符 GBase 8s ODBC Driver SQL 数据类型为 SQL\_CHAR、SQL\_LONGVARCHAR 和 SQL\_VARCHAR。

下表展示可将字符 SQL 数据转换为哪些 GBase 8s ODBC Driver C 数据类型。当 GBase 8s ODBC Driver 将字符 SQL 数据转换为数值、日期或时间戳 C 数据时，它忽略开头和结尾的空格。

可将字符 SQL 数据转换成的 GBase 8s ODBC Driver C 数据类型。

fCType	测试	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	数据的长度 $\leq cbValueMax$ 。	数据	数据的长度	不适用
	数据的长度 $> cbValueMax$ 。	截断的数据	数据的长度	01004
SQL_C_BIT	数据为 0 或 1。	数据	1	不适用
	数据大于 0，小于 2，且不等于 1。	截断的数据	1	01004
	数据小于 0 或大于或等于 2。	未受影响	未受影响	22003
	数据不是数值-文字。	未受影响	未受影响 (对应的 C 数据类型的大小为 1。)	22005
SQL_C_CHAR	数据的长度 $< cbValueMax$ 。	数据	数据的长度	不适用
	数据的长度 $\geq cbValueMax$ 。	截断的数据	数据的长度	01004

fCType	测试	rgbValue	pcbValue	SQLSTATE
SQL_C_DATE	数据值是有效的日期值。	数据	6	不适用
	数据值是有效的 <i>时间戳值</i> ；时间部分为零。	数据	6	不适用
	数据值是有效的 <i>时间戳值</i> ；时间部分非零。 (GBase 8s ODBC Driver 忽略 <i>时间戳值</i> 的日期部分。)	截断的数据	6	01004
	数据值不是有效的 <i>日期值</i> 或 <i>时间戳值</i> 。 (对于所有这些转换，GBase 8s ODBC Driver 忽略 <i>cbValueMax</i> 的值。对于该 C 数据类型的大小，GBase 8s ODBC Driver 使用 <i>rgbValue</i> 的大小。)	未受影响	未受影响 (对应的 C 数据类型的大小是 6。)	22008
SQL_C_DOUBLE SQL_C_FLOAT	数据在正在将数值转换为 其的数据类型的范围之内。	数据	C 数据类型的大小	不适用
	数据在正在将数值转换为 其的数据类型的范围之外。	未受影响	未受影响	22003
	数据不是数值-文字。 (对于所有这些转换，GBase 8s ODBC Driver 忽略 <i>cbValueMax</i> 的值。对于该 C 数据类型的大小，GBase 8s ODBC Driver 使用 <i>rgbValue</i> 的大小。)	未受影响	未受影响	22005
SQL_C_LONG SQL_C_SHORT SQL_C_SLONG SQL_C_SSHORT SQL_C_STINYINT SQL_C_TINYINT SQL_C_ULONG SQL_C_USHORT SQL_C_UTINYINT	无截断的数据转换。	数据	C 数据类型的大小	不适用
	截断小数位的转换的数据。	截断的数据	C 数据类型的大小	01004
	数据的转换会导致整个数字丢失(不同于小数位)。	未受影响	未受影响	22003
	数据不是数值-文字。 (对于所有这些转换，GBase 8s ODBC Driver 忽略 <i>cbValueMax</i> 的值。对于 C 数据类型的大小，GBase 8s ODBC Driver 使用 <i>rgbValue</i> 的大小。)	未受影响	未受影响	22005
SQL_C_TIMESTAMP	数据值是有效的 <i>时间戳值</i> ；不截断小数秒部分。	数据	16	不适用

fCType	测试	rgbValue	pcbValue	SQLSTATE
	数据值是有效的时间戳值；截断小数秒部分。	截断的数据	16	不适用
	数据值是有效的日期值。	数据（GBase 8s ODBC Driver将时间戳结构的时间域设置为零。）	16	不适用
	数据值是有效的时间值。	数据（GBase 8s ODBC Driver将时间戳结构的日期域设置为当前日期。）	16	不适用
	数据值不是有效的日期值、时间值或时间戳值。 （对于所有这些转换，GBase 8s ODBC Driver 忽略 cbValueMax的值。对于该 C 数据类型的大小，GBase 8s ODBC Driver使用 rgbValue 的大小。）	未受影响	未受影响（对应的 C 数据类型的大小为 16。）	22008

SQL 至 C: 日期

日期 GBase 8s ODBC Driver SQL 数据类型为 SQL\_DATE。

下表展示可将日期 SQL 数据转换为哪些 GBase 8s ODBC Driver C 数据类型。当 GBase 8s ODBC Driver 将日期 SQL 数据转换为字符 C 数据时，结果字符串的格式为 yyyy-mm-dd。

可将日期 SQL 数据转换成的 GBase 8s ODBC Driver C 数据类型。

fCType	测试	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	数据的长度 ≤ <i>cbValueMax</i>	数据	数据的长度	不适用
	数据的长度 > <i>cbValueMax</i>	未受影响	未受影响	22003
SQL_C_CHAR	<i>cbValueMax</i> ≥ 11	数据	10	不适用
	<i>cbValueMax</i> < 11	未受影响	未受影响	22003



fCType	测试	rgbValue	pcbValue	SQLSTATE
SQL_C_DATE	对于此转换，GBase 8s ODBC Driver 忽略 <i>cbValueMax</i> 的值。对于该 C 数据类型的大小，GBase 8s ODBC Driver 使用 <i>rgbValue</i> 的大小。	数据	6 (这是对应的 C 数据类型的大小。)	不适用
SQL_C_TIMESTAMP	对于此转换，GBase 8s ODBC Driver 忽略 <i>cbValueMax</i> 的值。对于该 C 数据类型的大小，GBase 8s ODBC Driver 使用 <i>rgbValue</i> 的大小。	数据 (GBase 8s ODBC Driver 将时间戳结构的时间域设置为零。)	16 (这是对应的 C 数据类型的大小。)	不适用

### SQL 至 C: 数值

数值 GBase 8s ODBC Driver SQL 数据类型是 SQL\_DECIMAL、SQL\_DOUBLE、SQL\_INTEGER、SQL\_REAL 和 SQL\_SMALLINT

下表展示可将数值 SQL 数据转换为哪些 GBase 8s ODBC Driver C 数据类型。

可将数值 SQL 数据转换成的 GBase 8s ODBC Driver C 数据类型。

fCType	测试	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	数据的长度 $\leq$ <i>cbValueMax</i> 。	数据	数据的长度	不适用
	数据的长度 $>$ <i>cbValueMax</i> 。	未受影响	未受影响	22003
SQL_C_BIT	数据为 0 或 1。	数据	1	不适用
	数据大于 0，小于 2，且不等于 1。	截断的数据	1	01004
	数据小于 0 或大于或等于 2。	未受影响	未受影响	22003
	数据不是数值-文字。	未受影响	未受影响 (对应的 C 数据类型的大小为 1。)	22005
SQL_C_CHAR	显示大小 $<$ <i>cbValueMax</i>	数据	数据的长度	不适用
	所有位数(相对于小数位) $<$ <i>cbValueMax</i> 。	截断的数据	数据的长度	01004
	所有位数(相对于小数位) $\geq$ <i>cbValueMax</i> 。	未受影响	未受影响	22003
SQL_C_DOUBLE	数据在正在将数值转换至	数据	C 数据类型	不适用



fCType	测试	rgbValue	pcbValue	SQLSTATE
SQL_C_FLOAT	的数据类型的范围之内。		的大小	
	数据在正在将数值转换至的数据类型的范围之外。 (对于此转换, GBase 8s ODBC Driver 忽略 <i>cbValueMax</i> 的值。对于该 C 数据类型的大小, GBase 8s ODBC Driver 使用 <i>rgbValue</i> 的大小。)	未受影响	未受影响	22003
SQL_C_LONG	未截断的转换的数据。	数据	C 数据类型的大小	不适用
SQL_C_SHORT				
SQL_C_SLONG	截断小数位的转换的数据。	截断的数据	C 数据类型的大小	01004
SQL_C_SSHORT				
SQL_C_STINYINT	数据的转换会导致所有位的丢失(相对于小数位)。 (对于此转换, GBase 8s ODBC Driver 忽略 <i>cbValueMax</i> 的值。对于该 C 数据类型的大小, GBase 8s ODBC Driver 使用 <i>rgbValue</i> 的大小。)	未受影响	未受影响	22003
SQL_C_TINYINT				
SQL_C_ULONG				
SQL_C_USHORT				
SQL_C_UTINYINT				

**SQL 至 C: 时间戳**

时间戳 GBase 8s ODBC Driver SQL 数据类型为 SQL\_TIMESTAMP。

下表展示可将时间戳 SQL 数据转换为哪些 GBase 8s ODBC Driver C 数据类型。

可将时间戳 SQL 数据转换成的 GBase 8s ODBC Driver C 数据类型。

fCType	测试	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	数据的长度 $\leq$ <i>cbValueMax</i> 。	数据	数据的长度	不适用
	数据的长度 $>$ <i>cbValueMax</i> 。	未受影响	未受影响	22003
SQL_C_CHAR	<i>cbValueMax</i> $>$ 显示大小。	数据	数据的长度	不适用
	$20 \leq cbValueMax \leq$ 显示大小。	截断的数据 (GBase 8s ODBC Driver 截断时间戳的小数秒部分。)	数据的长度	01004

fCType	测试	rgbValue	pcbValue	SQLSTATE
	<i>cbValueMax</i> < 20。	未受影响	未受影响	22003
SQL_C_DATE	时间戳的时间部分为零。	数据	6	不适用
	时间戳的时间部分非零。 (对于此转换, GBase 8s ODBC Driver 忽略 <i>cbValueMax</i> 的值。对于该 C 数据类型的大小, GBase 8s ODBC Driver 使用 <i>rgbValue</i> 的大小。)	截断的数据 (GBase 8s ODBC Driver 截断时间戳的时间部分。)	6 (对应的 C 数据类型的大小为 6。)	01004
SQL_C_TIMESTAMP	不截断时间戳的小数秒部分。	数据	16	不适用
	截断时间戳的小数秒部分。 (对于此转换, GBase 8s ODBC Driver 忽略 <i>cbValueMax</i> 的值。对于该 C 数据类型的大小, GBase 8s ODBC Driver 使用 <i>rgbValue</i> 的大小。)	截断的数据 (GBase 8s ODBC Driver 截断时间戳的小数秒部分。)	16 (对应的 C 数据类型的大小为 16。)	01004

当 GBase 8s ODBC Driver 将时间戳 SQL 数据转换为字符 C 数据时, 生成的字符串采用 yyyy-mm-dd hh:mm:ss[f...] 的格式, 在此, 小数秒最多可使用九位。除了小数点和小数秒以外, 不论时间戳 SQL 数据类型的精度, 都必须使用整个格式, 。

SQL 至 C 数据转换示例

这些示例展示 GBase 8s ODBC Driver 如何将 SQL 数据转换为 C 数据。

下表说明 GBase 8s ODBC Driver 如何将 SQL 数据转换为 C 数据。" " 表示空终止字节 (当该 C 数据类型为 SQL\_C\_WCHAR 时, " " 表示一个宽的空终止字符)。GBase 8s ODBC Driver 始终以空字符来终止 SQL\_C\_CHAR 和 SQL\_C\_WCHAR 数据。对于 SQL\_DATE 与 SQL\_C\_TIMESTAMP 的组合, GBase 8s ODBC Driver 将 *rgbValue* 列中的数值存储在 TIMESTAMP\_STRUCT 结构的字段中 。

展示 GBase 8s ODBC Driver 如何将 SQL 数据转换为 C 数据的六列表。

SQL 数据类型	SQL 数据值	C 数据类型	cbValue Max	rgbValue	SQLSTATE
SQL_CHAR	tigers	SQL_C_CHAR	7	tigers�	不适用
SQL_CHAR	tigers	SQL_C_CHAR	6	tiger�	01004
SQL_CHAR	tigers	SQL_C_WCHAR	14	tigers�	不适用
SQL_CHAR	tigers	SQL_C_WCHAR	12	tiger�	01004

SQL 数据类型	SQL 数据值	C 数据类型	cbValue Max	rgbValue	SQLSTATE
SQL_DECIMAL	1234.56	SQL_C_CHAR	8	1234.56\0	不适用
SQL_DECIMAL	1234.56	SQL_C_CHAR	5	1234\0	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	4	—	22003
SQL_DECIMAL	1234.56	SQL_C_WCHAR	16	1234.56\0	不适用
SQL_DECIMAL	1234.56	SQL_C_WCHAR	10	1234\0	01004
SQL_DECIMAL	1234.56	SQL_C_WCHAR	8	—	220023
SQL_DECIMAL	1234.56	SQL_C_FLOAT	忽略的	1234.56	不适用
SQL_DECIMAL	1234.56	SQL_C_SSHORT	忽略的	1234	01004
SQL_DECIMAL	1234.56	SQL_C_STINYINT	忽略的	—	22003
SQL_DOUBLE	1.2345678	SQL_C_DOUBLE	忽略的	1.234567	不适用
SQL_DOUBLE	1.2345678	SQL_C_FLOAT	忽略的	1.234567	不适用
SQL_DOUBLE	1.2345678	SQL_C_STINYINT	忽略的	1	不适用
SQL_DATE	1992-12-31	SQL_C_CHAR	11	1992-12-31\0	不适用
SQL_DATE	1992-12-31	SQL_C_CHAR	10	—	22003
SQL_DATE	1992-12-31	SQL_C_WCHAR	22	1992-12-31\0	不适用
SQL_DATE	1992-12-31	SQL_C_WCHAR	20	—	22003
SQL_DATE	1992-12-31	SQL_C_TIMESTAMP	忽略的	1992,12,31, 0,0,0,0	不适用
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	23	1992-12-31 23:45:55.12\0	不适用
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	22	1992-12-31 23:45:55.1\0	01004
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	18	—	22003
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_WCHAR	46	1992-12-31 23:45:55.12\0	不适用
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_WCHAR	44	1992-12-31 23:45:55.1\0	01004
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_WCHAR	36	—	22003

**重要：** 宽字符（wchar\_t）的大小与平台有关。前面的示例适用于 Windows™，在此，宽字符的大小为 2 字节。在大多数 UNIX™ 平台上，宽字符为 4 字节。在低于 AIX5L 版本的 GBase AIX 上，它是 2 字节。

#### 4.5.5 由 C 转换为 SQL

当调用 `SQLExecute` 或 `SQLExecDirect` 时, GBase 8s ODBC Driver 从应用程序中的存储位置检索使用 `SQLBindParameter` 绑定的参数的数据。

对于处于执行的数据参数, 请调用 `SQLPutData` 来发送参数数据。如有必要, GBase 8s ODBC Driver 将数据由 `SQLBindParameter` 中 `ValueType` 参数指定的数据类型转换为 `SQLBindParameter` 中 `fSqlType` 参数指定的数据类型。最终, GBase 8s ODBC Driver 将数据发送至数据源。

如果在 `SQLBindParameter` 中指定的 `rgbValue` 和 `pcbValue` 参数都是空指针, 则该函数返回 `SQLSTATE S1009` (无效的参数值)。要指定空 SQL 数据值, 请将 `SQLBindParameter` 的 `pcbValue`参数指向的值, 或将 `cbValue` 参数的值设置为 `SQL_NULL_DATA`。要指定 `rgbValue` 中的值为空终止字符串, 请将这些值设置为 `SQL_NTS`。

在表中使用下列术语:

- 数据的长度
- 无论数据在发送到应用程序之前是否被截断, 可用于发送至数据源的 SQL 数据的字节数。对于字符串数据, 这不包括空终止字节。
- 列长度和显示大小
- 定义了精度、小数位数、长度和显示大小 中的每个 SQL 数据。
- 位数
- 表示数值的字符数, 包括负号、小数点和指数 (如果需要的话)。
- 斜体词语
- 表示 GBase 8s ODBC Driver SQL 语法的元素。

C 至 SQL: 二进制

二进制 GBase 8s ODBC Driver C 数据类型为 `SQL_C_BINARY`。

下表展示可将二进制 C 数据转换为哪些 GBase 8s ODBC Driver SQL 数据类型。在“测试”列中, SQL 数据长度是在数据源上存储该数据需要的字节数。此长度可能不同于列长度, 如同 精度、小数位数、长度和显示大小 中定义的那样。

可将 C 数据转换为哪些 GBase 8s ODBC Driver SQL 数据类型。

fSqlType	测试	SQLSTATE
SQL_BIGINT	数据的长度 = SQL 数据长度。	不适用
	数据的长度 ≠ SQL 数据长度。	22003
SQL_BIT	数据的长度 = SQL 数据长度。	不适用
	数据的长度 ≠ SQL 数据长度。	22003
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	数据的长度 ≠ 列长度。	不适用
	数据的长度 > 列长度。	01004
SQL_DATE SQL_TIMESTAMP	数据的长度 = SQL 数据长度。	不适用
	数据的长度 ≠ SQL 数据长度。	22003
SQL_DECIMAL	数据的长度 = SQL 数据长度。	不适用

fSqlType	测试	SQLSTATE
SQL_DOUBLE SQL_INTEGER SQL_REAL SQL_SMALLINT	数据的长度 ≠ SQL 数据长度。	22003
SQL_LONGVARBINARY	数据的长度 ≠ 列长度。	不适用
	数据的长度 > 列长度。	01004

C 至 SQL：位

位 GBase 8s ODBC Driver C 数据类型为 **SQL\_C\_BIT**。

下表展示可将位 C 数据转换为哪些 GBase 8s ODBC Driver SQL 数据类型。

fSqlType	测试	SQLSTATE
SQL_BIGINT SQL_DECIMAL SQL_DOUBLE SQL_INTEGER SQL_REAL SQL_SMALLINT	无	不适用
SQL_BIT	无	不适用
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	无	不适用

GBase 8s ODBC Driver 会忽略 SQLBindParameter 的 *pcbValue* 参数指向的值，以及从布尔 C 数据类型转换数据时 SQLPutData 的 *cbValue* 参数的值。GBase 8s ODBC Driver 使用 *rgbValue* 的大小来表示布尔 C 数据类型的大小。

C 至 SQL：字符

字符 GBase 8s ODBC Driver C 数据类型为 **SQL\_C\_CHAR**。

下表展示可将 C 字符数据转换为哪些 GBase 8s ODBC Driver SQL 数据类型。

可将 C 字符数据转换成的 GBase 8s ODBC Driver SQL 数据类型。

fSqlType	测试	SQLSTATE
SQL_BIGINT	无截断转换的数据。	不适用
	截断小数位的转换的数据。	01004
	数据转换会导致所有位的丢失（与小数位相反）。	22003
	数据值不是数值-文字。	22005

fSqlType	测试	SQLSTATE
SQL_BIT	数据为 0 或 1。	不适用
	数据大于 0，小于 2，且不等于 1。	01004
	数据小于 0，或大于或等于 2。	22003
	数据不是数值-文字。	22005
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	数据的长度 $\leq$ 列长度。	不适用
	数据的长度 $>$ 列长度。	01004
SQL_DATE	数据值是有效的 GBase 8s ODBC 驱动程序日期-文字。	不适用
	数据值是有效的 GBase 8s ODBC 驱动程序时间戳-文字；时间部分为零。	不适用
	数据值是有效的 GBase 8s ODBC 驱动程序时间戳-文字；时间部分非零。GBase 8s ODBC 驱动程序截断时间戳的时间部分。	01004
	数据值不是有效的 GBase 8s ODBC 驱动程序日期-文字 或 GBase 8s ODBC 驱动程序时间戳-文字。	22008
SQL_DECIMAL SQL_INTEGER SQL_SMALLINT	未经截断转换的数据。	不适用
	截断小数位转换的数据。	01004
	数据的转换会导致所有位的丢失（与小数相反）。	22003
	数据值不是数值-文字。	22005
SQL_DOUBLE SQL_REAL	数据在正在将数值转换为其的数据类型的范围之内。	不适用
	数据在正在将数值转换为其的数据类型的范围之外。	22003
	数据值不是数值-文字。	22005
SQL_LONGVARBINARY	(数据的长度) / 2 $\leq$ 列长度。	不适用
	(数据的长度) / 2 $>$ 列长度。	01004
	数据值不是十六进制值。	22005
SQL_TIMESTAMP	数据值是有效的 GBase 8s ODBC 驱动程序时间戳-文字；不截断小数秒部分。	不适用
	数据值是有效的 GBase 8s ODBC 驱动程序时间戳-文字；截断小数秒部分。	01004

fSqlType	测试	SQLSTATE
	数据值是有效的 GBase 8s ODBC 驱动程序 <i>日期-文字</i> 。GBase 8s ODBC 驱动程序将时间戳的时间部分设置为零。	不适用
	数据值是有效的 GBase 8s ODBC 驱动程序 <i>时间-文字</i> 。GBase 8s ODBC 驱动程序将时间戳的日期部分设置为当前日期。	不适用
	数据值不是有效的 GBase 8s ODBC 驱动程序 <i>日期-文字</i> 、GBase 8s ODBC 驱动程序 <i>时间-文字</i> 或 GBase 8s ODBC 驱动程序 <i>时间戳-文字</i> 。	22008

当 GBase 8s ODBC Driver 将字符 C 时间转换为数值、日期或时间戳 SQL 数据时，它忽略开头和结尾的空格。当 GBase 8s ODBC Driver 将字符 C 数据转换为二进制 SQL 数据时，它将每一两字节字符数据转换为一字节二进制数据。每一两字节字符数据表示一个十六进制形式的数值。例如，GBase 8s ODBC Driver 将 "01" 转换为二进制 00000001，将 "FF" 转换为二进制 11111111。

GBase 8s ODBC Driver 始终将十六进制数字对转换为单独的字节，并忽略空终止字节。由于此转换，如果字符串的长度是奇数，则不转换该字符串的最后一个字节（不包括空终止字节，如果有的话）。

C 至 SQL: 日期

日期 GBase 8s ODBC Driver C 数据类型为 **SQL\_C\_DATE**。

下表展示可将日期 C 数据转换为哪些 GBase 8s ODBC Driver SQL 数据类型。

可将日期 C 数据转换成的 GBase 8s ODBC Driver SQL 数据类型。

fSqlType	测试	SQLSTATE
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	列长度 ≥ 10。	不适用
	列长度 < 10。	22003
	数据值不是有效的日期。	22008
SQL_DATE	数据值是有效的日期。	不适用
	数据值不是有效的日期。	22008
SQL_TIMESTAMP	数据值是有效的日期。GBase 8s ODBC 驱动程序将时间戳的时间部分设置为零。	不适用
	数据值不是有效的日期。	22008

当 GBase 8s ODBC Driver 将日期 C 数据转换为字符 SQL 数据时，生成的字符串采用 yyyy-mm-dd 的格式。



当它将数据由日期 C 数据类型转换时,GBase 8s ODBC Driver 忽略 SQLBindParameter 的 *pcbValue* 参数指向的值,以及 SQLPutData 的 *cbValue* 参数的值。对于该日期 C 数据类型的大小,GBase 8s ODBC Driver 使用 *rgbValue* 的大小。

C 至 SQL: 数值

总共有十种 GBase 8s ODBC Driver C 数据类型。

数值 GBase 8s ODBC Driver C 数据类型是:

- SQL\_C\_DOUBLE
- SQL\_C\_FLOAT
- SQL\_C\_LONG
- SQL\_C\_SHORT
- SQL\_C\_SLONG
- SQL\_C\_STINYINT
- SQL\_C\_TINYINT
- SQL\_C\_ULONG
- SQL\_C\_USHORT
- SQL\_C\_UTINYINT

下表展示可将数值 C 数据转换为哪些 GBase 8s ODBC Driver SQL 数据类型。

可将数值 C 数据转换成的 GBase 8s ODBC Driver SQL 数据类型。

fSqlType	测试	SQLSTATE
SQL_BIGINT	未截断的转换的数据。	不适用
	截断小数位的转换的数据。	01004
	数据的转换会导致所有位的丢失（与小数位相反）。	22003
SQL_BIT	数据为 0 或 1。	不适用
	数据大于 0, 小于 2, 且不等于 1。	01004
	数据小于 0 或大于或等于 2。	22003
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	位数 ≤ 列长度。	不适用
	所有位数（与小数位相反）≤ 列长度。	01004
	所有位数（与小数位相反）> 列长度。	22003
SQL_DECIMAL SQL_INTEGER SQL_SMALLINT	未截断的转换的数据	不适用
	截断小数位的转换的数据。	01004
	数据的转换会导致所有位的丢失（与小数位相反）。	22003



fSqlType	测试	SQLSTATE
SQL_DOUBLE SQL_REAL	数据在正在将数值转换为其的数据类型的范围之内。	不适用
	数据在正在将数值转换为其的数据类型的范围之外。	22003

当它从数值 C 数据类型转换数据时，GBase 8s ODBC Driver 忽略 SQLBindParameter 的 *pcbValue* 参数指向的值，以及 SQLPutData 的 *cbValue* 参数的值。对于数值 C 数据类型的大小，GBase 8s ODBC Driver 使用 *rgbValue* 的大小。

C 至 SQL：时间戳

时间戳 GBase 8s ODBC Driver C 数据类型为 SQL\_C\_TIMESTAMP。

下表展示可将时间戳 C 数据转换为哪些 GBase 8s ODBC Driver SQL 数据类型。

可将时间戳 C 数据转换成的 GBase 8s ODBC Driver SQL 数据类型。

fSqlType	测试	SQLSTATE
SQL_CHAR SQL_LONGVARCHAR SQL_VARCHAR	列长度 ≥ 显示大小。	不适用
	19 ≤ 列长度 < 显示大小。 GBase 8s ODBC Driver 截断时间戳的小数秒。	01004
	列长度 < 19。	22003
	数据值不是有效的日期。	22008
SQL_DATE	时刻域为零。	不适用
	时刻域非零。 GBase 8s ODBC Driver 截断时间戳结构的时刻域。	01004
	数据值不包含有效的日期。	22008
SQL_TIMESTAMP	不截断小数秒域。	不适用
	截断小数秒域。 GBase 8s ODBC Driver 截断时间戳结构的小数秒域。	01004
	数据值不是有效的时间戳。	22008

当 GBase 8s ODBC Driver 将时间戳 C 数据转换为字符 SQL 数据时，生成的字符串采用 yyyy-mm-dd hh:mm:ss[f...] 的格式。

当它由时间戳 C 数据类型转换数据时, GBase 8s ODBC Driver 忽略 SQLBindParameter 的 *pcbValue* 参数指向的值, 以及 SQLPutData 的 *cbValue* 参数的值。对于该时间戳 C 数据类型的大小, GBase 8s ODBC Driver 使用 *rgbValue* 的大小。

C 至 SQL 数据转换示例

这些示例展示 GBase 8s ODBC Driver 如何将 C 数据转换为 SQL 数据。

下表说明 GBase 8s ODBC Driver 如何将 C 数据转换为 SQL 数据。“\0”表示一个空终止字节。仅当该数据的长度为 SQL\_NTS 时, 才需要空终止字节。对于 SQL\_C\_DATE, 在“C 数据值”列中的数值是存储在 DATE\_STRUCT 结构的字段中的数值。对于 SQL\_C\_TIMESTAMP, 在“C 数据值”列中的数值是存储在 TIMESTAMP\_STRUCT 结构的字段中的数值。

C 数据类型	C 数据值	SQL 数据类型	列长度	SQL 数据值	SQLSTATE
SQL_C_CHAR	tigers\0	SQL_CHAR	6	tigers	不适用
SQL_C_CHAR	tigers\0	SQL_CHAR	5	tiger	01004
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	8 (除了数值的字节之外, 还需要一个符号字节和另一个小数点字节。)	1234.56	不适用
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	7 (除了数值的字节之外, 还需要一个符号字节和另一个小数点字节。)	1234.5	01004
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	4	—	22003
SQL_C_FLOAT	1234.56	SQL_FLOAT	不适用	1234.56	不适用
SQL_C_FLOAT	1234.56	SQL_INTEGER	不适用	1234	01004
SQL_C_FLOAT	1234.56	SQL_TINYINT	不适用	—	22003
SQL_C_DATE	1992,12,31	SQL_CHAR	10	1992-12-31	不适用

C 数据类型	C 数据值	SQL 数据类型	列长度	SQL 数据值	SQLSTATE
SQL_C_DATE	1992,12,31	SQL_CHAR	9	—	22003
SQL_C_DATE	1992,12,31	SQL_TIMESTAMP	不适用	1992-12-31 00:00:00.0	不适用
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000	SQL_CHAR	22	1992-12-31 23:45:55.12	不适用
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000	SQL_CHAR	21	1992-12-31 23:45:55.1	01004
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000	SQL_CHAR	18	—	22003

## 5 智能大对象

这些主题描述如何存储、创建和访问智能大对象；如何转换智能大对象数据；如何检索智能大对象的状态；以及如何将智能大对象写至文件或从文件读取。

仅当您的数据库服务器是 GBase 8s 时，这些主题中的信息才适用。

智能大对象是存储在磁盘上 `sbspace` 中的可恢复的大对象。您可以读、写和查找操作来访问智能大对象，类似于操作系统文件。两种智能大对象的数据类型为 **字符大对象 (CLOB)** 和 **二进制大对象 (BLOB)**。**CLOB** 由文本数据组成，**BLOB** 由不可分开的字节流中的二进制数据组成。

要获取关于智能大对象数据类型的更多信息，请参阅《GBase 8s SQL 指南：参考》。

### 5.1 智能大对象的数据结构

由于智能大对象可能很巨大，因此，GBase 8s 有两种存储智能大对象内容的选择。

因此，GBase 8s 不是将智能大对象存储在数据库表中，而是 进行如下处理：

- 在 sbspace 中存储智能大对象的内容
- 在数据库表中存储指向智能大对象的指针

由于智能大对象可很巨大，因此，GBase 8s ODBC Driver应用程序无法在变量中接收智能大对象。相反地，应用程序在一个数据结构中发送和接收关于智能大对象的信息。下表描述 GBase 8s ODBC Driver 用于智能大对象的数据结构。

数据结构	名称	描述
lofd	智能大对象文件描述符	提供对智能大对象的访问。如果它在操作系统文件中，则请使用文件描述符来访问智能大对象数据。
loptr	智能大对象指针结构	提供安全信息和执行智能大对象的指针。此结构是数据库服务器在数据库表中存储智能大对象的数据。因此，诸如 INSERT 和 SELECT 这样的 SQL 语句接受智能大对象指针作为有智能大对象数据类型的列或参数的值。
lospec	智能大对象规范结构	指定智能大对象的存储特征。
lostat	智能大对象状态结构	存储智能大对象的状态信息。通常，可以二进制或字符表示来访问用户定义的数据类型（UDT）。然而，无法将智能大对象状态结构转换为字符表示。因此，对于 lostat，您需要使用 SQL_C_BINARY 作为 GBase 8s ODBC Driver C 数据类型。

**限制：** 对于 GBase 8s ODBC Driver应用程序，这些数据结构是不透明的，且它们的内部结构可能更改。因此，请不要直接访问这些内部结构。请使用智能大对象客户机函数来操纵这些数据结构。

应用程序负责为这些智能大对象数据结构分配空间。

### 5.1.1 使用智能大对象数据结构

您可使用此过程来处理智能大对象数据结构。包括一个示例。

要使用智能大对象数据结构，请：

1. 确定智能大对象结构的大小。
2. 使用固定大小的数组，或动态分配缓冲区大小至少为该数据结构大小。
3. 当完成操作时，释放数组或缓冲区空间。

下列代码示例说明这些步骤：

```
rc = SQLGetInfo(hdbc, SQL_INFX_LO_SPEC_LENGTH, &lospec_size,
               sizeof(lospec_size), NULL);
lospec_buffer = malloc(lospec_size);
:
free(lospec_buffer);
```

## 5.2 智能大对象的存储

智能大对象规范结构存储智能大对象的磁盘存储信息和创建时刻标志。

5.2.1 磁盘存储信息

磁盘存储信息帮助 GBase 8s 确定如何在磁盘上最高效地存储智能大对象。

下表描述磁盘存储信息的类型，以及对应的客户机函数。对于大多数应用程序，推荐您使用数据库服务器确定的磁盘存储信息的值。

磁盘存储信息	描述	客户机函数
估计的大小	对智能大对象最终大小的估计，以字节计。数据库服务器使用此值来确定在哪些 extent 中存储智能大对象。此值提供优化的信息。如果该值很不正确，则它不导致不正确的行为。然而，它确实意味着数据库服务器可能不一定会为智能大对象选择优化的 extent 大小。	ifx_lo_specget_estbytes() ifx_lo_specset_estbytes()
最大大小	智能大对象的最大大小，以字节计。数据库服务器不允许智能大对象增长超出此大小。	ifx_lo_specget_maxbytes() ifx_lo_specset_maxbytes()
分配 extent 大小	指定的分配 extent 大小，以 KB 计。理想情况下，分配 extent 是保存所有智能大对象数据的 chunk 中的单个 extent。 数据库服务器以分配 extent 大小的增量来执行智能大对象的存储分配。它试图分配一个分配 extent 作为 chunk 中的单个 extent。然而，如果没有足够大的单个 extent，则数据库服务器必须使用必要的多个 extent 来满足请求。	ifx_lo_specget_extsz() ifx_lo_specset_extsz()
sbspace 的名称	包含智能大对象的 sbspace 的名称。在此数据库服务器上，sbspace 最多可为 128 字符长，且必须以空来终止。	ifx_lo_specget_sbspace() ifx_lo_specset_sbspace()

5.2.2 创建时刻标志

创建时刻标志告诉 GBase 8s 为智能大对象指定什么选项。

下表描述创建时刻标志。

指示符的类型	创建时刻标志	描述
--------	--------	----

指示符的类型	创建时刻标志	描述
日志记录	LO_LOG	告诉数据库服务器在系统日志文件中日志记录智能大对象的更改。 请慎重考虑是否使用 LO_LOG 标志值。要日志记录智能大对象，数据库服务器会产生大量开销。您还必须确保系统日志文件足够大，以保存智能大对象的值。要获取更多信息，请参阅《GBase 8s 管理员指南》。
	LO_NOLOG	对于涉及与智能大对象相关联的所有操作，告诉数据库服务器关闭日志记录。
最后访问时刻	LO_KEEP_LASTACCESS_TIME	告诉数据库服务器保存智能大对象的最后访问时刻。此访问时刻是最后的读或写操作的时刻。 请慎重考虑是否使用 LO_KEEP_LASTACCESS_TIME 标志值。要维护智能大对象的最后访问时刻，数据库服务器会产生大量开销。
	LO_NOKEEP_LASTACCESS_TIME	告诉数据库服务器不维护智能大对象的最后访问时刻。

ifx\_lo\_specset\_flags()函数将创建时刻标志设置为新的值。ifx\_lo\_specget\_flags() 函数检索创建时刻标志的当前值。

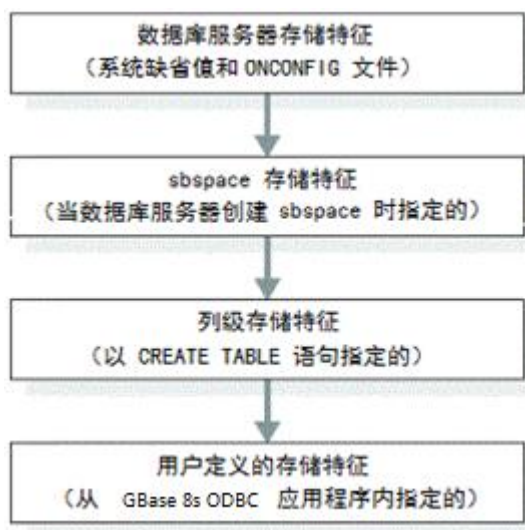
在智能大对象规范结构中存储日志记录指示符和最后访问时刻指示符，作为单个标志值。要从每一组设置标志，请使用 C 语言 OR 运算符来一起掩藏两个标志值。然而，彼此地掩藏互斥的标志会导致错误。如果您未为标志组之一指定值，则数据库服务器使用继承层级来确定此信息。

5.2.3 继承层级

GBase 8s 使用继承层级来获得存储特征。

下图展示智能大对象存储特征的继承层级。

图: 存储特征的继承层级



### 使用系统指定的存储特征

GBase 8s 使用一组存储特征作为系统指定的存储特征。

GBase 8s 使用下列存储特征组之一：

- 如果存储智能大对象的 `sbspace` 为特定的存储特征指定一个值，则数据库服务器使用该 `sbspace` 值作为系统指定的存储特征。

数据库管理员可使用 `onspaces` 实用程序来为 `sbspace` 定义存储特征。

- 如果存储智能大对象的 `sbspace` 没有为特定的存储特征指定值，则数据库服务器使用系统缺省值作为系统指定的存储特征。

数据库服务器为存储特征内部定义系统缺省值。然而，您可以 `onconfig` 文件中的 `SBSPACENAME` 配置参数来指定缺省的 `sbspace` 名称。而且，应用程序可以通过调用 `ifx_lo_col_info()` 或 `ifx_lo_specset_sbspace()` 来提供智能大对象规范结构中的目标 `sbspace`。

**重要：** 如果未指定 `SBSPACENAME` 配置参数，且智能大对象规范结构未包含目标 `sbspace` 的名称，则发生错误。

对于磁盘存储信息，推荐您使用系统指定的存储特征。要获取关于 `sbspace` 和 `onspaces` 实用程序的描述的更多信息，请参阅《GBase 8s 管理员指南》。

对于新的智能大对象，要使用系统指定的存储特征，请：

1. 调用 `ifx_lo_def_create_spec()`，来分配智能大对象规范结构，并将该结构初始化为空值。
2. 调用 `ifx_lo_create()`，来创建智能大对象的一个实例。

### 使用列级存储特征

`CREATE TABLE` 语句指定数据库列的存储特征。

`CREATE TABLE` 语句的 `PUT` 子句为智能大对象指定存储特征。`syscolattribs` 系统目录表存储列级存储特征。



对于新的智能大对象实例，要使用列级存储特征，请：

1. 调用 `ifx_lo_def_create_spec()`，来分配智能大对象规范结构，并将此结构初始化为空值。
2. 调用 `ifx_lo_col_info()`，来检索列级存储特征，并将它们存储在指定的智能大对象规范结构中。
3. 调用 `ifx_lo_create()`，来创建智能大对象的一个实例。

### 用户定义的存储特征

要指定用户定义的存储特征，请调用 `ifx_lo_specset_*` 函数。

您可为新的智能大对象定义唯一的存储特征集，如下：

- 对于将存储在列中的智能大对象，当创建智能大对象的实例时，可覆盖该列的某些存储特征。

如果您未覆盖某些或者所有这些存储特性，则智能大对象使用列级存储特征。

- 可为智能大对象指定更宽的特征级，因为智能大对象不受表列属性限制。

如果未覆盖某些或所有这些特征，则智能大对象继承系统指定的存储特征。

## 5.3 创建智能大对象的示例

代码示例 `locreate.c` 展示如何创建智能大对象。

在 UNIX™ 平台上的 `%GBS_HOME%/demo/clidemo` 中，以及在 Windows™ 环境中的 `%GBS_HOME%\demo\odbcdemo` 目录中，可找到 `locreate.c` 文件。在同一位置，您还可以找到关于如何构建 `odbc_demo` 数据库的指导。

```
/*
**      locreate.c
**
**      To create a smart large object
**
**      ODBC Functions:
**      SQLAllocHandle
**      SQLBindParameter
**      SQLConnect
**      SQLFreeStmt
**      SQLGetInfo
**      SQLDisconnect
**      SQLExecDirect
*/

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



```
#include <string.h>

#ifndef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN      12
#define ERRMSG_LEN      200

UCHAR    defDsn[] = "odbc_demo";

int checkError (SQLRETURN          rc,
                SQLSMALLINT        handleType,
                SQLHANDLE          handle,
                char                *errmsg)
{
    SQLRETURN          retcode = SQL_SUCCESS;

    SQLSMALLINT        errNum = 1;
    SQLCHAR            sqlState[6];
    SQLINTEGER         nativeError;
    SQLCHAR            errMsg[ERRMSG_LEN];
    SQLSMALLINT        textLengthPtr;

    if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
    {
        while (retcode != SQL_NO_DATA)
        {
            retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
                                     &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

            if (retcode == SQL_INVALID_HANDLE)
            {
                fprintf (stderr, "checkError function was called with an
                                invalid handle!!\n");
                return 1;
            }
        }
    }
}
```

```
        if ((retcode == SQL_SUCCESS)      || (retcode ==
SQL_SUCCESS_WITH_INFO))
            fprintf (stderr, "ERROR: %d:  %s : %s \n", nativeError,
                    sqlState, errMsg);

        errNum++;
    }

    fprintf (stderr, "%s\n", errMsg);
    return 1;  /* all errors on this handle have been reported */
}
else
    return 0;  /* no errors to report */
}

int main (long          argc,
          char          *argv[])
{
    /* Declare variables
    */

    /* Handles */
    SQLHDBC      hdbc;
    SQLHENV      henv;
    SQLHSTMT     hstmt;

    /* Smart large object file descriptor */
    long          lofd;
    long          lofd_valsize = 0;

    /* Smart large object pointer structure */
    char*         loptr_buffer;
    short         loptr_size;
    long          loptr_valsize = 0;

    /* Smart large object specification structure */
    char*         lospec_buffer;
    short         lospec_size;
    long          lospec_valsize = 0;

    /* Write buffer */
    char*         write_buffer;
    short         write_size;
```

```
long          write_valsize = 0;

/* Miscellaneous variables */
UCHAR        dsn[20];/*name of the DSN used for connecting to the
                      database*/

SQLRETURN     rc = 0;
int           in;

FILE*         hfile;
char*         lo_file_name = "advert.txt";

char          colname[BUFFER_LEN] = "item.advert";
long          colname_size = SQL_NTS;

long          mode = LO_RDWR;
long          cbMode = 0;

char*         insertStmt = "INSERT INTO item VALUES (1005, 'Helmet', 235,
                      'Each', '?', '39.95')";

/* STEP 1. Get data source name from command line (or use default).
**      Allocate environment handle and set ODBC version.
**      Allocate connection handle.
**      Establish the database connection.
**      Allocate the statement handle.
*/

/* If (dsn is not explicitly passed in as arg) */
if (argc != 2)
{
    /* Use default dsn - odbc_demo */
    fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
    strcpy ((char *)dsn, (char *)defDsn);
}
else
{
    /* Use specified dsn */
    strcpy ((char *)dsn, (char *)argv[1]);
    fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
}

/* Allocate the Environment handle */
```

```
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
    fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
    return (1);
}

/* Set the ODBC version to 3.5 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
    (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
    SQLSetEnvAttr failed\nExiting!!"))
    return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
    Handle Allocation failed\nExiting!!"))
    return (1);

/* Establish the database connection */
rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
    failed\n"))
    return (1);

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
    Handle Allocation failed\nExiting!!"))
    return (1);

fprintf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2.  Get the size of the smart large object specification
**          structure.
**          Allocate a buffer to hold the structure.
**          Create a default smart large object specification structure.
**          Reset the statement parameters.
**          */

/* Get the size of a smart large object specification structure */
```

```
rc = SQLGetInfo (hdbc, SQL_INFX_LO_SPEC_LENGTH, &lospec_size,
                sizeof(lospec_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 2 -- SQLGetInfo
                failed\n"))
    goto Exit;

/* Allocate a buffer to hold the smart large object specification
   structure*/
lospec_buffer = malloc (lospec_size);

/* Create a default smart large object specification structure */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT_OUTPUT,
SQL_C_BINARY,
                        SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
                        lospec_size, &lospec_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
                        SQLBindParameter failed\n"))
    goto Exit;
rc = SQLExecDirect (hstmt, "{call ifx_lo_def_create_spec(?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
                        SQLExecDirect failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
                        SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 2 done...default smart large object specification
                structure created\n");

/* STEP 3.  Initialise the smart large object specification structure
**          with values for the database column where the smart large
**          object is being inserted.
**          Reset the statement parameters.
*/

/* Initialise the smart large object specification structure */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR,
                        BUFFER_LEN, 0, colname, BUFFER_LEN, &colname_size);
```

```
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

lospec_valsize = lospec_size;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT,
SQL_C_BINARY,
    SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
    lospec_size, &lospec_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLBindParameter failed (param 2)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_col_info(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLExecDirect failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLFreeStm failed\n"))
    goto Exit;

fprintf(stdout, "STEP 3 done...smart large object specification
    structure initialised\n");

/* STEP 4.  Get the size of the smart large object pointer structure.
**      Allocate a buffer to hold the structure.
*/

/* Get the size of the smart large object pointer structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size,
    sizeof(loptr_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 4 --
    SQLGetInfo failed\n"))
    goto Exit;

/* Allocate a buffer to hold the smart large object pointer structure */
loptr_buffer = malloc (loptr_size);
```

```
printf (stdout, "STEP 4 done...smart large object pointer structure
allocated\n");

/* STEP 5. Create a new smart large object.
**      Reset the statement parameters.
*/

/* Create a new smart large object */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
    lospec_size, &lospec_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG,
    SQL_INTEGER, (UDWORD)0, 0, &mode, sizeof(mode), &cbMode);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
    SQLBindParameter failed (param 2)\n"))
    goto Exit;

loptr_valsize = loptr_size;

rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT_OUTPUT,
SQL_C_BINARY,
    SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
    loptr_size, &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
    SQLBindParameter failed (param 3)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 4, SQL_PARAM_OUTPUT, SQL_C_SLONG,
    SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
    SQLBindParameter failed (param 4)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_create(?, ?, ?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
    SQLExecDirect failed\n"))
    goto Exit;
```

```
/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 5 done...smart large object created\n");

/* STEP 6.  Open the file containing data for the new smart large object.
**      Allocate a buffer to hold the smart large object data.
**      Read data from the input file into the smart large object.
**      data buffer
**      Write data from the data buffer into the new smart large.
**      object.
**      Reset the statement parameters.
*/

/* Open the file containing data for the new smart large object */
hfile = open (lo_file_name, "rt");
/* sneaky way to get the size of the file */
write_size = lseek (open (lo_file_name, "rt"), 0L, SEEK_END);

/* Allocate a buffer to hold the smart large object data */
write_buffer = malloc (write_size + 1);

/* Read smart large object data from file */
read (hfile, write_buffer, write_size);

write_buffer[write_size] = '\0';
write_valsize = write_size;
/* Write data from the data buffer into the new smart large object */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
    SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR,
    (UDWORD)write_size, 0, write_buffer, write_size, &write_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLBindParameter failed (param 2)\n"))
```



```
goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_write(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLExecDirect failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 6 done...data written to new smart large
    object\n");

/* STEP 7.  Insert the new smart large object into the database.
**      Reset the statement parameters.
*/

/* Insert the new smart large object into the database */
loptr_valsize = loptr_size;

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
    loptr_size, &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
    SQLBindParameter failed\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, insertStmt, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
    SQLExecDirect failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 7 done...smart large object inserted into the
```

```
        database\n");

/* STEP 8.  Close the smart large object.
 */

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
        SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
        SQLBindParameter failed\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_close(?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
        SQLExecDirect failed\n"))
    goto Exit;

fprintf (stdout, "STEP 8 done...smart large object closed\n");

/* STEP 9.  Free the allocated buffers.
 */

free (lospec_buffer);
free (loptr_buffer);
free (write_buffer);

fprintf (stdout, "STEP 9 done...smart large object buffers freed\n");

Exit:

/* CLEANUP: Close the statement handle
**      Free the statement handle
**      Disconnect from the datasource
**      Free the connection and environment handles
**      Exit
 */

/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);
```

```
/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);

fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);
}
```

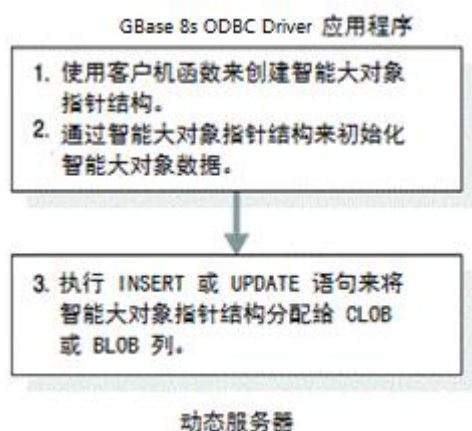
## 5.4 转移智能大对象

INSERT 或 UPDATE 语句不执行智能大对象数据的实际输入。然而，它确实为应用程序提供一种方式，来标识哪些智能大对象数据与该列相关联。

数据库表中的 BLOB 或 CLOB 列存储智能大对象的智能大对象指针结构。因此，当您存储 BLOB 或 CLOB 列时，请为 INSERT 或 UPDATE 语句的 **loptr** 变量中的列提供智能大对象指针结构。

下图展示应用程序如何将智能大对象的数据转移至数据库服务器。

图: 将智能大对象数据由客户机应用程序转移至数据库服务器



如果智能大对象指针结构存在，则智能大对象指针结构标识的智能大对象存在。当在数据库中存储智能大对象指针结构时，数据库服务器适时地重新分配智能大对象。

如果应用程序未在数据库中存储新的智能大对象的智能大对象指针结构，则当将该指针传给应用程序时，智能大对象指针结构仅对访问智能大对象的当前版本有效。如果后来更新了智能大对象，则该指针无效。当对象版本更改时，在行中存储的智能大对象指针结构不过期。

当您检索一行，然后更新该行中包含的智能大对象时，数据库服务器在该行更新智能大对象时排他锁定该行。此外，如果智能大对象花费很长时间来更新或创建，则智能大对象的长时间更新（无论是否启用日志记录，以及是否与表行相关联）会造成潜在的长事务条件。

在数据库中的 CLOB 或 BLOB 列中存储智能大对象指针结构，而不是 CLOB 或 BLOB 数据本身。因此，诸如 INSERT 和 SELECT 这样的 SQL 语句接受并返回智能大对象指针结构作为智能大对象列的列值。

## 5.5 访问智能大对象

本部分描述如何通过使用标准 ODBC API，或通过使用 ifx\_lo 函数，来选择、打开、删除、修改和关闭智能大对象。

### 5.5.1 智能大对象自动化

不以 ifx\_lo 函数来访问智能大对象，您可通过使用标准 ODBC API 来访问智能大对象。

当以标准 ODBC API 来访问智能大对象时，支持的操作包括 CLOB 和 BLOB 数据类型的选择、插入、更新和删除。不可以这种方式来访问 BYTE 和 TEXT 简单大对象。

#### 使用 SQL\_INFX\_ATTR\_LO\_AUTOMATIC 来设置访问方法

可使用 SQL\_INFX\_ATTR\_LO\_AUTOMATIC 属性来告诉数据库服务器，是通过使用 ODBC API 还是通过使用 ifx\_lo 函数，来访问智能大对象。

如果应用程序启用 SQL\_INFX\_ATTR\_LO\_AUTOMATIC 作为连接属性，则该连接的所有语句都继承该属性值。要更改每个语句的此属性值，您必须设置并重置它作为语句属性。如果为该语句启用此属性，则应用程序可通过使用如前所述的标准 ODBC 的方式来访问智能大对象。如果未为该语句启用此属性，则应用程序通过使用 ifx\_lo 函数来访问智能大对象。如果为该语句启用此属性，则应用程序不可使用 ifx\_lo 函数。

对于 GBase 8s Driver DSN，还可通过开启 ODBC Administration 的 Advanced 标签之下的报告标准 ODBC 类型选项，来启用 SQL\_INFX\_ATTR\_LO\_AUTOMATIC 属性。

CLOB 数据类型列的 SQLDescribeCol 返回 DataPtrType 的 SQL\_LONGVARCHAR。如果为该语句启用 SQL\_INFX\_ATTR\_LO\_AUTOMATIC 属性，则 BLOB 类型列的 SQLDescribeCol 返回 SQL\_LONGVARBINARY。

CLOB 类型列的 SQLColAttributes 为 SQL\_DESC\_TYPE 的 Field Identifier 返回 SQL\_LONGVARCHAR，而对于 BLOB 数据类型列，仅当为该语句启用 SQL\_INFX\_ATTR\_LO\_AUTOMATIC 属性时，它才返回 SQL\_LONGVARBINARY。

#### 通过使用 ODBC API，来插入、更新和删除智能大对象

当您插入、更新和删除 CLOB 或 BLOB 数据类型时，应用程序通过使用 SQLBindParameter 来将该数据类型与一 C 类型绑定。

当您插入、更新和删除 CLOB 数据类型时，应用程序通过使用 `SQLBindParameter`（C 类型为 `SQL_C_CHAR`，SQL 类型为 `SQL_LONGVARCHAR`）来将绑定 CLOB 数据类型。

当您插入、更新和删除 BLOB 数据类型时，应用程序通过使用 `SQLBindParameter`（C 类型为 `SQL_C_BINARY`，SQL 类型为 `SQL_LONGVARBINARY`）来绑定 BLOB 数据类型。

GBase 8s ODBC Driver 以下列方式来执行智能大对象的插入：

- 驱动程序发送请求至数据库服务器，来以新文件的形式在服务器端创建智能大对象。
- 驱动程序从数据库服务器取回此文件的文件描述符（例如，`lofd`）。
- 驱动程序将上述 `lofd` 文件和由应用程序以 `SQLBindParameter` 绑定了的智能大对象数据发送至数据库服务器。
- 数据库服务器将数据写入文件。

#### 使用 ODBC API 来选择智能大对象

当选择 CLOB 数据类型时，应用程序绑定该列的 C 类型作为 `SQL_C_CHAR`。当选择 BLOB 数据类型时，绑定该 C 类型作为 `SQL_C_BINARY`。

GBase 8s ODBC Driver 以下列方式来选择智能大对象：

- 驱动程序发送请求至数据库服务器，来打开智能大对象作为服务器端的文件。
- 驱动程序从数据库服务器取回此文件的文件描述符（例如，`lofd`）。
- 驱动程序将上述 `lofd` 和读请求发送至数据库服务器，来从该文件读取智能大对象。
- 数据库服务器通过使用上述 `lofd` 来从对应的文件读取数据，并将它发送至驱动程序。
- 驱动程序将该数据写入到应用程序使用 `SQLBindParameter` 绑定的缓冲区。

### 5.5.2 ifx\_lo 函数

这部分描述如何通过使用 `ifx_lo` 函数来选择、打开、删除、修改和关闭智能大对象。

#### 使用 ifx\_lo 函数来选择智能大对象

`SELECT` 语句不执行实际的智能大对象数据的输出。然而，它确实为应用程序建立标识智能大对象的方法，以便于应用程序可以在对智能大对象上执行操作。

下图展示数据库服务器如何将智能大对象的数据转移至应用程序。

*图：将智能大对象数据由数据库服务器转移至客户机应用程序*



使用 ifx\_lo 函数来打开智能大对象

当打开智能大对象时，您取得该智能大对象的一个智能大对象文件描述符。  
通过智能大对象文件描述符，您可以像访问操作系统文件一样访问智能大对象的数据。

访问模式

当您打开智能大对象时，请指定该数据的访问模式。访问模式确定对该打开的智能大对象，哪些读和写操作是有效的。

下表描述 ifx\_lo\_open() 和 ifx\_lo\_create() 支持的访问模式。

访问模式	用途	常量
Read only	对该数据，仅读操作是有效的。	LO_RDONLY
Dirty read	允许您读取智能大对象的未提交的数据页。在将模式设置为 LO_DIRTY_READ 之后，不可写智能大对象。当您设置此标志时，请为此智能大对象复位当前的事务隔离模式。请不要对从智能大对象以脏读模式取得的数据进行基本更新。	LO_DIRTY_READ
Write only	对该数据，仅写操作是有效的。	LO_WRONLY
Append	旨在用于 LO_WRONLY 或 LO_RDWR 。在每次写之前，立即将位置指针设置为该对象的结尾。将您写的任何数据追加至智能大对象的结尾。如果单独使用 LO_APPEND，则打开该对象仅用于读。	LO_APPEND
Read/write	对该数据，读和写操作同时有效。	LO_RDWR
Buffered access	使用标准数据库服务器缓冲池。	LO_BUFFER
Lightweight I/O	使用来自数据库服务器的会话池的私有缓冲池。	LO_NOBUFFER

当您仅以 LO\_APPEND 来打开智能大对象时，数据库服务器以 read-only 的方式打开该智能大对象。搜索操作和读操作移动该文件指针。写操作失败，且不移动该文件指针。



可以另一访问模式来掩藏 LO\_APPEND 标志。在任何这些 OR 组合中，搜索操作保持不受影响。下表展示每一 OR 组合对读和写操作的影响。

OR 操作	读操作	写操作
LO_RDONLY   LO_APPEND	在文件位置处发生，然后将文件位置移至已读取的数据的结尾。	失败，且不移动文件位置。
LO_WRONLY   LO_APPEND	失败，且不移动文件位置。	将文件位置移动至智能大对象的结尾，并写数据；在写之后，文件位置位于数据的结尾处。
LO_RDWR   LO_APPEND	在文件位置处发生，然后将文件位置移至已读取的数据的结尾。	将文件位置移至智能大对象的结尾，然后写数据；在写之后，文件位置至该数据的结尾。

轻量 I/O

当数据库服务器访问智能大对象时，对于缓冲的访问，它使用来自缓冲池的缓冲区。无缓冲的访问称为*轻量 I/O*。

轻量 I/O 使用私有缓冲区而不是缓冲池来保存智能大对象。在数据库服务器会话池之外分配这些私有缓冲区。

轻量 I/O 允许您避免最近最少使用的（LRU）队列的开销，数据库服务器使用 LRU 来管理缓冲池。要获取关于 LRU 队列的更多信息，请参阅《GBase 8s 性能指南》。

当创建或打开智能大对象时，通过将标志参数设置为 LO\_NOBUFFER 来指定轻量 I/O。要指定缓冲的访问（缺省值），请使用 LO\_BUFFER 标志。

**重要：** 当使用轻量 I/O 时，请记住下列事项：

- 结束时请使用 ifx\_lo\_close() 来关闭智能大对象，以释放分配给私有缓冲区的内存。
- 对于特定的智能大对象，使用轻量 I/O 的所有打开操作都共享相同的私有缓冲区。因此，一个操作可能导致缓冲区中的页面被刷新，而其它操作期望该对象出现存在于缓冲区中。

对于从轻量 I/O 切换为缓冲的 I/O，数据库服务器施加了以下限制：

- 如果智能大对象未打开，则可使用 ifx\_lo\_alter() 函数，来将智能大对象从轻量 I/O（LO\_NOBUFFER）切换为缓冲的 I/O（LO\_BUFFER）。然而，如果试图将使用缓冲的 I/O 的智能大对象更改为使用轻量 I/O 的智能大对象，则 ifx\_lo\_alter() 生成错误。
- 除非先使用 ifx\_lo\_alter() 将访问模式更改为缓冲的访问（LO\_BUFFER），否则，您仅可以 LO\_NOBUFFER 访问模式标志来打开使用轻量 I/O 创建的智能大对象。如果打开操作指定了 LO\_BUFFER，则数据库服务器忽略该标志。
- 只有以 read-only 模式打开对象时，才可以使用 LO\_NOBUFFER 标志打开使用缓冲存取（LO\_BUFFER）创建的智能大对象。如果试图写该对象，则数据库服务器

返回错误。要写智能大对象，您必须关闭它，然后使用 `LO_BUFFER` 标志和允许写操作的访问标志来重新打开它。

对于 `sbspace` 中的所有智能大对象，可使用数据库服务器实用程序 `onspaces` 来指定轻量 I/O。要获取关于 `onspaces` 实用程序的更多信息，请参阅《GBase 8s 管理员指南》。

## 智能大对象锁

要防止对智能大对象数据的同时访问，数据库服务器在打开智能大对象时锁定一个智能大对象。

智能大对象上的锁与行锁不同。如果从一行检索智能大对象，则数据库服务器可能持有一行锁以及一智能大对象锁。数据库服务器锁定智能大对象，因为许多列可包含同一智能大对象数据。

要指定智能大对象的锁模式，请将访问模式标志 `LO_RDONLY`、`LO_DIRTY_READ`、`LO_APPEND`、`LO_WRONLY`、`LO_RDWR` 和 `LO_TRUNC` 传至 `ifx_lo_open()` 和 `ifx_lo_create()` 函数。当指定 `LO_RDONLY` 时，数据库服务器在智能大对象数据上放置锁。当指定 `LO_DIRTY_READ` 时，数据库服务器不在智能大对象数据上放置锁。如果指定任何其他访问模式标志，则数据库服务器获取一个更新锁，在第一次写操作或其他更新操作时，它将更新锁升级为互斥锁。

持有共享锁和更新锁（在更新操作发生之前的只读模式或写模式），直到应用程序采取下列活动之一为止：

- 关闭智能大对象
- 提交该事务，或回滚它

即使关闭该智能大对象，互斥锁也会一直保留到事务结束。

**重要：** 即使该智能大对象保持打开，在事务结束时，也会失去锁。当数据库服务器检测到智能大对象没有活动的锁时，它在您下次访问该智能大对象时会放置新锁。所放置的锁是基于智能大对象的原始打开模式。

## 智能大对象上打开操作的期间

在使用 `ifx_lo_create()` 函数或 `ifx_lo_open()` 函数打开智能大对象之后，它将保持打开状态，直到发生某些事件。

智能大对象保持打开，直到这些事件发生为止：

- `ifx_lo_close()` 函数关闭智能大对象。
- 会话结束。

**重要：** 当前事务的结束不关闭智能大对象。但是，它会释放智能大对象上的任何锁。

一结束使用智能大对象，就关闭它们。让智能大对象不必要地打开着，会使用系统内存。让许多智能大对象打开着，最终会造成内存用尽的情况。

## 删除智能大对象



直到满足某些条件，才能删除智能大对象。

直到满足下列条件时，才能删除智能大对象：

- 当前事务提交。
- 关闭智能大对象，如果应用程序打开了智能大对象的话。

### 修改智能大对象

可通过使用 `UPDATE` 或 `INSERT` 语句来修改智能大对象。

要修改智能大对象的数据，请：

1. 在打开的智能大对象中读和写数据。
2. 使用 `UPDATE` 或 `INSERT` 语句来在数据库中存储智能大对象指针。

### 关闭智能大对象

在结束修改智能大对象之后，请调用 `ifx_lo_close()` 来归还分配给它的资源。

当释放资源时，可将它们重新分配给应用程序需要的其他结构。还可将智能大对象文件描述符重新分配给其他智能大对象。

### 使用 `ifx_lo` 函数从数据库检索智能大对象的示例

代码示例 `loselect.c` 展示如何从数据库检索智能大对象。

在 `UNIX™` 平台上的 `%GBS_HOME%/demo/clidemo` 目录中，以及在 `Windows™` 平台上的 `%GBS_HOME%\demo\odbcdemo` 目录中，可找到 `loselect.c` 文件。在同一位置，还可找到关于如何构建 `odbc_demo` 数据库的指导。

```
/*
**      loselect.c
**
** To access a smart large object
**      SQLBindCol
**      SQLBindParameter
**      SQLConnect
**      SQLFetch
**      SQLFreeStmt
**      SQLGetInfo
**      SQLDisconnect
**      SQLExecDirect
**/

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

```
#ifndef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define ERRMSG_LEN 200

UCHAR defDsn[] = "odbc_demo";

int checkError (SQLRETURN rc,
                SQLSMALLINT handleType,
                SQLHANDLE handle,
                char *errmsg)
{
    SQLRETURN retcode = SQL_SUCCESS;

    SQLSMALLINT errNum = 1;
    SQLCHAR sqlState[6];
    SQLINTEGER nativeError;
    SQLCHAR errMsg[ERRMSG_LEN];
    SQLSMALLINT textLengthPtr;

    if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
    {
        while (retcode != SQL_NO_DATA)
        {
            retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
                                     &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

            if (retcode == SQL_INVALID_HANDLE)
            {
                fprintf (stderr, "checkError function was called with an
                             invalid handle!!\n");
                return 1;
            }

            if ((retcode == SQL_SUCCESS) || (retcode ==
                                             SQL_SUCCESS_WITH_INFO))
```

```
        fprintf(stderr, "ERROR: %d:  %s : %s \n", nativeError,
                    sqlState, errMsg);

        errNum++;
    }

    fprintf(stderr, "%s\n", errMsg);
    return 1;  /* all errors on this handle have been reported */
}
else
    return 0;  /* no errors to report */
}

int main (long    argc,
          char    *argv[])
{
    /* Declare variables
    */

    /* Handles */
    SQLHDBC      hdbc;
    SQLHENV      henv;
    SQLHSTMT     hstmt;

    /* Smart large object file descriptor */
    long         lofd;
    long         lofd_valsize = 0;

    /* Smart large object pointer structure */
    char*        loptr_buffer;
    short        loptr_size;
    long         loptr_valsize = 0;

    /* Smart large object status structure */
    char*        lostat_buffer;
    short        lostat_size;
    long         lostat_valsize = 0;

    /* Smart large object data */
    char*        lo_data;
    long         lo_data_valsize = 0;

    /* Miscellaneous variables */
```

```
    UCHAR      dsn[20]; /*name of the DSN used for connecting to the
                        database*/
    SQLRETURN   rc = 0;
    int         in;

    char*       selectStmt = "SELECT advert FROM item WHERE item_num =
                        1004";
    long        mode = LO_RDONLY;
    long        lo_size;
    long        cbMode = 0, cbLoSize = 0;

/* STEP 1.  Get data source name from command line (or use default)
**         Allocate the environment handle and set ODBC version
**         Allocate the connection handle
**         Establish the database connection
**         Allocate the statement handle
*/

/* If(dsn is not explicitly passed in as arg) */
if (argc != 2)
{
    /* Use default dsn - odbc_demo */
    fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
    strcpy ((char *)dsn, (char *)defDsn);
}
else
{
    /* Use specified dsn */
    strcpy ((char *)dsn, (char *)argv[1]);
    fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
}

/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
    fprintf (stdout, "Environment Handle Allocation
                failed\nExiting!!\n");
    return (1);
}

/* Set the ODBC version to 3.5 */
```

```
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
                    (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
    SQLSetEnvAttr failed\nExiting!!\n"))
    return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
    Handle Allocation failed\nExiting!!\n"))
    return (1);

/* Establish the database connection */
rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
    failed\nExiting!!"))
    return (1);

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
    Handle Allocation failed\nExiting!!"))
    return (1);

fprintf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2.  Select a smart-large object from the database
**      -- the select statement executed is -
**      "SELECT advert FROM item WHERE item_num = 1004"
*/

/* Execute the select statement */
rc = SQLExecDirect (hstmt, selectStmt, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
    SQLExecDirect failed\n"))
    goto Exit;

fprintf (stdout, "STEP 2 done...select statement executed...smart large
    object retrieved from the databse\n");

/* STEP 3.  Get the size of the smart large object pointer structure.
**      Allocate a buffer to hold the structure.
```

```
**      Get the smart large object pointer structure from the
**      database.
**      Close the result set cursor.
*/

/* Get the size of the smart large object pointer structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size,
                sizeof(loptr_size),
                NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 3 -- SQLGetInfo
                failed\n"))
    goto Exit;

/* Allocate a buffer to hold the smart large object pointer structure */
loptr_buffer = malloc (loptr_size);

/* Bind the smart large object pointer structure buffer allocated to the
   column in the result set & fetch it from the database */
rc = SQLBindCol (hstmt, 1, SQL_C_BINARY, loptr_buffer, loptr_size,
                &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
                SQLBindCol failed\n"))
    goto Exit;

rc = SQLFetch (hstmt);
if (rc == SQL_NO_DATA_FOUND)
{
    fprintf (stdout, "No Data Found\nExiting!!\n");
    goto Exit;
}
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 -- SQLFetch
                failed\n"))
    goto Exit;

/* Close the result set cursor */
rc = SQLCloseCursor (hstmt);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
                SQLCloseCursor failed\n"))
    goto Exit;

fprintf (stdout, "STEP 3 done...smart large object pointer structure
                fetched from the database\n");
```

```
/* STEP 4. Use the smart large object's pointer structure to open it
**      and obtain the smart large object file descriptor.
**      Reset the statement parameters.
*/

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG,
                      SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
                      SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
                      SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
                      loptr_size, &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
                      SQLBindParameter failed (param 2)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_LONG,
                      SQL_INTEGER, (UDWORD)0, 0, &mode, sizeof(mode), &cbMode);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
                      SQLBindParameter failed (param 3)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{? = call ifx_lo_open(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
                      SQLExecDirect failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
                      SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 4 done...smart large object opened... file
                      descriptor obtained\n");

/* STEP 5. Get the size of the smart large object status structure.
**      Allocate a buffer to hold the structure.
**      Get the smart large object status structure from the
```

```
**      database.
**      Reset the statement parameters.
*/

/* Get the size of the smart large object status structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_STAT_LENGTH, &lostat_size,
                sizeof(lostat_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 5 -- SQLGetInfo
                failed\n"))
    goto Exit;

/* Allocate a buffer to hold the smart large object status structure. */
lostat_buffer = malloc(lostat_size);

/* Get the smart large object status structure from the database. */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
                      SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
                      SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT,
SQL_C_BINARY,
                      SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
                      lostat_size, &lostat_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
                      SQLBindParameter failed (param 2)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_stat(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
                      SQLExecDiret failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
                      SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 5 done...smart large object status structure
                fetched from the database\n");
```



```
/* STEP 6. Use the smart large object's status structure to get the
**      size of the smart large object.
**      Reset the statement parameters.
**/

/* Use the smart large object status structure to get the size of the
   smart large object */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
    lostat_size, &lostat_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_LONG,
    SQL_BIGINT, (UDWORD)0, 0, &lo_size, sizeof(lo_size), &cbLoSize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_stat_size(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLExecDirect failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 6 done...smart large object size = %ld bytes\n",
    lo_size);

/* STEP 7. Allocate a buffer to hold the smart large object's data.
**      Read the smart large object's data using its file descriptor.
**      Null-terminate the last byte of the smart large-object's data.
**      Print out the contents of the smart large object.
**      Reset the statement parameters.
**/
```

```
/* Allocate a buffer to hold the smart large object's data chunks */
lo_data = malloc (lo_size + 1);

/* Read the smart large object's data */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
    SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR,
SQL_CHAR,
    lo_size, 0, lo_data, lo_size, &lo_data_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
    SQLBindParameter failed (param 2)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_read(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
    SQLExecDirect failed\n"))
    goto Exit;

/* Null-terminate the last byte of the smart large objects data */
lo_data[lo_size] = '\0';

/* Print the contents of the smart large object */
fprintf (stdout, "Smart large object contents are.....\n\n%s\n\n",
    lo_data);

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 7 done...smart large object read completely\n");

/* STEP 8.  Close the smart large object.
*/

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
```

```
        SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
        SQLBindParameter failed\n"))
        goto Exit;

    rc = SQLExecDirect (hstmt, "{call ifx_lo_close(?)", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
        SQLExecDirect failed\n"))
        goto Exit;

    fprintf (stdout, "STEP 8 done...smart large object closed\n");

/* STEP 9.  Free the allocated buffers.
*/

    free (loptr_buffer);
    free (lostat_buffer);
    free (lo_data);

    fprintf (stdout, "STEP 9 done...smart large object buffers freed\n");

Exit:

/* CLEANUP: Close the statement handle
**      Free the statement handle
**      Disconnect from the datasource
**      Free the connection and environment handles
**      Exit
*/

/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);
```

```
fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);
```

## 5.6 检索智能大对象的状态

智能大对象的状态信息有对应的客户机函数。

下表描述状态信息和对应的客户机函数。

磁盘存储信息	描述	客户机函数
最后访问时刻	最后访问智能大对象的时刻，以秒计。 仅当为智能大对象设置 <b>LO_KEEP_LASTACCESS_TIME</b> 标志 时，此值才可用。	ifx_lo_stat_atime()
状态更改的最后时刻	智能大对象最后状态更改的时刻，以秒计。 状态的更改包括所有权更改和引用数更改。	ifx_lo_stat_ctime()
最后修改时刻（秒）	最后修改智能大对象的时刻，以秒计。	ifx_lo_stat_mtime_sec()
最后修改时刻（微秒）	最后修改时刻的微秒组件。 仅在提供微秒级系统时间的平台上支持此值。	ifx_lo_stat_mtime_usec()
引用计数	对智能大对象应用数的计数。	ifx_lo_stat_refcnt()
大小	以字节计的智能大对象的大小。	ifx_lo_stat_size()

这些时刻值（诸如最后访问时刻和最后更改时刻）可能与系统时刻略有差异。此差异是由于数据库服务器从操作系统获取时刻的算法导致的。

### 检索关于智能大对象的信息的示例

代码示例 `loinfo.c` 展示如何检索关于智能大对象的信息。

在 **UNIX™** 平台上的 `%GBS_HOME%/demo/clidemo` 目录中，以及在 **Windows™** 环境中的 `%GBS_HOME%\demo\odbcdemo` 目录中，可找到 `loinfo.c` 文件。在同一位置，还可找到关于如何构建 **odbc\_demo** 数据库的指导。

```
/*
**      loinfo.c
**
**
** To check the status of a smart large object
**
**      ODBC Functions:
```

```
**      SQLBindCol
**      SQLBindParameter
**      SQLConnect
**      SQLFetch
**      SQLFreeStmt
**      SQLDisconnect
**      SQLExecDirect
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN  20
#define ERRMSG_LEN  200

UCHAR  defDsn[] = "odbc_demo";

int checkError (SQLRETURN      rc,
                SQLSMALLINT    handleType,
                SQLHANDLE       handle,
                char            *errmsg)
{
    SQLRETURN      retcode = SQL_SUCCESS;

    SQLSMALLINT    errNum = 1;
    SQLCHAR        sqlState[6];
    SQLINTEGER     nativeError;
    SQLCHAR        errMsg[ERRMSG_LEN];
    SQLSMALLINT    textLengthPtr;

    if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
    {
        while (retcode != SQL_NO_DATA)
```

```
{
    retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
        &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

    if (retcode == SQL_INVALID_HANDLE)
    {
        fprintf (stderr, "checkError function was called with an
            invalid handle!!\n");
        return 1;
    }

    if ((retcode == SQL_SUCCESS) || (retcode ==
        SQL_SUCCESS_WITH_INFO))
        fprintf (stderr, "ERROR: %d:  %s : %s \n", nativeError,
            sqlState, errMsg);

    errNum++;
}

fprintf (stderr, "%s\n", errMsg);
return 1;  /* all errors on this handle have been reported */
}
else
    return 0;  /* no errors to report */
}

int main (long    argc,
          char    *argv[])
{
    /* Declare variables
    */

    /* Handles */
    SQLHDBC      hdbc;
    SQLHENV      henv;
    SQLHSTMT     hstmt;

    /* Smart large object file descriptor */
    long          lofd;
    long          lofd_valsize = 0;

    /* Smart large object specification structure */
    char*         lospec_buffer;
```

```
short      lospec_size;
long       lospec_valsize = 0;

/* Smart large object status structure */
char*      lostat_buffer;
short      lostat_size;
long       lostat_valsize = 0;

/* Smart large object pointer structure */
char*      loptr_buffer;
short      loptr_size;
long       loptr_valsize = 0;

/* Miscellaneous variables */
UCHAR      dsn[20]; /*name of the DSN used for connecting to the
                    database*/
SQLRETURN  rc = 0;
int         in;

char*      selectStmt = "SELECT advert FROM item WHERE item_num =
                        1004";
long       lo_size;
long       mode = LO_RDONLY;

char       sbospace_name[BUFFER_LEN];
long       sbospace_name_size = SQL_NTS;

long       cbMode = 0, cbLoSize = 0;

/* STEP 1.  Get data source name from command line (or use default).
**        Allocate the environment handle and set ODBC version.
**        Allocate the connection handle.
**        Establish the database connection.
**        Allocate the statement handle.
*/

/* If (dsn is not explicitly passed in as arg) */
if (argc != 2)
{
    /* Use default dsn - odbc_demo */
    fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
    strcpy ((char *)dsn, (char *)defDsn);
}
```

```
}
else
{
    /* Use specified dsn */
    strcpy ((char *)dsn, (char *)argv[1]);
    fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
}

/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
    fprintf (stdout, "Environment Handle Allocation
                failed\nExiting!!\n");
    return (1);
}

/* Set the ODBC version to 3.5 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
                    (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
                SQLSetEnvAttr failed\nExiting!!\n"))
    return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
                Handle Allocation failed\nExiting!!\n"))
    return (1);

/* Establish the database connection */
rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
                failed\nExiting!!"))
    return (1);

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
                Handle Allocation failed\nExiting!!"))
    return (1);
```



```
printf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2.  Select a smart-large object from the database.
**      -- the select  statement executed is -
**      "SELECT advert FROM item WHERE item_num = 1004"
*/

/* Execute the select statement */
rc = SQLExecDirect (hstmt, selectStmt, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
        SQLExecDirect failed\n"))
    goto Exit;

printf (stdout, "STEP 2 done...select statement executed...smart large
        object retrieved from the databse\n");

/* STEP 3.  Get the size of the smart large object pointer structure.
**      Allocate a buffer to hold the structure.
**      Get the smart large object pointer structure from the database.
**      Close the result set cursor.
*/

/* Get the size of the smart large object pointer structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size,
        sizeof(loptr_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 3 -- SQLGetInfo
        failed\n"))
    goto Exit;

/* Allocate a buffer to hold the smart large object pointer structure */
loptr_buffer = malloc (loptr_size);

/* Bind the smart large object pointer structure buffer allocated to the
        column in the result set & fetch it from the database */
rc = SQLBindCol (hstmt, 1, SQL_C_BINARY, loptr_buffer, loptr_size,
        &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLBindCol failed\n"))
    goto Exit;

rc = SQLFetch (hstmt);
```

```
if (rc == SQL_NO_DATA_FOUND)
{
    fprintf (stdout, "No Data Found\nExiting!!\n");
    goto Exit;
}
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 -- SQLFetch
    failed\n"))
    goto Exit;

/* Close the result set cursor */
rc = SQLCloseCursor (hstmt);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLCloseCursor failed\n"))
    goto Exit;

fprintf (stdout, "STEP 3 done...smart large object pointer structure
    fetched from the database\n");

/* STEP 4. Use the smart large object's pointer structure to open it
**         and obtain the smart large object file descriptor.
**         Reset the statement parameters.
*/

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG,
    SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
    loptr_size, &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
    SQLBindParameter failed (param 2)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_LONG,
    SQL_INTEGER, (UDWORD)0, 0, &mode, sizeof(mode), &cbMode);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
    SQLBindParameter failed (param 3)\n"))
    goto Exit;
```

```
rc = SQLExecDirect (hstmt, "{? = call  ifx_lo_open(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
    SQLExecDirect failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 4 done...smart large object opened... file
    descriptor obtained\n");

/* STEP 5.  Get the size of the smart large object status structure.
**      Allocate a buffer to hold the structure.
**      Get the smart large object status structure from the database.
**      Reset the statement parameters.
*/

/* Get the size of the smart large object status structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_STAT_LENGTH, &lostat_size,
    sizeof(lostat_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 5 -- SQLGetInfo
    failed\n"))
    goto Exit;

/* Allocate a buffer to hold the smart large object status structure. */
lostat_buffer = malloc(lostat_size);

/* Get the smart large object status structure from the database. */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
    SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT,
SQL_C_BINARY,
    SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
    lostat_size, &lostat_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
```

```
        SQLBindParameter failed (param 2)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_stat(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
        SQLExecDirect failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
        SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 5 done...smart large object status structure
        fetched from the database\n");

/* STEP 6. Use the smart large object's status structure to get the size
**         of the smart large object.
**         Reset the statement parameters.
**         You can use additional ifx_lo_stat_*() functions to get more
**         status information about the smart large object.
**         You can also use it to retrieve the smart large object
**         specification structure and get further information about the
**         smart large object using its specification structure.
**
*/

/* Use the smart large object status structure to get the size of the
   smart large object. */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
        SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
        lostat_size, &lostat_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
        SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_LONG,
        SQL_BIGINT, (UDWORD)0, 0, &lo_size, sizeof(lo_size), &cbLoSize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
        SQLBindParameter failed (param 1)\n"))
    goto Exit;
```

```
rc = SQLExecDirect (hstmt, "{call ifx_lo_stat_size(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLExecDirect failed\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "LARGE OBJECT SIZE = %ld\n", lo_size);
fprintf (stdout, "STEP 6 done...smart large object size retrieved\n");

/* STEP 7.  Get the size of the smart large object specification structure.
**      Allocate a buffer to hold the structure.
**      Get the smart large object specification structure from the
**      database.
**      Reset the statement parameters.
*/

/* Get the size of the smart large object specification structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_SPEC_LENGTH, &lo_spec_size,
    sizeof(lo_spec_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 7 -- SQLGetInfo
    failed\n"))
    goto Exit;

/* Allocate a buffer to hold the smart large object specification
    structure */
lo_spec_buffer = malloc (lo_spec_size);

/* Get the smart large object specification structure from the
    database */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_INFX_UDT_FIXED, (UDWORD)lo_spec_size, 0, lo_spec_buffer,
    lo_spec_size, &lo_spec_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_BINARY,
```

```
        SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
        lospec_size, &lospec_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
        SQLBindParameter failed (param 2)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_stat_cspec(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
        SQLExecDirect failed\n"))
    goto Exit;

fprintf (stdout, "STEP 7 done...smart large object status structure
        fetched from the database\n");

/* STEP 8.  Use the smart large object's specification structure to get
**         the sbospace name where the smart large object is stored.
**         Reset the statement parameters.
*/

/*  Use the smart large object's specification structure to get the
    sbospace name of the smart large object. */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
        SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
        lospec_size, &lospec_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
        SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR,
SQL_CHAR,
        BUFFER_LEN, 0, sbospace_name, BUFFER_LEN, &sbospace_name_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
        SQLBindParameter failed (param 2)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_specget_sbospace(?, ?)}",
        SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
        SQLExecDirect failed\n"))
    goto Exit;

fprintf (stdout, "LARGE OBJECT SBOSPACE NAME = %s\n", sbospace_name);
```

```
printf (stdout, "STEP 8 done...large object sbspace name retrieved\n");

/* STEP 9. Close the smart large object.
 */

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
    SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 9 --
    SQLBindParameter failed\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_close(?)", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 9 --
    SQLExecDirect failed\n"))
    goto Exit;

printf (stdout, "STEP 9 done...smart large object closed\n");

/* STEP 10.Free the allocated buffers.
 */

free (loptr_buffer);
free (lostat_buffer);
free (lospec_buffer);

printf (stdout, "STEP 10 done...smart large object buffers freed\n");

Exit:

/* CLEANUP: Close the statement handle.
**      Free the statement handle.
**      Disconnect from the datasource.
**      Free the connection and environment handles.
**      Exit.
 */

/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
```

```
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);

fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);
}
```

## 5.7 从文件读取智能大对象，或将它写至文件

可使用 SQL 函数来从文件读取智能大对象，或将它写至文件。

可使用 SQL 函数 FILETOBLOB() 和 FILETOCLOB(), 来将数据从文件转移至智能大对象。该文件可在客户机计算机上，或在服务器计算机上。

可使用 SQL 函数 LOTOFILE(), 来将数据从智能大对象转移至文件。该文件可能在客户机计算机上，或在服务器计算机上。LOTOFILE()接受智能大对象指针作为参数。对于此参数，可使用智能大对象指针结构。

要获取关于这些 SQL 函数的更多信息，请参阅《GBase 8s SQL 指南：语法》。



## 6 行和集合

行和集合是由一个或多个元素组成的复合值。

仅当您的数据库服务器是 GBase 8s 时才能应用这些主题的信息。

可以使用 `SELECT`、`UPDATE`、`INSERT` 和 `DELETE` 语句访问整个行或集合。但是，这些 `SQL` 语句不会让您访问集合或行中的任一元素。要访问元素，需要检索行或集合然后从行或集合的本地副本访问元素。

有关行和集合的更多信息，请参阅《GBase 8s SQL 指南：参考》和《GBase 8s 用户定义的例程和数据类型开发者指南》。

### 6.1 分配和绑定行或集合缓冲区

当检索行或集合时，数据库服务器将行或集合放在 GBase 8s ODBC Driver 应用程序本地的缓冲区中。

要调用和绑定集合或行缓冲区：

1. 调用 `ifx_rc_create()` 分配缓冲区。
2. 调用 `SQLBindCol()` 绑定缓冲区以绑定数据库列。
3. 执行 `SELECT` 语句将行或集合传输到本地缓冲区。
4. 使用行或集合缓冲区。
5. 调用 `ifx_rc_free()` 释放缓冲区。

#### 6.1.1 固定型缓冲区和不固定型缓冲区

固定型缓冲区和不固定型缓冲区之间存在一些差异。

下表描述了固定型缓冲区和不固定型缓冲区之间的差异。

缓冲区	描述
固定型	<p>当调用 ifx_rc_create() 创建行或集合缓冲区时，为缓冲区指定下列数据类型：</p> <ul style="list-style-type: none"><li>缓冲区数据类型（行或一种集合类型）</li><li>行或集合中的元素的数据类型</li></ul> <p>当检索行或集合时，数据库服务器比较源和目标数据类型并将数据从 GBase 8s SQL 数据类型转换为其它必需的数据类型。</p> <p>在将数据检索到缓冲区中之前可以修改行或集合。</p>
不固定型	<p>当调用 ifx_rc_create()创建行或集合缓冲区时，您只能指定缓冲区数据类型（行或集合），而不能指定元素类型。</p> <p>当检索行或集合时，数据库服务器不会比较源和目标数据类型，因为未指定目标数据类型。相反，行或集合缓冲区采用源数据的数据类型。</p> <p>在修改行或集合缓冲区后，您必须初始化它们。要初始化缓冲区，将行或集合检索到其中。</p> <p>即使包含数据，缓冲区类型也不会保留。</p>

6. 1. 2 缓冲区和内存分配

当将数据检索到包含行或集合的缓冲区时，GBase 8s ODBC Driver不会重复使用相同的缓冲区。

相反，GBase 8s ODBC Driver 执行以下操作：

- 1. 创建行或集合缓冲区。
- 2. 将缓冲区和给定的缓冲区句柄关联。
- 3. 释放原来的缓冲区。

6. 1. 3 SQL 数据

数据库服务器调用转型函数将数据从源GBase 8s SQL 数据类型转换为目标 GBase 8s SQL 数据类型。

如果数据库服务器上的行或集合的数据类型与检索行或集合缓冲区的数据类型不同，则数据库服务器调用强制转型函数将数据从源 GBase 8s SQL 数据类型转换为目标GBase 8s SQL 数据类型。下表列出了源数据类型和目标数据类型每组组合的强制转型函数的提供者。数据类型提供的强制转型函数位于数据库服务器上。

源数据类型	目标数据类型	强制转型函数的提供者
内置	内置	数据库服务器
内置	扩展	数据类型
扩展	内置	数据类型
扩展	扩展	数据类型

### 6.1.4 执行本地访存

GBase 8s ODBC Driver从客户端计算机的一个位置检索行或集合到客户端计算机上的另一个位置时，执行本地访存。

本地访存在 SQL 数据转换上具有以下限制：

- GBase 8s ODBC Driver无法转换在数据库服务器上强制转型函数的扩展数据类型。
- GBase 8s ODBC Driver无法将一个命名行类型转换为另一种类型。只有数据库服务器可以执行此类型的转换。
- 在检索整个行或集合时，GBase 8s ODBC Driver 无法转换 SQL 数据类型。因此，只有在源和目标的内部结构相同或者目标是不固定型缓冲区时，GBase 8s ODBC Driver 可以执行整个行或集合的本地访存。

例如，如果您将本地集合定义为 **list (char(1) not null)**，数据库服务器可以将 **list (int not null)**值从数据库服务器放到本地集合中。在此操作期间，数据库服务器将每个整数转换为字符串，并建立新的列表返回到客户端计算机。在将本地整数列表检索到字符列表时，不能在客户端计算机上执行此操作。

执行本地访存：

1. 调用 `ifx_rc_create()` 分配一个行或集合缓冲区。
2. 调用 `SQLBindCol()` 绑定缓冲区来处理本地行或集合。
3. 执行 `SELECT` 语句将行或集合数据传输到本地缓冲区。
4. 对于行或集合中的每一个元素，调用 `ifx_rc_fetch()` 将值复制到缓冲区。
5. 使用行或集合缓冲区。
6. 调用 `ifx_rc_free()` 释放缓冲区。

### 6.1.5 从数据库检索行和集合的示例

该样本程序 `rcselect.c`，从数据库检索并显示行和集合的数据。

该示例还说明了相同的客户端函数可以交替使用行和集合句柄。

可以在 UNIX™ 系统的 `%GBS_HOME%/demo/clidemo` 目录中找到 `rcselect.c`文件，在 Windows™ 系统的 `%GBS_HOME%\demo\odbcdemo` 目录中找到此文件。还可以找到有关如何在同一位置创建 `odbc_demo` 数据库的说明。

```
/*
**      rcselect.c
**
** To access rows and collections
**      ODBC Functions:
**      SQLBindParameter
**      SQLConnect
**      SQLDisconnect
**      SQLExecDirect
```

```
**      SQLFetch
**      SQLFreeStmt
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN      25
#define ERRMSG_LEN      200

UCHAR    defDsn[] = "odbc_demo";

int checkError (SQLRETURN      rc,
                SQLSMALLINT    handleType,
                SQLHANDLE      handle,
                char            *errmsg)
{
    SQLRETURN      retcode = SQL_SUCCESS;

    SQLSMALLINT    errNum = 1;
    SQLCHAR        sqlState[6];
    SQLINTEGER     nativeError;
    SQLCHAR        errMsg[ERRMSG_LEN];
    SQLSMALLINT    textLengthPtr;

    if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
    {
        while (retcode != SQL_NO_DATA)
        {
            retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
                                     &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

            if (retcode == SQL_INVALID_HANDLE)
            {
```

```

        fprintf (stderr, "checkError function was called with an
                    invalid handle!!\n");
        return 1;
    }

    if ((retcode == SQL_SUCCESS) || (retcode ==
SQL_SUCCESS_WITH_INFO))
        fprintf (stderr, "ERROR: %d:  %s : %s \n", nativeError,
                    sqlState, errMsg);

    errNum++;
}

fprintf (stderr, "%s\n", errMsg);
return 1;  /* all errors on this handle have been reported */
}
else
    return 0;  /* no errors to report */
}

/*
** Executes the given select statement and assumes the results will be
** either rows or collections. The 'hrc' parameter may reference either
** a row or a collection. Rows and collection handles may often be used
** interchangeably.
**
** Each row of the select statement will be fetched into the given row or
** collection handle. Then each field of the row or collection will be
** individually converted into a character buffer and displayed.
**
** This function returns 0 if an error occurs, else returns 1
**
*/

int do_select (SQLHDBCjdbc,
               char*    select_str,
               HINFX_RChrc)
{
    SQLHSTMT    hRCstmt;
    SQLHSTMT    hSelectStmt;
    SQLRETURN    rc = 0;

    short        index, rownum;

```

```
short      position = SQL_INFX_RC_ABSOLUTE;
short      jump;

char       fname[BUFFER_LEN];
char       lname[BUFFER_LEN];
char       rc_data[BUFFER_LEN];

SQLINTEGER      cbFname = 0, cbLname = 0, cbHrc = 0;
SQLINTEGERcbPosition = 0, cbJump = 0, cbRCData = 0;

/* STEP A.  Allocate the statement handles for the select statement and
**          the statement used to retrieve the row/collection data.
*/

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hRCStmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step A -- Statement
          Handle Allocation failed for row/collection
          statement\nExiting!!"))
    return 0;

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hSelectStmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step A -- Statement
          Handle Allocation failed for select statement\nExiting!!"))
    return 0;

fprintf (stdout, "STEP A done...statement handles allocated\n");

/* STEP B.  Execute the select statement.
**          Bind the result set columns -
**          -- col1 = fname
**          col2 = lname
**          col3 = row/collection data
*/

/* Execute the select statement */
rc = SQLExecDirect (hSelectStmt, select_str, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
          SQLExecDirect failed\n"))
    return 0;
```

```
/* Bind the result set columns */
rc = SQLBindCol (hSelectStmt, 1, SQL_C_CHAR, (SQLPOINTER)fname,
    BUFFER_LEN, &cbFname);
if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
    SQLBindCol failed for column 'fname'\n"))
    return 0;

rc = SQLBindCol (hSelectStmt, 2, SQL_C_CHAR, (SQLPOINTER)lname,
    BUFFER_LEN, &cbLname);
if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
    SQLBindCol failed for column 'lname'\n"))
    return 0;

rc = SQLBindCol (hSelectStmt, 3, SQL_C_BINARY, (SQLPOINTER)hrc,
    sizeof(HINFX_RC), &cbHrc);
if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
    SQLBindCol failed for row/collection column\n"))
    return 0;

fprintf (stdout, "STEP B done...select statement executed and result set
    columns bound\n");

/* STEP C. Retrieve the results.
*/

for (rownum = 1;; rownum++)
{
    rc = SQLFetch (hSelectStmt);
    if (rc == SQL_NO_DATA_FOUND)
    {
        fprintf (stdout, "No data found...\n");
        break;
    }
    else if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in
        Step C -- SQLFetch failed\n"))
        return 0;

    fprintf(stdout, "Retrieving row number %d:\n\tfname -- %s\n\tlname --
        %s\n\tRow/Collection Data --\n", rownum, fname, lname);

    /* For each row in the result set, display each field of the
```

```
        retrieved row/collection */
    for (index = 1;; index++)
    {
        strcpy(rc_data, "<null>");

        /* Each value in the local row/collection will be fetched into a
         * character buffer and displayed using fprintf().
         */

        rc = SQLBindParameter (hRCStmt, 1, SQL_PARAM_OUTPUT,
SQL_C_CHAR,
        SQL_CHAR, 0, 0, rc_data, BUFFER_LEN, &cbRCData);
        if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
        SQLBindParameter failed (param 1)\n"))
            return 0;

        rc = SQLBindParameter (hRCStmt, 2, SQL_PARAM_INPUT,
SQL_C_BINARY,
        SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hrc,
        sizeof(HINFX_RC), &cbHrc);
        if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
        SQLBindParameter failed (param 2)\n"))
            return 0;

        rc = SQLBindParameter (hRCStmt, 3, SQL_PARAM_INPUT,
SQL_C_SHORT,
        SQL_SMALLINT, 0, 0, &position, 0, &cbPosition);
        if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
        SQLBindParameter failed (param 3)\n"))
            return 0;

        jump = index;
        rc = SQLBindParameter (hRCStmt, 4, SQL_PARAM_INPUT,
SQL_C_SHORT,
        SQL_SMALLINT, 0, 0, &jump, 0, &cbJump);
        if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
        SQLBindParameter failed (param 4)\n"))
            return 0;

        rc = SQLExecDirect(hRCStmt, "{ ? = call ifx_rc_fetch( ?, ?, ? )
        SQL_NTS);
        if (rc == SQL_NO_DATA_FOUND)
        {
```



```

        break;
    }
    else if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in
        Step C -- SQLExecDirect failed\n"))
        return 0;

    /* Display retrieved row */
    fprintf(stdout, "\\t\\t%d: %s\\n", index, rc_data);
}
}

fprintf (stdout, "STEP C done...results retrieved\\n");

/* Free the statement handles */
SQLFreeHandle (SQL_HANDLE_STMT, hSelectStmt);
SQLFreeHandle (SQL_HANDLE_STMT, hRCStmt);

return 1; /* no error */
}

/*
 * This function allocates the row and collection buffers, passes
 * them to the do_select() function, along with an appropriate select
 * statement and then frees all allocated handles.
 */
int main (long argc,
        char *argv[])
{
    /* Declare variables
    */

    /* Handles */
    SQLHDBC      hdbc;
    SQLHENV      henv;
    SQLHSTMT     hstmt;
    HINFX_RC     hrow, hlist;

    /* Miscellaneous variables */

    UCHAR        dsn[20]; /*name of the DSN used for connecting to the
                           database*/

    SQLRETURN     rc = 0;
    int          in;

```

```
int      data_size = SQL_NTS;
char*    listSelectStmt = "SELECT fname, lname, contact_dates FROM
                           customer";
char*    rowSelectStmt = "SELECT fname, lname, address FROM
                           customer";

SQLINTEGER  cbHlist = 0, cbHrow = 0;

/* STEP 1.  Get data source name from command line (or use default).
**          Allocate environment handle and set ODBC version.
**          Allocate connection handle.
**          Establish the database connection.
**          Allocate the statement handle.
*/

/* If(dsn is not explicitly passed in as arg) */
if (argc != 2)
{
    /* Use default dsn - odbc_demo */
    fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
    strcpy ((char *)dsn, (char *)defDsn);
}
else
{
    /* Use specified dsn */
    strcpy ((char *)dsn, (char *)argv[1]);
    fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
}

/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
    fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
    return (1);
}

/* Set the ODBC version to 3.5 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
                    (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
```

```
        SQLSetEnvAttr failed\nExiting!!"))
    return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
        Handle Allocation failed\nExiting!!"))
    return (1);

/* Establish the database connection */
rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
        failed\n"))
    return (1);

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
        Handle Allocation failed\nExiting!!"))
    return (1);

fprintf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2.  Allocate an unfixed collection handle.
**      Retrieve database rows containing a list.
**      Reset the statement parameters.
*/

/* Allocate an unfixed list handle */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
        SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, &hlist, sizeof(HINFX_RC),
        &cbHlist);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
        SQLBindParameter (param 1) failed\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR,
        0, 0, (UCHAR *) "list", 0, &data_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
        SQLBindParameter (param 2) failed\n"))
    goto Exit;
```

```
rc = SQLExecDirect (hstmt, "{? = call ifx_rc_create(?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
    SQLExecDirect failed\n"))
    goto Exit;

/* Retrieve database rows containing a list */
if (!do_select (hdbc, listSelectStmt, hlist))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 2 done...list data retrieved\n");
fprintf (stdout, "\nHit <Enter> to continue...");
in = getchar ();

/* STEP 3.  Allocate an unfixed row handle.
**      Retrieve database rows containing a row.
**      Reset the statement parameters.
*/

/* Allocate an unfixed row handle */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
    SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, &hrow, sizeof(HINFX_RC),
    &cbHrow);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLBindParameter (param 1) failed\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR,
    0, 0, (UCHAR *) "row", 0, &data_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLBindParameter (param 2) failed\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, "{? = call ifx_rc_create(?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLExecDirect failed\n"))
```

```
    goto Exit;

/* Retrieve database rows containing a row */
if (!do_select (hdbc, rowSelectStmt, hrow))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 3 done...row data retrieved\n");

/* STEP 4. Free the row and list handles.
 */

/* Free the row handle */
rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC),
    &cbHrow);

rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

/* Free the list handle */
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC),
    &cbHlist);

rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

fprintf (stdout, "STEP 4 done...row and list handles freed\n");

Exit:

/* CLEANUP: Close the statement handle.
**      Free the statement handle.
**      Disconnect from the datasource.
**      Free the connection and environment handles.
**      Exit.
*/
```

```
/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);
fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);
```

## 6.2 在客户端创建行和列表的示例

此代码示例 `rccreate.c`，在客户端创建一个行和列表，添加项，并将它们插入到数据库。

可以在 UNIX<sup>™</sup> 的 `%GBS_HOME%/demo/clidemo` 目录和 Windows<sup>™</sup> 的 `%GBS_HOME%\demo\odbcdemo` 目录中找到 `rccreate.c` 文件。还可以找到有关如何在同一位置创建 **odbc\_demo** 数据库的说明。

```
/*
**      rccreate.c
**
**  To create a collection & insert it into the database table
**
**
**      ODBC Functions:
**      SQLBindParameter
**      SQLConnect
**      SQLDisconnect
**      SQLExecDirect
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef NO_WIN32
#include <io.h>
```

```
#include <windows.h>
#include <conio.h>
#ifdef /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN    25
#define ERRMSG_LEN    200

UCHAR    defDsn[] = "odbc_demo";

int checkError (SQLRETURNrc,
                SQLSMALLINT    handleType,
                SQLHANDLE      handle,
                char            *errmsg)
{
    SQLRETURN    retcode = SQL_SUCCESS;

    SQLSMALLINT    errNum = 1;
    SQLCHAR        sqlState[6];
    SQLINTEGER     nativeError;
    SQLCHAR        errMsg[ERRMSG_LEN];
    SQLSMALLINT    textLengthPtr;
    if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
    {
        while (retcode != SQL_NO_DATA)
        {
            retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
                                     &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

            if (retcode == SQL_INVALID_HANDLE)
            {
                fprintf (stderr, "checkError function was called with an
                                invalid handle!!\n");
                return 1;
            }

            if ((retcode == SQL_SUCCESS) || (retcode ==
                                             SQL_SUCCESS_WITH_INFO)) fprintf (stderr,
"ERROR: %d:  %s
                                : %s \n", nativeError, sqlState, errMsg);

            errNum++;
        }
    }
}
```

```

    }

    fprintf (stderr, "%s\n", errmsg);
    return 1;    /* all errors on this handle have been reported */
}
else
    return 0;    /* no errors to report */
}

int main (long    argc,
          char    *argv[])
{
    /* Declare variables
    */

    /* Handles */
    SQLHDB      hdbc;
    SQLHENV      henv;
    SQLHSTMT     hstmt;

    HINFX_RC     hrow;
    HINFX_RC     hlist;

    /* Miscellaneous variables */
    UCHAR        dsn[20]; /*name of the DSN used for connecting to the
                           database*/

    SQLRETURN     rc = 0;
    int          i, in;
    int          data_size = SQL_NTS;
    short        position = SQL_INFX_RC_ABSOLUTE;
    short        jump;

    UCHAR        row_data[4][BUFFER_LEN] = {"520 Topaz Way", "Redwood City",
                                             "CA", "94062"};
    int          row_data_size = SQL_NTS;

    UCHAR        list_data[2][BUFFER_LEN] = {"1991-06-20", "1993-07-17"};
    int          list_data_size = SQL_NTS;

    char*        insertStmt = "INSERT INTO customer VALUES (110, 'Roy',
                          'Jaeger', ?, ?)";

    SQLINTEGER    cbHrow = 0, cbHlist = 0, cbPosition = 0, cbJump = 0;
    /* STEP 1.  Get data source name from command line (or use default).

```



```
**      Allocate environment handle and set ODBC version.
**      Allocate connection handle.
**      Establish the database connection.
**      Allocate the statement handle.
*/

/* If(dsn is not explicitly passed in as arg) */
if (argc != 2)
{
    /* Use default dsn - odbc_demo */
    fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
    strcpy ((char *)dsn, (char *)defDsn);
}
else
{
    /* Use specified dsn */
    strcpy ((char *)dsn, (char *)argv[1]);
    fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
}

/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
    fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
    return (1);
}

/* Set the ODBC version to 3.5 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
                    (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
                    SQLSetEnvAttr failed\nExiting!!"))
    return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
                    Handle Allocation failed\nExiting!!"))
    return (1);

/* Establish the database connection */
rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
```

```
        failed\n"))
    return (1);

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt );
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
        Handle Allocation failed\nExiting!!"))
    return (1);

fprintf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2.  Allocate fixed-type row handle -- this creates a non-null row
**          buffer, each of whose values is null, and can be updated.
**          Allocate a fixed-type list handle -- this creates a non-null
**          but empty list buffer into which values can be inserted.
**          Reset the statement parameters.
*/

/* Allocate a fixed-type row handle -- this creates a row with each
   value empty */

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
        SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, &hrow, sizeof(HINFX_RC),
        &cbHrow);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
        SQLBindParameter (param 1) failed for row handle\n"))        goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR,
        0, 0, (UCHAR *) "ROW(address1 VARCHAR(25), city VARCHAR(15), state
        VARCHAR(15), zip  VARCHAR(5))", 0, &data_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
        SQLBindParameter (param 2) failed for row handle\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, (UCHAR *) "{? = call ifx_rc_create(?)",
        SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
        SQLExecDirect failed for row handle\n"))
    goto Exit;

/* Allocate a fixed-type list handle */
```

```

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
    SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, &hlist, sizeof(HINFX_RC),
    &cbHlist);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
    SQLBindParameter (param 1) failed for list handle\n"))
    goto Exit;

data_size = SQL_NTS;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR,
    0, 0, (UCHAR *) "LIST (DATETIME YEAR TO DAY NOT NULL)", 0,
    &data_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
    SQLBindParameter (param 2) failed for list handle\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, (UCHAR *) "{? = call ifx_rc_create(?)}",
    SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
    SQLExecDirect failed for list handle\n"))
    goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 2 done...fixed-type row and collection handles
    allocated\n");

/* STEP 3.  Update the elements of the fixed-type row buffer allocated.
**      Insert elements into the fixed-type list buffer allocated.
**      Reset the statement parameters.
*/

/* Update elements of the row buffer */
for (i=0; i<4; i++)
{
    rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
        SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC),
        &cbHrow);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --

```

```
        SQLBindParameter (param 1) failed for row handle\n"))
    goto Exit;

    rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
        SQL_CHAR, BUFFER_LEN, 0, row_data[i], 0, &row_data_size);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLBindParameter (param 2) failed for row handle\n"))
        goto Exit;

    rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT,
        SQL_SMALLINT, 0, 0, &position, 0, &cbPosition);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLBindParameter (param 3) failed for row handle\n"))
        goto Exit;        jump = i + 1;
    rc = SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT,
        SQL_SMALLINT, 0, 0, &jump, 0, &cbJump);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLBindParameter (param 4) failed for row handle\n"))
        goto Exit;

    rc = SQLExecDirect (hstmt,
        (UCHAR *)"{call ifx_rc_update(?, ?, ?, ?)}", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLExecDirect failed for row handle\n"))
        goto Exit;
}

/* Insert elements into the list buffer */
for (i=0; i<2; i++)
{
    rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
        SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC),
        &cbHlist);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLBindParameter (param 1) failed for list handle\n"))
        goto Exit;

    rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
        SQL_DATE, 25, 0, list_data[i], 0, &list_data_size);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLBindParameter (param 2) failed for list handle\n"))
        goto Exit;
```

```

    rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT,
        SQL_SMALLINT, 0, 0, &position, 0, &cbPosition);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLBindParameter (param 3) failed for list handle\n"))
        goto Exit;

    jump = i + 1;
    rc = SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT,
        SQL_SMALLINT, 0, 0, &jump, 0, &cbJump);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLBindParameter (param 4) failed for list handle\n"))
        goto Exit;

    rc = SQLExecDirect (hstmt,
        (UCHAR *) "{call ifx_rc_insert( ?, ?, ?, ? )}", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
        SQLExecDirect failed for list handle\n"))
        goto Exit;
}

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
    SQLFreeStmt failed\n"))
    goto Exit;

fprintf (stdout, "STEP 3 done...row and list buffers populated\n");

/* STEP 4.  Bind parameters for the row and list handles.
**      Execute the insert statement to insert the new row into table
**      'customer'.
**/

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hrow,
    sizeof(HINFX_RC), &cbHrow);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
    SQLBindParameter failed (param 1)\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hlist,

```

```
        sizeof(HINFX_RC), &cbHlist);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
        SQLBindParameter failed (param 2)\n"))
    goto Exit;

rc = SQLExecDirect (hstmt, (UCHAR *)insertStmt, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
        SQLExecDirect failed\n"))
    goto Exit;

fprintf (stdout, "STEP 4 done...new row inserted into table
        'customer'\n");

/* STEP 5.  Free the row and list handles.
 */

/* Free the row handle */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
        SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC),
        &cbHrow);

rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

/* Free the list handle */
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
        SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC),
        &cbHlist);

rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

fprintf (stdout, "STEP 5 done...row and list handles freed\n");

Exit:

/* CLEANUP: Close the statement handle.
**      Free the statement handle.
**      Disconnect from the datasource.
**      Free the connection and environment handles.
**      Exit.
*/

/* Close the statement handle */
```

```
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);

fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);
```

### 6.3 修改行或集合

GBase 8s ODBC Driver 提供可用于修改行和集合的函数。

下表提供了 GBase 8s ODBC Driver提供的用于修改行和集合的函数的概述。

函数	修改	行	集合
ifx_rc_delete()	删除元素	否	是
ifx_rc_insert()	插入元素	否	是（参阅下表。）
ifx_rc_setnull()	将行或集合设置为空	是	是
ifx_rc_update()	更改元素的值	是	是

下表描述了您可以插入元素的集合的位置。只能在 SET 或 MULTISSET 集合的末尾插入元素，因为这些类型的集合中的元素的位置没有顺序。

	开始	中间	末尾
<b>List</b>	是	是	是
<b>Multiset</b>	否	否	是
<b>Set</b>	否	否	是

**提示：** 如果您只需要插入或更新具有文字值的的行或集合，则不需要使用行或集合缓冲区。相反，您可以在 INSERT 语句的 INTO 子句或 UPDATE 语句的 SET 子句中显式列出文字值。

每一行和集合都维护一个指向行或集合中的当前元素的查找位置。创建行或集合时，该查找位置指向行或集合中的第一个元素。所有对客户端函数的调用都共享同一个行或集合缓冲区的查找位置。因此，一个客户端函数会影响另一个使用相同缓冲区句柄的客户端函数。下表描述了客户端函数如何更改查找位置。

客户端函数	作用于	变更
ifx_rc_delete()	指定的位置。	将查找位置设置为删除后的位置。
ifx_rc_fetch()	指定的位置。	将查找位置设置到指定的位置。
ifx_rc_insert()	指定位置之前。	将查找位置设置到指定的位置。
ifx_rc_update()	指定的位置。	将查找位置设置到指定的位置。

## 6.4 检索行或集合的信息

GBase 8s ODBC Driver提供可用于检索有关行和集合的函数。

下表提供了 GBase 8s ODBC Driver用于检索行和集合信息的函数概述。ifx\_rc\_describe() 函数返回行或集合中元素的数据类型。

函数	信息	参考
ifx_rc_count()	列数	ifx_rc_count() 函数
ifx_rc_describe()	数据类型信息	ifx_rc_describe() 函数
ifx_rc_isnull()	指示是否为空的值	ifx_rc_isnull() 函数
ifx_rc_typespec()	类型描述	ifx_rc_typespec() 函数



## 7 客户端函数

这些主题描述了 GBase 8s ODBC Driver 客户端函数。使用这些函数访问和操纵智能大对象以及行和集合。

仅当您的数据库服务器是 GBase 8s 时，这些函数才适用。

### 7.1 调用客户端函数

本节描述了客户端函数的语法，它们的输入/输出参数，返回值和 SQL\_BIGINT。

#### 7.1.1 SQL 语法

这是客户端函数的 SQL 语法。

```
{? = call client_function(?, ?,...)}
```

仅当第一个参数是输出参数时才使用首个参数标记符（“?”）。

以下代码示例当第一个参数是输出参数时，调用客户端函数的语法：

```
{? = call ifx_lo_open(?, ?, ?)}
```

以下代码示例当第一个参数不是输出参数时，调用客户端函数的语法：

```
{call ifx_lo_create(?, ?, ?, ?)}
```

#### 7.1.2 函数语法

数据库服务器和应用程序都可以部分实现客户端函数。

可以使用 SQLPrepare() 和 SQLExecute() 或者使用 SQLExecDirect() 执行客户端函数。在调用 SQLExecute() 或 SQLExecDirect() 之前，需要调用 SQLBindParameter() 或 SQLBindCol() 绑定每个参数。

使用 SQLPrepare() 和 SQLExecute() 执行客户端函数

可以使用 SQLPrepare() 和 SQLExecute() 执行客户端函数。

要使用 SQLPrepare() 和 SQLExecute() 执行客户端函数：

1. 为客户端函数准备 SQL 语句。
2. 绑定参数。
3. 执行 SQL 语句。

下列代码示例说明了使用 ifx\_lo\_open() 的这些步骤：

```
rc = SQLPrepare(hstmt, "{? = call ifx_lo_open(?, ?, ?)}", SQL_NTS);  
rc = SQLBindParameter(...);
```

```
rc = SQLExecute(hstmt);
```

### 使用 **SQLExecDirect()** 执行客户端函数

可以使用 **SQLExecDirect()** 函数执行客户端函数。

要使用 **SQLExecDirect()** 执行客户端函数：

1. 绑定参数。
2. 执行 SQL 语句。

以下示例代码说明了使用 **ifx\_lo\_open()**的这些步骤：

```
rc = SQLBindParameter(...);  
rc = SQLExecDirect(hstmt, "{? = call ifx_lo_open(?, ?, ?)}", SQL_NTS);
```

### 7.1.3 输入和输出参数

大多数客户端函数的输入和输出参数是客户端应用程序的输出参数。

但是，接受输入/输出参数的客户端函数会在内部初始化这些参数，然后将其发送数据库服务器，并请求执行客户端函数。因此，您需要将这些参数作为输入/输出参数传递给驱动程序。

### 7.1.4 **SQL\_BIGINT** 数据类型

GBase 8s 支持 **INT8** GBase 8s SQL 数据类型。

缺省情况下，驱动程序将 **INT8** 映射到 **SQL\_BIGINT** GBase 8s ODBC Driver SQL 数据类型，**SQL\_C\_CHAR** 缺省映射为 GBase 8s ODBC Driver C 数据类型。但是，客户端函数不能访问所有的数据类型转换函数。因此，当使用 **SQL\_BIGINT** 类型的值时，您必须使用除 **SQL\_C\_CHAR** 以外的数据类型。

例如，在调用 **ifx\_lo\_specset\_estbytes()** 之前，必须为 *estbytes* 输入参数绑定一个变量。因为 *estbytes* 是 **SQL\_BIGINT**，通常会将 *estbytes* 绑定到 **SQL\_C\_CHAR**。但是，对客户端函数，**SQL\_C\_CHAR** 不适用于 **SQL\_BIGINT**。以下代码说明如何将 *estbytes* 绑定到 **SQL\_C\_LONG** 而不是针对 **ifx\_lo\_specset\_estbytes()** 的 **SQL\_C\_CHAR**：

```
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,  
                      SQL_BIGINT, (UDWORD)0, 0, &estbytes, sizeof(estbytes), NULL);  
rc = SQLExecDirect(hstmt, "{call ifx_lo_specset_estbytes(?, ?)}", SQL_NTS);
```

### 7.1.5 返回码

客户端函数不提供返回码。

有关成功或失败信息，请参阅用于调用客户端函数（**SQLExecDirect()** 或 **SQLExecute()**）的 GBase 8s ODBC Driver 函数的返回码。

## 7.2 智能大对象函数

本节描述了驱动程序为智能大对象提供的每个客户端函数。

### 7.2.1 ifx\_lo\_alter() 函数

ifx\_lo\_alter() 函数更改智能大对象的存储特性。

语法

ifx\_lo\_alter(*loptr*, *lospec*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>loptr</i>	SQL_INFX_UDT_FIXED	输入	智能大对象指针结构
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构

用法

ifx\_lo\_alter() 函数执行以下步骤来更改智能大对象的存储特性：

1. 获取智能大对象的互斥锁。
2. 利用 *lospec* 智能大对象中的特性来更新智能大对象的存储特性。ifx\_lo\_alter() 函数允许您更改以下存储特征：
  - 日志记录特性
  - 最后访问时间特性
  - 扩展大小
3. 解锁智能大对象。

作为调用此函数的另一种方法，如果您只想要更改其中的一个特性，则可以调用以下函数之一：

- ifx\_lo\_specset\_flags()
- ifx\_lo\_specset\_extsz()

### 7.2.2 ifx\_lo\_close() 函数

ifx\_lo\_close() 函数关闭智能大对象。

语法

ifx\_lo\_close(*lofd*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输入	智能大对象文件描述符

用法

ifx\_lo\_close() 函数关闭智能大对象。在此函数的操作期间，数据库服务器尝试解锁智能大对象。如果隔离方式是可重复读或者锁是互斥锁，则数据库服务器不会释放此锁，除非事务结束。

**提示：** 如果您不在 BEGIN WORK 事务块中修改智能大对象，则每次更改是一个单独的事务。

7. 2. 3 ifx\_lo\_col\_info() 函数

ifx\_lo\_col\_info() 函数更改具有列级存储特性的智能大对象规范结构。

语法

ifx\_lo\_col\_info(*colname*, *lospec*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>colname</i>	SQL_CHAR	输入	指向包含数据库列名称的缓冲区 该值必须具有下列格式： database@server_name:table.column  如果列在 ANSI 兼容的数据库中，则可以包含所有者名称。在这种情况下，使用以下格式： database@server_name:owner.table.column
<i>lospec</i>	SQL_INFX_UDT_FIXED	I/O	智能大对象规范结构

用法

ifx\_lo\_col\_info() 函数将智能大对象规范结构的字段设置为 *colname* 数据库列的存储特征。如果指定的列没有定义列级别存储特征，则数据库服务器将使用继承的存储特征。

**重要：** 在调用此函数之前，必须调用 ifx\_lo\_def\_create\_spec()。

7. 2. 4 ifx\_lo\_create() 函数

ifx\_lo\_create() 函数创建并打开新的智能大对象。

语法

ifx\_lo\_create(*lospec*, *flags*, *loptr*, *lofd*)

参数

该函数接受以下参数。

参数	类型	用于	描述
----	----	----	----

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	包含新智能大对象的存储结构的智能大对象规范结构
<i>flags</i>	SQL_INTEGER	输入	打开智能大对象的方式。
<i>loptr</i>	SQL_INFX_UDT_FIXED	I/O	智能大对象指针结构
<i>lofd</i>	SQL_INTEGER	输出	智能大对象文件描述符。该文件描述符仅在当前数据库连接中有效。

用法

`ifx_lo_create()` 函数执行以下步骤创建并打开智能大对象：

- 1. 创建智能大对象指针结构。
- 2. 分配指针到此结构，在 *loptr* 中返回此指针。
- 3. 从 *lospec* 指示的智能大对象规范结构中为智能大对象分配存储特征。

如果 *lospec* 为空，则 `ifx_lo_create()` 使用系统指定的存储特征。如果智能大对象规范结构存在，但是不包含存储特征，则 `ifx_lo_create()` 使用继承层次结构中的存储特征。

- 4. 以 *flags* 指定的访问模式中打开智能大对象。
- 5. 关联智能大对象和当前连接。

关闭连接时，数据库服务器将释放任何引用计数为零的智能大对象。引用数表示引用智能大对象的数据库列数。

- 6. 返回标识智能大对象的文件描述符。

数据库服务器使用调用 `ifx_lo_create()` 建立的缺省参数来确定是否锁定或记录智能大对象上后续的操作。

7. 2. 5 `ifx_lo_def_create_spec()` 函数

`ifx_lo_def_create_spec()` 函数创建智能大对象规范结构。

语法

`ifx_lo_def_create_spec(lospec)`

参数

该函数接受以下参数：

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	I/O	智能大对象规范结构

用法

`ifx_lo_def_create_spec()` 函数创建智能大对象结构并初始化字段为空。如果不更改这些值，则空值告知数据库服务器使用系统指定的缺省值来存储智能大对象的存储特征。

7. 2. 6 ifx\_lo\_open() 函数

ifx\_lo\_open() 函数打开智能大对象。

语法

ifx\_lo\_open(*lofd*, *loptr*, *flags*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输出	智能大对象文件描述符。该文件描述符只在当前数据库连接中有效。
<i>loptr</i>	SQL_INFX_UDT_FIXED	输入	智能大对象指针结构
<i>flags</i>	SQL_INTEGER	输入	打开智能大对象的模式。

用法

ifx\_lo\_open() 函数执行以下步骤来打开智能大对象：

1. 以 *flags* 指定的访问模式打开 *loptr* 智能大对象。
2. 将查找位置设置为字节零。
3. 锁定智能大对象。

**重要：** 数据库服务器不会检查智能大对象的访问权限。您的应用程序必须确保用户或应用程序可信。

如下表所述，访问模式确定锁的类型。

访问模式	锁的类型
脏读	没有锁
只读	共享锁
只写， 写/附加，或读/写	更新锁。当对智能大对象调用 ifx_lo_write() 或 ifx_lo_writewithseek() 时，数据库服务器会将锁升级为互斥锁。

当当前连接终止时，数据库服务器失去此锁。数据库服务器在下一次调用需要锁的函数时再次获取该锁。

另外，可以使用 BEGIN WORK 事务块，将 COMMIT WORK 或 ROLLBACK WORK 语句放到需要使用锁的最后一语句之后。

1. 分配具有当前连接的智能大对象。  
当关闭连接时，数据库服务器将释放任何引用计数零的智能大对象。引用数表示引用智能大对象的数据库列数。
2. 返回标识智能大对象的文件描述符。

数据库服务器使用调用 ifx\_lo\_open() 建立的缺省参数来确定是否锁定或记录智能大对象上后续的操作。

7. 2. 7 ifx\_lo\_read() 函数

ifx\_lo\_read() 函数从打开的智能大对象读取数据。

语法

ifx\_lo\_read(lofd, buf)

参数

该函数接受以下参数。

函数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输入	智能大对象文件描述符
<i>buf</i>	SQL_CHAR	输出	指向函数将读取数据的字符缓冲区的指针

用法

ifx\_lo\_read() 函数从打开的智能大对象读取数据。该读取从 lofd 当前的查找位置开始。可以调用 ifx\_lo\_tell() 获取当前查找位置。

ifx\_lo\_read() 函数读取 cbValueMax 字节数据。cbValueMax 是 SQLBindParameter() 和 SQLBindCol() 的输入参数。buf 或 cbValueMax 的大小不能超过 2 千兆字节。要读取大于 2 千兆字节的智能大对象，请在 2 千兆字节的 chunk 中读取它。ifx\_lo\_read() 函数将这些数据读入 buf 指向的用户定义缓冲区。

如果 SQLBindParameter() 或 SQLBindCol() 返回 SQL\_SUCCESS，则 pcbValue（这些函数的参数）包含函数从智能大对象中读取的字节数。如果 SQLBindParameter() 或 SQLBindCol() 返回 SQL\_SUCCESS\_WITH\_INFO，则 pcbValue 包含从智能大对象中读取的字节数。

7. 2. 8 ifx\_lo\_readwithseek() 函数

ifx\_lo\_readwithseek() 函数执行查找操作，然后从打开的智能大对象中读取数据。

语法

ifx\_lo\_readwithseek(lofd, buf, offset, whence)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输入	智能大对象文件描述符
<i>buf</i>	SQL_CHAR	输出	指向函数将读取数据的字符缓冲区的指针
<i>offset</i>	SQL_BIGINT	输入	从起始查找位置偏移的偏移量，以字节为单位。使用 SQL_C_LONG 或 SQL_C_SHORT，而不是使用缺省的

参数	类型	用于	描述
			GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 offset。
<i>whence</i>	SQL_INTEGER	输入	开始查找位置。可能值为： LO_SEEK_CUR 当前查找位置在智能大对象中的位置 LO_SEEK_END 智能大对象的末尾位置 LO_SEEK_SET 智能大对象的起始位置

### 用法

`ifx_lo_readwithseek()` 函数执行查找操作并读取来自智能大对象的数据。该读取从 `offset` 和 `whence` 参数指定的查找位置开始。

`ifx_lo_readwithseek()` 函数读取数据的 `cbValueMax` 字节。`cbValueMax` 是 `SQLBindParameter()` 和 `SQLBindCol()` 的输入参数。`buf` 或 `cbValueMax` 的大小不能超过 2 GB。要读取大于 2 千兆字节的智能大对象，请在 2-GB chunk 中读取。

`ifx_lo_readwithseek()` 函数将数据读到 `buf` 指向的用户定义的缓冲区。

如果 `SQLBindParameter()` 或 `SQLBindCol()` 返回 `SQL_SUCCESS`，则 `pcbValue`（这些函数的一个参数）包含函数从智能大对象中读取的字节数。如果 `SQLBindParameter()` 或 `SQLBindCol()` 返回 `SQL_SUCCESS_WITH_INFO`，则 `pcbValue` 包含可用于从智能大对象读取的字节数。

### 7.2.9 ifx\_lo\_seek() 函数

`ifx_lo_seek()` 函数设置打开的智能大对象上下一次读取或写入操作的文件位置。

#### 语法

`ifx_lo_seek(lofd, offset, whence, seek_pos)`

#### 参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输入	智能大对象文件描述符
<i>offset</i>	SQL_BIGINT	输入	从起始查找位置偏移的偏移量，以字节为单位。使用 SQL_C_LONG 或 SQL_C_SHORT，而不是使用缺省的 GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 offset。
<i>whence</i>	SQL_INTEGER	输入	开始查找位置。可能值为：



参数	类型	用于	描述
			LO_SEEK_CUR 当前查找位置在智能大对象中的位置 LO_SEEK_END 智能大对象的末尾位置 LO_SEEK_SET 智能大对象的起始位置
<i>seek_pos</i>	SQL_BIGINT	I/O	新的查找位置。使用 SQL_C_LONG，而不是使用缺省的 GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 seek_pos。

用法

ifx\_lo\_seek() 函数将 lofd 的查找位置设置为 offset 和 whence 参数指示的查找位置。

7. 2. 10 ifx\_lo\_specget\_estbytes() 函数

ifx\_lo\_specget\_estbytes() 函数从智能大对象规范结构中获取估计的字节数。

语法

ifx\_lo\_specget\_estbytes(*lospec*, *estbytes*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构
<i>estbytes</i>	SQL_BIGINT	输出	智能大对象的估计最终大小，以字节为单位。此估计值是智能大对象优化程序的优化提示。使用 SQL_C_LONG，而不是使用缺省的 GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 estbytes。

用法

ifx\_lo\_specget\_estbytes() 函数获取智能大对象规范结构的估计的字节数。

7. 2. 11 ifx\_lo\_specget\_extsz() 函数

fx\_lo\_specget\_extsz() 函数从智能大对象规范结构获取分配的 extent。

语法

`ifx_lo_specget_extsz(lospec, extsz)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构
<i>extsz</i>	SQL_INTEGER	输出	智能大对象的 Extent 大小，以字节为单位。该值是当数据库服务器写入超出当前范围的末尾时，为智能大对象分配的分配 Extent 的大小。该值会覆盖数据库服务器估计的 extent 大小。

用法

`ifx_lo_specget_extsz()` 函数从智能大对象规范结构获取分配的 extent。

7. 2. 12 `ifx_lo_specget_flags()` 函数

`ifx_lo_specget_flags()` 函数从智能大对象获取创建时间标记。

语法

`ifx_lo_specget_flags(lospec, flags)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构
<i>flags</i>	SQL_INTEGER	输出	创建时间标记。

7. 2. 13 `ifx_lo_specget_maxbytes()` 函数

`ifx_lo_specget_maxbytes()` 函数获取智能大对象规范结构的最大字节数。

语法

`ifx_lo_specget_maxbytes(lospec, maxbytes)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构

参数	类型	用于	描述
<i>maxbytes</i>	SQL_BIGINT	输入	智能大对象的最大大小, 以字节为单位。使用 SQL_C_LONG 而不是使用 GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 maxbytes。

用法

`ifx_lo_specget_maxbytes()` 函数获取智能大对象规范结构的最大字节数。

7. 2. 14 `ifx_lo_specget_sbospace()` 函数

`ifx_lo_specget_sbospace()` 函数从智能大对象规范结构获取 `sbospace` 的名称。

语法

`ifx_lo_specget_sbospace(lospec, sbospace)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构
<i>sbospace</i>	SQL_CHAR	输出	智能大对象的 <code>sbospace</code> 名称。 <code>sbospace</code> 名称可以达到 18 字节长并且必须是以空字符串为终止。

用法

`ifx_lo_specget_sbospace()` 函数返回存储智能大对象的 `sbospace` 名称。该函数将 (`pcbValue-1`) 字节复制到 `sbospace` 缓冲区并确保它以空字符串终止。`pcbValue` 是 `SQLBindParameter()` 和 `SQLBindCol()` 的参数。

7. 2. 15 `ifx_lo_specset_estbytes()` 函数

`ifx_lo_specset_estbytes()` 函数设置智能大对象规范结构中估计的字节数。

语法

`ifx_lo_specset_estbytes(lospec, estbytes)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构
<i>estbytes</i>	SQL_BIGINT	输入	<p>智能大对象的估计最终值，以字节为单位。此估计值是智能大对象优化程序的优化提示。该值不能超过 2 千兆字节。</p> <p>如果在创建新的智能大对象时，未指定 <i>estbytes</i> 值，则数据库服务器从存储特征的继承层次结构中获取该值。</p> <p>除非您指定智能大对象的估计大小，否则不要更改此系统值。如果您设置了智能大对象的估计大小，则不要指定的值不要高出智能大对象的最终大小太多。否则，数据库服务器可能分配未使用的存储。</p> <p>使用 SQL_C_LONG 或 SQL_C_SHORT，而不是使用缺省的 GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 <i>estbytes</i>。</p>

用法

`ifx_lo_specset_estbytes()` 函数设置智能大对象规范结构中估计的字节数。

7. 2. 16 `ifx_lo_specset_extsz()` 函数

`ifx_lo_specset_extsz()` 函数设置智能大对象规范结构中的分配 Extent 大小。

语法

`ifx_lo_specset_extsz(lospec, extsz)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构
<i>extsz</i>	SQL_INTEGER	输入	<p>智能大对象的 Extent 大小，以字节为单位。该值是当数据库服务器写入超出当前范围的末尾时，为智能大对象分配的分配 Extent 的大小。该值会覆盖数据库服务器估计的 extent 大小。</p> <p>如果在创建新的智能大对象时，未指定 extsz 值，则数据库服务器尝试根据智能大对象的历史操作和从存储特征获取的继承层次结构的其它存储特征（例如，最大字节数）来优化 extent 大小。</p> <p>除非您知道智能大对象的分配的 extent 大小，否则请不要更改此系统值。只有遇到苛刻存储分片的应用程序才能设置分配 extent 大小。有关这些应用程序，请确保您确切知道 extent 智能大对象的字节数。</p>

用法

`ifx_lo_specset_extsz()` 函数设置智能大对象规范结构中的分配 Extent 大小。

7. 2. 17 `ifx_lo_specset_flags()` 函数

`ifx_lo_specset_flags()` 函数设置智能大对象规范结构中的创建时间标志。

语法

`ifx_lo_specset_flags(lospec, flags)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构
<i>flags</i>	SQL_INTEGER	输入	创建时间标志

用法

`ifx_lo_specset_flags()` 设置智能大对象规范结构中的创建时间标志。

7. 2. 18 `ifx_lo_specset_maxbytes()` 函数

`ifx_lo_specset_maxbytes()` 函数设置智能大对象规范结构中的最大字节数。

语法

`ifx_lo_specset_maxbytes(lospec, maxbytes)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构。
<i>maxbytes</i>	SQL_BIGINT	输入	智能大对象的最大大小，以字节为单位。该值不能超过 2 千兆字节。使用 <code>SQL_C_LONG</code> 或 <code>SQL_C_SHORT</code> 而不是使用 GBase 8s ODBC Driver C 数据类型 <code>SQL_C_CHAR</code> 作为 <code>maxbytes</code> 。

用法

`ifx_lo_specset_maxbytes()` 函数设置智能大对象规范结构中的最大字节数。

7. 2. 19 `ifx_lo_specset_sbospace()` 函数

`ifx_lo_specset_sbospace()` 函数设置智能大对象规范结构中的 `sbospace` 名称。

语法

`ifx_lo_specset_sbospace(lospec, sbospace)`

参数

该函数接受以下参数。

参数	类型	用于	描述
----	----	----	----

参数	类型	用于	描述
<i>lospec</i>	SQL_INFX_UDT_FIXED	输入	智能大对象规范结构。
<i>sbspace</i>	SQL_CHAR	输入	智能大对象的 <i>sbspace</i> 名称。  <i>sbspace</i> 名称可以达到 18 字节长并且必须是以空字符串为终止。如果在创建智能大对象时，未指定 <i>sbspace</i> ，则数据库服务器从列信息或从 <i>onconfig</i> 文件的 <b>SBSPACENAME</b> 参数获取 <i>sbspace</i> 名称。

用法

`ifx_lo_specset_sbspace()` 函数使用 `pcbValue` 确定 *sbspace* 名称的长度。`pcbValue` 是 `SQLBindParameter()` 和 `SQLBindCol()` 的一个参数。

7. 2. 20 `ifx_lo_stat()` 函数

`ifx_lo_stat()` 函数初始化智能大对象状态结构。

语法

`ifx_lo_stat(lofd, lostat)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输入	智能大对象文件描述符
<i>lostat</i>	SQL_INFX_UDT_FIXED	I/O	智能大对象状态结构

用法

在调用 `ifx_lo_stat()` 之前，请调用 `SQLGetInfo()` 获取智能大对象状态结构的大小。使用该大小为结构分配内存。

`ifx_lo_stat()` 函数分配智能大对象状态结构并使用智能大对象的状态信息初始化它。

7. 2. 21 `ifx_lo_stat_atime()` 函数

`ifx_lo_stat_atime()` 函数检索智能大对象的最近一次访问时间。

语法

`ifx_lo_stat_atime(lostat, atime)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lostat</i>	SQL_INFX_UDT_FIXED	输入	智能大对象状态结构
<i>atime</i>	SQL_INTEGER	输出	智能大对象最近一次访问的时间，以秒为单位。数据库服务器仅当为智能大对象设置 LO_KEEP_LASTACCESS_TIME 标志时，才会维护最近访问时间。

用法

`ifx_lo_stat_atime()` 函数检索智能大对象的最近一次访问时间。

7. 2. 22 `ifx_lo_stat_cspec()` 函数

`ifx_lo_stat_cspec()` 函数检索智能大对象结构。

语法

`ifx_lo_stat_cspec(lostat, lospec)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lostat</i>	SQL_INFX_UDT_FIXED	输入	智能大对象状态结构
<i>lospec</i>	SQL_INFX_UDT_FIXED	输出	智能大对象规范结构

用法

`ifx_lo_stat_cspec()` 函数检索智能大对象规范结构，并返回指向结构的指针。

7. 2. 23 `ifx_lo_stat_ctime()` 函数

`ifx_lo_stat_ctime()` 函数检索智能大对象最近一次更改的时间。

语法

`ifx_lo_stat_ctime(lostat, ctime)`

参数

该函数接受以下参数。

参数	类型	用于	描述
----	----	----	----



参数	类型	用于	描述
<i>lostat</i>	SQL_INFX_UDT_FIXED	输入	智能大对象状态结构
<i>ctime</i>	SQL_INTEGER	输出	智能大对象最近一次更改的时间，以秒为单位。最近一次更改时间包括存储特征的修改，引用次数的更改和写入智能大对象。

用法

`ifx_lo_stat_ctime()` 函数检索智能大对象最近一次更改的时间。

7. 2. 24 `ifx_lo_stat_refcnt()` 函数

`ifx_lo_stat_refcnt()` 函数检索智能大对象的引用次数。

语法

`ifx_lo_stat_refcnt(lostat, refcount)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lostat</i>	SQL_INFX_UDT_FIXED	输入	智能大对象状态结构
<i>refcount</i>	SQL_INTEGER	输出	智能大对象的引用次数。该值是数据库列引用智能大对象的次数。

用法

`ifx_lo_stat_refcnt()` 函数检索智能大对象的引用次数。

数据库服务器可以在智能大对象引用次数为零时，或发生以下之一的情况时，移除智能大对象，重新利用分配给它的资源：

- 提交引用计数递减为零的事务。
- 创建智能大对象的连接终止，但引用计数不增加。

将智能大对象指针结构存储在一行中时，数据库服务器将增加一个引用计数。

7. 2. 25 `ifx_lo_stat_size()` 函数

`ifx_lo_stat_size()` 函数检索智能大对象的大小，

语法

ifx\_lo\_stat\_size(*lostat*, *size*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lostat</i>	SQL_INFX_UDT_FIXED	输入	智能大对象状态结构
<i>size</i>	SQL_BIGINT	输出	智能大对象的大小，以字节为单位。该值不能超过 2 千兆字节。使用 SQL_C_LONG 而不是使用缺省的 GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 <i>size</i> 。

用法

ifx\_lo\_stat\_size() 函数检索智能大对象的大小。

7. 2. 26 ifx\_lo\_tell() 函数

ifx\_lo\_tell() 函数检索打开的智能大对象的当前文件或查找位置。

语法

ifx\_lo\_tell(*lofd*, *seek\_pos*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输入	智能大对象文件描述符
<i>seek_pos</i>	SQL_BIGINT	I/O	新的查找位置，智能大对象上的下一次读或写操作的偏移量。使用 SQL_C_LONG 而不是使用缺省的GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 <i>seek_pos</i> 。

用法

ifx\_lo\_tell() 函数检索打开的智能大对象的当前文件或查找位置。

该函数当智能大对象长达 2 千兆字节时也能正常运行。

7. 2. 27 ifx\_lo\_truncate() 函数

ifx\_lo\_truncate() 函数在指定位置截断智能大对象。

语法

ifx\_lo\_truncate(*lofd*, *offset*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输入	智能大对象文件描述符
<i>offset</i>	SQL_BIGINT	输入	智能大对象的末尾。如果该值超出智能大对象的末尾，则函数扩展智能大对象。如果该值小于智能大对象的末尾，则数据库服务器将回收从偏移位置到智能大对象的末尾的所有存储。 使用 SQL_C_LONG 或 SQL_C_SHORT，而不是使用缺省的 GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 offset。

用法

ifx\_lo\_truncate() 函数设置将智能大对象的末尾设置为 offset 参数指定的位置。

7. 2. 28 ifx\_lo\_write() 函数

ifx\_lo\_write() 函数将数据写入智能大对象。

语法

ifx\_lo\_write(*lofd*, *buf*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输入	智能大对象文件描述符
<i>buf</i>	SQL_CHAR	输入	包含函数写入到智能大对象中的数据的缓冲区。该缓冲区的大小不能超过 2 千兆字节。

用法

ifx\_lo\_write() 函数将数据写入智能大对象。写入从 lofd 的当前查找位置开始。可以调用 ifx\_lo\_tell() 获取当前查找位置。

ifx\_lo\_write() 函数写入 cbValueMax 字节数据。cbValueMax 是 SQLBindParameter() 和 SQLBindCol() 的输入参数。buf 或 cbValueMax 的大小不能超过 2 GB。要写入一个大于 2 千兆字节的智能大对象，将其写入 2-GB chunk。ifx\_lo\_write() 函数获取从用户定义的缓冲区到 buf 指向的数据。

如果 SQLExecDirect() 或 SQLExecute() 返回 SQL\_SUCCESS\_WITH\_INFO，则数据库服务器将写入少于 cbValueMax 字节的数据到智能大对象，pcbValue（这些函数的一个参数）包含函数写入的字节数。当 sbospace 超出空间范围时，会发生此情况。

7. 2. 29 ifx\_lo\_writewithseek() 函数

ifx\_lo\_writewithseek() 函数执行查找操作然后将数据写入到打开的智能大对象。

语法

ifx\_lo\_writewithseek(*lofd, buf, offset, whence*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>lofd</i>	SQL_INTEGER	输入	智能大对象文件描述符
<i>buf</i>	SQL_CHAR	输入	包含函数写入到智能大对象中的数据的缓冲区。该缓冲区的大小不能超过 2 千兆字节。
<i>offset</i>	SQL_BIGINT	输入	起始查找位置的偏移量，以字节为单位。 使用 SQL_C_LONG 或 SQL_C_SHORT，而不是使用缺省的 GBase 8s ODBC Driver C 数据类型 SQL_C_CHAR 作为 offset。
<i>whence</i>	SQL_INTEGER	输入	开始查找位置。可能值为： LO_SEEK_CUR 当前查找位置在智能大对象中的位置 LO_SEEK_END 智能大对象的末尾位置 LO_SEEK_SET 智能大对象的起始位置

用法

ifx\_lo\_writewithseek() 函数执行查找操作并将数据写入到自智能大对象。该写入从 offset 和 whence 参数指定的 lofd 查找位置开始。

ifx\_lo\_writewithseek() 函数写入 cbValueMax 字节数据。cbValueMax 是 SQLBindParameter() 和 SQLBindCol() 的一个输入参数。buf 或 cbValueMax 的大小不能超过 2 GB。要写入大于 2 GB 的智能大对象，将它写入到 2-GB chunk。  
ifx\_lo\_writewithseek() 函数从 buf 指向的用户定义的缓冲区获取数据。

如果 SQLExecDirect() 或 SQLExecute() 返回 SQL\_SUCCESS\_WITH\_INFO，则数据库服务器写入少于 cbValueMax 字节的数据到智能大对象，pcbValue（这些函数的一个参数）包含函数写入的字节数。当 sbospace 超出空间范围时，发生此情况。

## 7.3 行和集合的函数

本节描述 GBase 8s ODBC Driver 提供用于行和集合的客户端函数。

### 7.3.1 ifx\_rc\_count() 函数

ifx\_rc\_count() 函数返回行或集合中元素的数量。

#### 语法

```
ifx_rc_count(rowcount, rchandle)
```

#### 参数

该函数接受以下参数。

参数	类型	用于	描述
rowcount	SQL_SMALLINT	输出	行或集合中元素的数量
rchandle	HINFX_RC	输入	行或集合存取的句柄

#### 用法

ifx\_rc\_count() 函数返回行或集合中元素的数量。

### 7.3.2 ifx\_rc\_create() 函数

ifx\_rc\_create() 函数为行或集合创建缓冲区。

#### 语法

```
ifx_rc_create(rchandle, typespec)
```

#### 参数

该函数接受以下参数。

参数	类型	用于	描述
----	----	----	----

参数	类型	用于	描述
<i>rchandle</i>	HINFX_RC	输出	行或集合缓冲区的句柄
<i>typespec</i>	SQL_CHAR	输入	缓冲区的类型规范。参见下表。

下表描述了 *typespec* 参数的语法。

缓冲区的类型	语法	示例
不固定型集合	COLLECTION	COLLECTION
固定型集合	COLLECTION {SET   MULTISET   LIST} (type not null) 或 {SET   MULTISET   LIST (type not null) type 是集合中元素的 GBase 8s SQL 数据类型	COLLECTION SET (int not null) 或 SET (int not null)
不固定型集合	ROW	ROW
固定型集合	ROW [ “name” ] (field_id type [, field_id type, ...]) 其中： 7. name 是整个行的可选名称 8. field_id 是字段的名称 9. type 是字段的 GBase 8s SQL 数据类型	ROW “employee_t” (name char(255), id_num int, dept int)

用法

*ifx\_rc\_create()* 函数为行和集合分片内存，并返回一个句柄到缓冲区。下表描述了函数初始化缓冲区的方式。

缓冲区的类型	行或集合的初始值	行或集合的内容的初始值
Fixed-type collection	非空	空
Fixed-type row	非空	每个值都为空
Unfixed-type collection	空	空
Unfixed-type row	空	空

对于行，函数将第一个元素设置为查找位置。空集合缓冲区不具有查找位置。

7. 3. 3 ifx\_rc\_delete() 函数

ifx\_rc\_delete() 函数从集合删除元素。

语法

ifx\_rc\_delete(*rchandle*, *action*, *jump*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>rchandle</i>	HINFX_RC	输入	集合缓冲区的句柄
<i>action</i>	SQL_SMALLINT	输入	元素相对于查找位置的位置。值可能为： <ul style="list-style-type: none"><li>SQL_INFX_RC_ABSOLUTE：：元素 <i>jump</i>，缓冲区中的第一个元素是元素 1</li><li>SQL_INFX_RC_CURRENT：当前元素</li><li>SQL_INFX_RC_FIRST：第一个元素</li><li>SQL_INFX_RC_LAST：最后一个元素</li><li>SQL_INFX_RC_NEXT：下一个元素</li><li>SQL_INFX_RC_PRIOR：上一个元素</li><li>SQL_INFX_RC_RELATIVE：： <i>jump</i> 跳过当前元素的元素</li></ul>
<i>jump</i>	SQL_SMALLINT	输入	当 <i>action</i> 是 SQL_INFX_RC_ABSOLUTE 或 SQL_INFX_RC_RELATIVE 的偏移量

用法

ifx\_rc\_delete() 函数从 *action* 和 *jump* 指定的位置删除集合中的元素。该函数将查找位置设置为被删除的值的位位置。从行中删除元素是不可能的。

7. 3. 4 ifx\_rc\_describe() 函数

ifx\_rc\_describe() 函数返回有关行或集合数据类型的信息或行或集合中元素的信息。

语法

ifx\_rc\_describe(*rchandle*, *fieldnum*, *fieldname*, *typecode*,  
*columnsize*, *decdigits*, *nullable*, *typename*, *typeowner*)

参数

该函数接受以下参数。

参数	类型	用于	描述
----	----	----	----

参数	类型	用于	描述
<i>rchandle</i>	HINFX_RC	输入	行或集合缓冲区的句柄
<i>fieldnum</i>	SQL_SMALLINT	输入	字段数。如果该值为 0，则函数返回整个行或集合的信息。对于集合，任何不是 0 的值会使函数返回集合中元素的信息。对于行，该值指定函数返回信息的元素。
<i>fieldname</i>	SQL_CHAR	输出	字段名称。该函数仅当元素在行中时返回此值。
<i>typecode</i>	SQL_SMALLINT	输出	元素的 GBase 8s ODBC Driver SQL 数据类型
<i>columnsize</i>	SQL_INTEGER	输出	列大小。对于字符元素，该值是列的大小，以字节为单位。对于数字元素，该值是精度。对于其它数据类型，该函数不会返回值。
<i>decdigits</i>	SQL_SMALLINT	输出	小数点位数。对于数字元素，该值是小数点后小数位数。对于其它数据类型，该函数不会返回值。
<i>nullable</i>	SQL_SMALLINT	输出	NULL 指示符。值可能为： SQL_NO_NULLS SQL_NULLABLE
<i>typename</i>	SQL_CHAR	输出	类型名称。对于已命名的行，该值是行的名称。对于集合和未命名行，此函数不会返回值。
<i>typeowner</i>	SQL_CHAR	输出	类型所有者。该值是数据类型的所有者的名称。该名称的长度不能超出 18 个字符。

## 用法

`ifx_rc_describe()` 函数返回有关行或集合数据类型的信息或行或集合中元素的信息。对于集合中的元素，该信息与集合中所有元素的信息一致。该函数不会更改查找位置。

### 7.3.5 `ifx_rc_fetch()` 函数

`ifx_rc_fetch()` 函数检索行或集合中元素的值。



语法

`ifx_rc_fetch(result, rchandle, action, jump)`

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>result</i>	元素的数据类型	输出	检索到的值
<i>rchandle</i>	HINFX_RC	输入	行或集合缓冲区的句柄
<i>action</i>	SQL_SMALLINT	输入	元素相对于查找位置的位置。值可能为： <ul style="list-style-type: none"><li>SQL_INFX_RC_ABSOLUTE: : 元素 <i>jump</i> , 缓冲区中的第一个元素是元素 1</li><li>SQL_INFX_RC_CURRENT: 当前元素</li><li>SQL_INFX_RC_FIRST: 第一个元素</li><li>SQL_INFX_RC_LAST: 最后一个元素</li><li>SQL_INFX_RC_NEXT: 下一个元素</li><li>SQL_INFX_RC_PRIOR: 上一个元素</li><li>SQL_INFX_RC_RELATIVE: : <i>jump</i> 跳过当前元素的元素</li></ul>
<i>jump</i>	SQL_SMALLINT	输入	当 <i>action</i> 是 SQL_INFX_RC_ABSOLUTE 或 SQL_INFX_RC_RELATIVE 时, 偏移

用法

`ifx_rc_fetch()` 函数检索由 `action` 和 `jump` 指定的元素的值, 并在 `result` 中返回值。该函数将查找位置设置为刚获取的值的位位置。

7. 3. 6 `ifx_rc_free()` 函数

`ifx_rc_free()` 函数释放行或集合句柄。

语法

`ifx_rc_free(rchandle)`

参数

该函数接受以下参数,

参数	类型	用于	模式
<i>rchandle</i>	HINFX_RC	输入	行或集合缓冲区的句柄

用法

ifx\_rc\_free() 函数释放与集合或行句柄相关联的资源，并释放此句柄。

7. 3. 7 ifx\_rc\_insert() 函数

ifx\_rc\_insert() 函数将新的值插入到集合中。

语法

```
ifx_rc_insert(rchandle, value, action, jump)
```

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>rchandle</i>	HINFX_RC	输入	集合缓冲区的句柄
<i>value</i>	元素的数据类型	输入	要插入的值
<i>action</i>	SQL_SMALLINT	输入	元素相对于查找位置的位置。值可能为： <ul style="list-style-type: none"><li>● SQL_INFX_RC_ABSOLUTE：：元素 <i>jump</i> ，缓冲区中的第一个元素是元素 1</li><li>● SQL_INFX_RC_CURRENT：当前元素</li><li>● SQL_INFX_RC_FIRST：第一个元素</li><li>● SQL_INFX_RC_LAST：最后一个元素</li><li>● SQL_INFX_RC_NEXT：下一个元素</li><li>● SQL_INFX_RC_PRIOR：上一个元素</li><li>● SQL_INFX_RC_RELATIVE：： <i>jump</i> 跳过当前元素的元素</li></ul>
<i>jump</i>	SQL_SMALLINT	输入	当 <i>action</i> 是 SQL_INFX_RC_ABSOLUTE 或 SQL_INFX_RC_RELATIVE 时，偏移

用法

ifx\_rc\_insert() 函数将新的元素插入到 *action* 和 *jump* 指定之前的位置。函数将查找位置设置为已插入的值的位​​置。无法向行插入新的元素。

下表描述了每种集合类型的允许的插入操作。

集合的类型	允许插入的位置
List	缓冲区中的任何位置
Set 或 multiset	缓冲区的末尾

如果由 *action* 和 *jump* 值的查找位置超出缓冲区的末尾，则 ifx\_rc\_insert() 将新的元素附加到缓冲区的末尾。同样地，如果 *action* 和 *jump* 指定查找位置在缓冲区的起始位置之前，

则 ifx\_rc\_insert() 在缓冲区的开头插入新的元素。如果 action 指定一个插入的指针而不是 set 或 multiset 的末尾，则 ifx\_rc\_insert() 失败。

例如，如果 action 是 SQL\_INFX\_RC\_LAST，则该函数在最后一个元素之前插入新值。要附加新的元素，请采取以下操作：

- 将查找位置设置为缓冲区的末尾，并将 action 设置为 SQL\_INFX\_RC\_NEXT。
- 将 action 设置为 SQL\_INFX\_RC\_ABSOLUTE 或 SQL\_INFX\_RC\_RELATIVE ，并将 jump 设置为超出缓冲区末尾的值。

要在缓冲区的开始位置插入新的值，请将 action 设置为 SQL\_INFX\_RC\_FIRST。

7. 3. 8 ifx\_rc\_isnull() 函数

ifx\_rc\_isnull() 函数返回指示行或集合是否为空的值。

语法

ifx\_rc\_isnull(*nullflag*, *rchandle*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>nullflag</i>	SQL_SMALLINT	输出	指示行或集合是否为空的标志。值可能为： <ul style="list-style-type: none"><li>• TRUE</li><li>• FALSE</li></ul>
<i>rchandle</i>	HINFX_RC	输入	行或集合的句柄

用法

ifx\_rc\_isnull() 函数返回指示行或集合是否为空的值。

7. 3. 9 ifx\_rc\_setnull() 函数

ifx\_rc\_setnull() 函数将行或集合设置为空。

语法

ifx\_rc\_setnull(*rchandle*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>rchandle</i>	HINFX_RC	输入	行或集合缓冲区的句柄

用法

ifx\_rc\_setnull() 函数将行或集合设置为空。ifx\_rc\_setnull() 函数不会将行或集合中的每个元素设置为空。

7. 3. 10 ifx\_rc\_typespec() 函数

ifx\_rc\_typespec() 函数返回行或集合的类型规范。

语法

ifx\_rc\_typespec(*typespec*, *rchandle*, *flag*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>typespec</i>	SQL_CHAR	输出	类型规范。该值的格式与 ifx_rc_create() 的规范语法相同。
<i>rchandle</i>	HINFX_RC	输入	行或集合缓冲区的句柄
<i>flag</i>	SQL_SMALLINT	输入	指定是返回当前还是原始类型规范的标志。如果该值为 TRUE，则函数返回原始类型规范。否则，该函数返回当前类型规范。

用法

ifx\_rc\_typespec() 函数返回行或集合的类型规范。

7. 3. 11 ifx\_rc\_update() 函数

ifx\_rc\_update() 函数更新行或集合中的元素的值。

语法

ifx\_rc\_update(*rchandle*, *value*, *action*, *jump*)

参数

该函数接受以下参数。

参数	类型	用于	描述
<i>rchandle</i>	HINFX_RC	输入	行或集合缓冲区的句柄
<i>value</i>	元素的数据类型	输入	更新的元素的值
<i>action</i>	SQL_SMALLINT	输入	元素相对于查找位置的位置。值可能为： <ul style="list-style-type: none"><li>SQL_INFX_RC_ABSOLUTE：：元素 <i>jump</i> ，缓冲区中的第一个元素是元素 1</li><li>SQL_INFX_RC_CURRENT：当前元素</li></ul>

参数	类型	用于	描述
			<ul style="list-style-type: none"><li>• SQL_INFX_RC_FIRST: 第一个元素</li><li>• SQL_INFX_RC_LAST: 最后一个元素</li><li>• SQL_INFX_RC_NEXT: 下一个元素</li><li>• SQL_INFX_RC_PRIOR: 上一个元素</li><li>• SQL_INFX_RC_RELATIVE : : jump 跳过当前元素的元素</li></ul>
<i>jump</i>	SQL_SMALLINT	输入	当 action 是 SQL_INFX_RC_ABSOLUTE 或 SQL_INFX_RC_RELATIVE 时，偏移

用法

ifx\_rc\_update() 函数更改 action 和 jump 指定的位置之前的元素的值。该函数将查找位置设置为已更改的值的位位置。

## 8 提高应用程序性能

这些主题描述提高 GBase 8s ODBC Driver应用程序的性能的方法。

### 8.1 在数据传输过程中进行错误检查

IFX\_LOB\_XFERSIZE 环境变量用于指定在检查是否发生错误之前,从客户端应用程序传输到数据库服务器的 CLOB 或 BLOB 中的千字节数。

每当传输指定的千字节数据时都会发生错误检查。如果发生错误,则不会发送剩余的数据,并报告错误。如果没有发生错误,则继续文件传输直到结束。

IFX\_LOB\_XFERSIZE 值的范围是 1 到 9223372036854775808 千字节。在客户端设置 IFX\_LOB\_XFERSIZE 环境变量。

有关 IFX\_LOB\_XFERSIZE 的更多信息,请参阅《GBase 8s SQL 指南: 参考》。

### 8.2 在 ODBC 中启用分隔标识符

缺省情况下,分隔标识符在通过 ODBC 连接时被禁用。

有三种方式启用它们,按优先级递减的顺序如下所示:

#### DELIMIDENT 连接字符串关键字

如果使用连接字符串连接,可以将关键字 DELIMIDENT 设置为启用或禁用分隔标识符。如果关键字设置为 y则为连接启用分隔标识符。如果关键字设置为 n 则为连接禁用分隔标识符。如果关键字存在但是没有设置值,则对是否启用分隔标识符没有影响。

例如,该连接字符串使用数据源(DSN) mydsn连接,并为此连接启用分隔标识符。

```
"DSN=mydsn;DELIMIDENT=y;"
```

该连接字符串还使用 DSN mydsn连接,但是对是否使用分隔标识符没有影响。

```
"DSN=mydsn;DELIMIDENT=;"
```

在连接字符串中设置 DELIMIDENT 关键字会覆盖任何启用或禁用分隔标识符的连接属性或环境变量。

#### SQL\_INFX\_ATTR\_DELIMIDENT 连接属性

可以在连接之前通过设置 SQL\_INFX\_ATTR\_DELIMIDENT 连接属性来启用或禁用给出连接的分隔标识符。SQL\_INFX\_ATTR\_DELIMIDENT 连接属性接受下表列出的值。

表 1. SQL\_INFX\_ATTR\_DELIMIDENT 连接属性允许的值

值	影响
SQL_TRUE	为此连接启用分隔标识符。

值	影响
SQL_FALSE	为此连接禁用分隔标识符。
SQL_IFX_CLEAR	清除之前的设置，以便该连接属性对是否使用分隔标识符没有影响。

例如，该调用导致在创建连接时启用分隔标识符：

```
SQLSetConnectAttr(hdbc, SQL_INFX_ATTR_DELIMIDENT, SQL_TRUE, SQL_IS_INTEGER);
```

如果该连接属性设置为 SQL\_TRUE 或 SQL\_FALSE，则该设置覆盖 DELIMIDENT 环境变量，但是不覆盖 DELIMIDENT 连接字符串关键字。

**DELIMIDENT 环境变量**

在一些 GBase 8s API 中，例如 ESQL/C，通过将 DELIMIDENT 环境变量设置为任何值来启用分隔标识符。但是，在 ODBC 中，通过将 DELIMIDENT 环境变量设置为 y 来启用分隔标识符，将它设置为 n 来禁用分隔标识符。

8.3 连接级别优化

建立到数据库的连接是一个花费时间的过程。理想情况下，应用程序在连接打开时执行尽可能多的任务。

该过程可以通过以下方式实现：

- 使用 Windows™ Driver Manager 时汇集连接
- 在同一个的连接句柄上使用多个语句句柄

此外，可以通过设置以下连接级别属性来优化应用程序性能：

- AutoCommit 优化
- 消息传输优化（OPTMSG）
- Open-Fetch-Close 优化（OPTOFC）

8.4 优化查询执行

当使用准备好的 SQL 查询时，您必须考虑几个方面。

当使用准备好的 SQL 查询时，请考虑以下几点：

- SQLExecDirect 是针对 SQL 语句的单个执行的优化。因此，它用于不重复执行的 SQL 查询。
- 在多次执行 SQL 查询的情况下，使用 SQLPrepare 和 SQLExecute提高性能。通常情况下，可以使用输入和输入参数执行此操作。

- 可以从 ODBC 应用程序调用 SPL 例程来执行某些 SQL 任务，并可以单独使用 SQL 扩展完成的任务。因为，SPL 是数据库的本地语言，SPL 例程在创建时被解析和优化，而不是在运行时，SPL 例程可以提高某些任务的性能。SPL 例程还可以减少客户端应用程序和数据库服务器之间的流量，并降低程序的复杂性。
- 当使用 GBase 8s ODBC Driver 执行具有返回值的存储过程时，在结果集上调用一个访存之前，过程返回的错误不会返回到应用程序。执行过程后，立即可以获得来自存储过程的没有返回值的错误信息。

## 8.5 插入多行

使用插入游标有效地将行插入到批表中。

要创建插入游标，通过使用 `SQLSetStmtOption` 设置 `SQL_ENABLE_INSERT_CURSOR` 属性，然后调用 `SQLParamOptions`，使用行数作为参数。可以对 `VARCHAR`、`LVARCHAR` 和不透明数据类型创建插入游标。

当打开插入游标时，在内存中创建一个缓冲区保存行。缓冲区在程序生成数据时接收数据行；然后当缓冲区为空时，将它们发送到数据库服务器。该缓冲区减少程序和数据服务器之间的流量。从而，插入速度更快。

## 8.6 自动释放游标

当应用程序使用游标时，通常会向数据库服务器发送 `FREE` 语句，以在不再需要该游标之后释放分配给游标的内存。

该语句的执行调用应用程序和数据库服务器之间的消息请求。当启用 `AUTOFREE` 时，GBase 8s ODBC Driver 保存消息请求，因为它不需要执行 `FREE` 语句。当数据库服务器关闭插入游标时，它自动释放分配给它的内存。

### 8.6.1 启用 `AUTOFREE` 功能

有两种方法启用 ODBC 应用程序的 `AUTOFREE` 功能。

使用 `SQLSetConnectAttr` 设置 `SQL_INFX_ATTR_AUTO_FREE` 属性时，可以在 C2 和 C5 之间（包括两者）的任何连接状态中进行设置，而只有当语句处于 S1（分配）状态时，才可以使用 `SQLSetStmtAttr` 设置 `SQL_INFX_ATTR_AUTO_FREE` 属性。可以通过使用 `SQLGetConnectAttr` 或 `SQLSetStmtAttr` 检索 `SQL_INFX_ATTR_AUTO_FREE` 属性的值。

可以是使用以下方法启用 ODBC 应用程序的 `AUTOFREE` 功能：

- 使用 `SQLSetConnectAttr` 设置 `SQL_INFX_ATTR_AUTO_FREE` 属性。

当您使用 `SQLSetConnectAttr` 启用该属性时，该连接的所有的新的语句都会继承该值。更改该属性的唯一方法是设置每一条语句并将它重新设置为语句的属性。连接属性缺省为 `DISABLED`。



- 使用 `SQLSetStmtAttr` 设置 `SQL_INFX_ATTR_AUTO_FREE` 属性。

### 8.6.2 AUTOFREE 功能

AUTOFREE 功能仅适用于使用 `SQLExecDirect` 执行的结果生成语句，因为它打开了由相应的 `SQLCloseCursor` 或 `SQLFreeStmt` 关闭和释放的游标。

当应用程序必须准备一次语句并执行多次时，AUTOFREE 功能不起作用。（例如，使用 `SQLPrepare` 准备然后多次调用 `SQLExecute` 执行它。）当在 `SQLExecute` 之后使用 `SQLCloseCursor` 关闭游标时，它只关闭游标但不释放数据库服务器端的游标内存。但是如果您使用具有 `SQL_CLOSE` 或 `SQL_DROP` 的 `SQLFreeStmt` 关闭游标时，它不仅关闭游标还释放此游标。在后一种情况下，会节省网络往返，但是应用程序不会执行该语句直到它重新表达它为止。

启用 AUTOFREE 后，当应用程序使用具有 `SQL_DROP` 的 `SQLCloseCursor` 或 `SQLFreeStmt` 关闭游标时，应用程序会看到网络性能的提升。

## 8.7 延迟执行 SQL PREPARE 语句

可以通过启用 deferred-PREPARE 功能来延迟 `SQLPrepare` 语句的执行。

此功能适用于应用程序执行一系列的 `SQLPrepare` 和 `SQLExecute` 语句的动态 SQL 语句。它通过在应用程序对该语句调用 `SQLExecute` 之前不向数据库服务器发送 `SQLPrepare` 语句的方式来优化到数据库服务器的往返消息的数量。

当启用 deferred-PREPARE 后，应用程序执行以下行为：

- 执行 `SQLPrepare` 不会将语句置于准备好的状态。
- 在执行语句之前，`SQLPrepare` 语句中的语法错误是未知的，因为该 SQL 语句在执行之前不会发送到数据库服务器。如果打开 `open-fetch-close` 优化功能，则直到第一次访存之前错误不会返回到客户端，因为 `open-fetch-close` 优化了 `OPEN/FETCH`，所以在第一次访存时发送 `OPEN`。
- 如果应用程序在调用 `SQLPrepare` 之后，`SQLExecute` 之前调用 `SQLColAttributes`、`SQLDescribeCol`、`SQLNumResultCols` 和 `SQLNumParams`，则始终返回 HY010（函数序列错误）。
- 如果源描述符句柄在 `SQLPrepare` 之后，但在应用程序执行 `SQLExecute` 之前调用，则 `SQLCopyDesc` 返回 HY010。
- 如果描述符句柄是一个 IRD，并且应用程序在 `SQLPrepare` 之后 `SQLExecute` 之前调用，则 `SQLGetDescField` 和 `SQLGetDescRec` 返回 HY010。

可以使用以下方式启用 ODBC 应用程序的 deferred-PREPARE 功能：

- 使用 `SQLSetConnectAttr` 设置 `SQL_INFX_ATTR_DEFERRED_PREPARE` 属性。

- 当使用 `SQLSetConnectAttr` 启用该属性时，所有为此连接新分配的语句都会继承该属性值。更改该属性的唯一方法是设置每一条语句并将它重新设置为语句的属性。连接属性缺省为 `DISABLED`。
- 使用 `SQLSetStmtAttr` 设置 `SQL_INFX_ATTR_AUTO_FREE` 属性。

当使用 `SQLSetConnectAttr` 进行设置 `SQL_INFX_ATTR_DEFERRED_PREPARE` 属性时，可以在 C2 和 C5 之间（包括两者）的任何连接状态中进行设置

`SQL_INFX_ATTR_DEFERRED_PREPARE` 属性。而只有当语句处于 S1（分配）状态时，才能使用 `SQLSetStmtAttr` 设置此属性。可以通过使用 `SQLGetConnectAttr` 或 `SQLSetStmtAttr` 检索 `SQL_INFX_ATTR_DEFERRED_PREPARE` 属性的值。

## 8.8 设置简单大对象访存数组大小

为了减少涉及多行简单大对象数据的访存的网络开销，可以设置数组大小。

当驱动程序接收到多行访存请求时，请设置数组大小，它优化访存缓冲区大小和内在的访存数组大小，并消除每个简单大对象的数据库服务器的往返行程。

将数组大小设置为大于 1，也会使其它数据类型的数据的性能提升。因为如果需要，它会自动增加访存缓冲区的大小。（如果指定的行数可以放在当前访存缓冲区中，则设置它的效果不大）。

应用程序可以通过设置语句属性 `SQL_ATTR_ROW_ARRAY_SIZE` 或将 `ARD` 头字段 `SQL_DESC_ARRAY_SIZE` 设置为大于 1 的值，然后调用 `SQLFetch` 或 `SQLFetchScroll` 来请求返回多行。（`SQL_ATTR_ROW_ARRAY_SIZE` 的缺省值为 1。）该驱动程序在检索多行访存请求时会识别它并优化访存缓冲区大小和内部访存数组大小的设置。这些设置都基于内部元组大小，用户行数组大小的设置以及访存数组大小的当前设置。

不能在以下情景中使用内部访存数组功能：

- 当启用了 `OPTOFC` 和 `deferred-PREPARE` 时

要使用访存数组功能，驱动程序需要知道从数据库服务器接收数据之后，将访存请求发送到数据库服务器之前，行将有多大。当这两个功能都被启用时，在执行访存之前是获得不了这个信息的。

- 当使用滚动游标时

内部用于滚动游标的客户端到服务器的协议与用于访存数组的那些协议不同。数据库服务器在滚动游标中不支持简单大对象类，会返回一个错误。

- 当使用 `SQLGetData` 时

为了驱动程序使用访存数组功能，它必须能够告知数据库服务器准备在访存请求时准备接收多少数据。在 `SQLFetch` 之后调用 `SQLGetData`。

根据 ODBC 标准，当使用 `block` 游标时，应用程序必须在调用 `SQLGetData` 之前调用 `SQLSetPos` 定位特定行上的游标。`SQLSetPos` 只能用于滚动游标，并且不能在滚动游标中

使用简单大对象列。同样根据标准，SQLGetData 不能与行集大小小于 1 的只进游标一起使用。

使用 SQLGetData 的替代方法是使用 SQLBindCol，它在调用 SQLFetch 之前出现。

您可能想要优化 SQL\_ATTR\_ROW\_ARRAY\_SIZE 的使用，以便应用程序根据传输到单个缓冲区中的最大行数设置该值。在准备好语句之后，应用程序可能调用 SQLGetStmtAttr 获取 SQL\_INFX\_ATTR\_FET\_ARR\_SIZE 的值。如果数据符合一个访存缓冲区，则 SQL\_INFX\_ATTR\_FET\_ARR\_SIZE 的内部设置等于 SQL\_ATTR\_ROW\_ARRAY\_SIZE 的应用程序的设置。在实践中，这只对大型结果集有用。

## 8.9 SPL 输出参数功能

GBase 8s ODBC Driver 支持 ODBC 定义的从数据库过程获取返回值的方法。

具体来说，ODBC 支持在过程调用转义序列中的等号前面的参数。与该参数关联的主机变量在执行语句时使用 SQLExecute 或 SQLExecDirect 更新。

在过程调用转义序列的 GBase 8s ODBC Driver 定义中，只返回一个值；因此，此功能具有以下限制：

- 使用此功能的过程必须只能返回一个值，尽管它们可能返回多行。

如果不符合条件，则会忽略参数及其绑定。

- 来自第一行的数据只能放在与绑定参数关联的主变量中，尽管用于此功能的过程可以返回多行。

要从 GBase 8s 数据库服务器返回多值，多行结果集，您必须像访存 select 语句的结果列那样访存数据。此输出参数功能可以与绑定列或列的现有应用程序一起使用，并在通过过程调用访问数据时调用 SQLFetch或调用 SQLFetch 和 SQLGetData 。因此，当返回多行时不会生成错误或警告。

可以使用其中一种或两种方法从存储过程检索数据。主机变量可以绑定为参数或列，或两者。如果使用独立的缓冲区，则只有作为参数的主机变量在语句执行时被更新，并且只有作为列绑定的主机变量在访存时被更新，通过 SQLGetData 访问的未绑定的列不受影响。

## 8.10 OUT 和 INOUT 参数

GBase 8s Client Software Development KitVersion 4.10 支持执行 SPL 期间使用 OUT 和 INOUT 参数。

支持以下数据类型：

- BIGINT
- BLOB
- BOOLEAN
- DATETIME

- CHAR
- CLOB
- DECIMAL
- FLOAT
- INT8
- INTEGER
- INTERVAL
- LVARCHAR
- MONEY
- NCHAR
- NVARCHAR
- SMALLFLOAT
- SMALLINT
- VARCHAR

在执行 SPL 时使用 OUT 或 INOUT 参数具有以下限制：

- 不支持集合数据类型。例如 LIST 、MULTISET 、ROW 和 SET。
- 不支持返回的结果集。执行具有 OUT 或 INOUT 参数的 SPL 之后，不能调用 SQLFetch 或 SQLGetData。
- 只能返回一个值，即每执行一个 SPL 只返回一个 OUT 或 INOUT 参数的集合。

下列 SPL 执行示例创建一个 OUT，一个 INOUT 和一个 IN（缺省）参数以及一个返回值。

```
create procedure myproc(OUT intparam INT, INOUT charparam char(20),
    inparam int) returns int
<body of SPL>
end procedure;
```

下列代码示例 outinoutparamblob.c，显示如何使用具有 BLOB 、INTEGER 和 VARCHAR 数据类型的 OUT 和 INOUT 参数。

```
/* Drop procedure */
SQLExecDirect(hstmt, (UCHAR *)"drop procedure spl_out_param_blob;",
SQL_NTS);
SQLExecDirect(hstmt, (UCHAR *)"drop table tab_blob;", SQL_NTS);

/* Create table with BLOB column */
rc = SQLExecDirect(hstmt, (UCHAR *)"create table tab_blob(c_blob BLOB,
c_int INTEGER, c_char varchar(20));", SQL_NTS);
if (checkError(rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error in Step 2 --
SQLExecDirect failed\n"))
    goto Exit;

/* Insert one row into the table */
rc = SQLExecDirect(hstmt, (UCHAR *)"insert into tab_blob
```

```

values(filetoblob('insert.data', 'c'), 10, 'blob_test');", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error in Step 2
-- SQLExecDirect failed\n"))
        goto Exit;

    /* Create procedure */
    rc = SQLExecDirect(hstmt, "CREATE PROCEDURE
spl_out_param_blob(inParam int,
    OUT blobparam BLOB, OUT intparam int, OUT charparam varchar(20)) \n"
        "returning integer; \n"
        "select c_blob, c_int, c_char into blobparam,
        intparam, charparam from tab_blob; \n"
        "return inParam; \n"
        "end procedure; ",
        SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error in Step 2
-- SQLExecDirect failed\n"))
        goto Exit;

    /* Prepare stored procedure to be executed */
    rc = SQLPrepare(hstmt, (UCHAR *) "{? = call spl_out_param_blob
(?, ?, ?, ?)}", SQL_NTS);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLPrepare failed\n"))
        goto Exit;

    /* Bind the required parameters */
    rc = SQLBindParameter(hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG,
SQL_INTEGER, 3, 0, &sParm1, 0, &cbParm1);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 1 failed\n"))
        goto Exit;

    rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 10, 0, &sParm2, 0, &cbParm2);
    if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 2 failed\n"))
        goto Exit;

    rc = SQLBindParameter(hstmt, 3, SQL_PARAM_OUTPUT,
SQL_C_BINARY,
    SQL_LONGVARIABLE, sizeof(blob_buffer), 0, blob_buffer,
    sizeof(blob_buffer), &cbParm3);

```

```

if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 3 failed\n"))
    goto Exit;

rc = SQLBindParameter(hstmt, 4, SQL_PARAM_OUTPUT, SQL_C_LONG,
SQL_INTEGER, 10, 0, &sParm3, 0, &cbParm4);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 4 failed\n"))
    goto Exit;

rc = SQLBindParameter (hstmt, 5, SQL_PARAM_OUTPUT, SQL_C_CHAR,
SQL_VARCHAR, sizeof(schar), 0, schar, sizeof(schar), &cbParm6);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 5 failed\n"))
    goto Exit;

/* Exeute the prepared stored procedure */
rc = SQLExecute(hstmt);
if (checkError (rc, SQL_HANDLE_STMT, hstmt,
(SQLCHAR *)
"Error in Step 2 -- SQLExecute failed\n"))
    goto Exit;

len =
strlen("123456789abcdefghijklmnopqrstuvwxyz
1234567890123456789012345678901234567890 ");

if( (sParm2 != sParm1) || (10 != sParm3) ||
(strcmp("blob_test", schar)) || (cbParm3 != len) )
{
    fprintf(stdout, "\n 1st Data compare failed!");
    goto Exit;
}
else
{
    fprintf(stdout, "\n 1st Data compare successful");
}

/* Reset the parameters */
rc = SQLFreeStmt(hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 3 -- SQLFreeStmt failed\n"))
    goto Exit;

```



```
/* Reset variables */
sParm1 = 0;
        cbParm6 = cbParm1 = SQL_NTS;
cbParm3 = SQL_NULL_DATA;
schar[0]=0;
        blob_buffer[0]=0;
```

## 8.11 异步执行

设计应用程序利用支持异步执行的数据源。异步调用不会加快执行速度，但是设计良好的应用程序运行效率更高。

开启异步执行本身不会提高性能。但是，设计良好的应用程序可以利用异步查询执行，允许用户在数据库服务器上对查询进行评估时处理其它事情。也许用户启动一个或多个子查询或者选择在其它应用程序中工作，而所有这些查询都在数据库服务器上执行。设计用于异步执行的应用程序，允许用户同时处理多个任务，从而使应用程序看起来运行得更快。

缺省情况下，应用程序调用 ODBC 驱动程序，然后以同步方式对数据库服务器执行语句。在这种操作模式下，驱动程序不会返回控制应用程序直到它自己的到数据库服务器的请求完成。对于需要几秒钟才能完成的语句，此控制返回延迟可能会导致性能下降。

一些数据源支持异步执行。当处于异步模式，应用程序调用 ODBC 驱动程序，控制会立刻返回。在此模式中，驱动程序将状态 `SQL_STILL_EXECUTING` 返回到应用程序，然后将适当的请求发送到数据库服务器执行。应用程序以不同的时间间隔轮询驱动程序，驱动程序在该时间点轮询数据库服务器以查看查询是否已经执行完成。如果查询仍在执行，则状态 `SQL_STILL_EXECUTING` 返回到应用程序。如果已经完成，则返回类似 `SQL_SUCCESS` 的状态，然后应用程序可以开始访问记录。

## 8.12 使用定位更新和删除更新数据

虽然定位更新不适用于所有类型的应用程序，但尽可能使用定位更新和删除。

定位更新（使用 `UPDATE WHERE CURRENT OF CURSOR`）允许您通过将数据库游标定位到要更改的行来更新数据，并通知驱动程序更改数据。您不必强制构建复杂的 SQL 语句；提供要更改的语句。

除了要使代码更易于维护，定位更新通常会提高性能。由于数据库服务器已经在行上定位（对于当前正在处理 `SELECT` 语句），因此定位要更改的行的多余查询是不必要的。如果该行必须定位，则数据库服务器通常具有指向可用行的内部指针（例如 `ROWID`）。

要支持使用滚动游标定位的 `UPDATE` 和 `DELETE` 语句，GBase 8s ODBC Driver 从原始的定位语句构造一个新的搜索的 `UPDATE` 或 `DELETE` 语句。但是，数据库服务器不能直接更新滚动游标。相反，GBase 8s ODBC Driver 会构造一个 `WHERE` 子句，用于引用在

WHERE CURRENT OF CURSOR 子句中引用的 SELECT 语句中访存的每个列。SELECT 语句的行集数据缓存中的值绑定到已经建立的 WHERE 子句中的每个值。

该定位方法比使用具有 FORWARD ONLY 游标的子句 WHERE CURRENT OF CURSOR 子句更慢，更容易出错。如果访存的行不包含唯一键值，则构建的 WHERE 子句可能标识一行或多行，这会导致许多行被删除和更新。以这种方式删除行会影响定位的 UPDATE 和 DELETE 语句，和使用滚动游标的 SQLSetPos 语句。

使用 SQLSpecialColumns 确定在 WHERE 子句中用于更新数据的最佳列集。很多时候，伪列提供了对数据的最快访问；您只能通过使用 SQLSpecialColumns 确定这些列。

许多应用程序不能设计为利用定位的更新和删除。这些应用程序通常通过形成 WHERE 子句来更新数据。该 WHERE 子句由结果集中返回的一部分列值组成。某些应用程序可能会使用所有可搜索的结果列或通过调用 SQLStatistics 来查找可能属于唯一索引的列的 WHERE 子句。这些方法通常有效，但是会导致相当复杂的查询。

考虑以下示例：

```
rc = SQLExecDirect (hstmt, "SELECT first_name, last_name, ssn,
    address, city, state, zip FROM emp", SQL_NTS);
// fetchdata
:
rc = SQLExecDirect (hstmt, "UPDATE EMP SET ADDRESS = ?
    WHERE first_name = ? AND last_name = ? AND ssn = ? AND
    address = ? AND city = ? AND state = ? AND zip = ?", SQL_NTS);
// fairly complex query
```

应用程序应该调用 SQLSpecialColumns/SQL\_BEST\_ROWID 检索标识任何给定记录的最佳列集（可能是伪列）。许多数据库支持在表定义中没有显式用户定义的特定列，但是是每个表的隐藏列（例如，ROWID、TID 和其它列）。这些伪列总是提供对数据的最快访问。因为它们通常指向记录的确切位置。因为伪列不是显式表定义的一部分，所以它们不会从 SQLSpecialColumns 返回。确定伪列是否存在的唯一方法是调用 SQLSpecialColumns。

考虑之前的示例，这次使用 SQLSpecialColumns：

```
:
rc = SQLSpecialColumns (hstmt, ..... 'emp', ...);
:
rc = SQLExecDirect (hstmt, "SELECT first_name, last_name, ssn,
    address, city, state, zip, ROWID FROM emp", SQL_NTS);
// fetch data and probably "hide" ROWID from the user
:
rc = SQLExecDirect (hstmt, "UPDATE emp SET address = ? WHERE
    ROWID = ?", SQL_NTS);
// fastest access to the data!
```



如果您的数据源不包含特定的伪列，则 `SQLSpecialColumns` 的结果集由指定表上的最佳唯一索引组成（如果唯一索引存在）。因此，您的应用程序不会另外调用 `SQLStatistics` 来查找最小的唯一索引。

## 8.13 BIGINT 和 BIGSERIAL 数据类型

`BIGINT` 和 `BIGSERIAL` 数据类型具有与 `INT8` 和 `SERIAL8` 数据类型相同的范围。

但是，`BIGINT` 和 `BIGSERIAL` 在 `INT8` 和 `SERIAL8` 上的存储和计算有优势。

## 8.14 消息传输优化

如果激活消息传输优化功能（`OPTMSG`），则驱动程序可以最大限度地减少大多数 GBase 8s ODBC 函数的数据库服务器的消息传输。

另外，该驱动程序将来自数据库服务器的消息链接在一起，消除一些小消息包以实现优化消息传输。

要激活消息传输优化，将 `SQL_INFX_ATTR_OPTMSG` 语句属性设置为 1。优化的缺省值为：OFF。

### 8.14.1 消息链接限制

即使启用消息传输优化，GBase 8s ODBC 也不会链接 SQL 函数。

ODBC 不链接的 SQL 函数有：

- `SQLDisconnect`
- `SQLConnect`
- `SQLEndTran`
- `SQLExecute`（如果驱动程序使用 `select` 或调用过程返回结果，并且驱动程序使用 `insert` 游标指向批量插入）
- `SQLExtendedFetch`
- `SQLFetch`
- `SQLFetchScroll`
- `SQLPrepare`

当驱动程序接触到上表所列的函数时，它执行以下操作：

1. 只有当遇到需要数据库服务器响应的 SQL 语句时，会将消息队列刷新到数据库服务器。  
当驱动程序运行不需要网络流量的函数时，它不会刷新消息队列，例如 `SQLAllocStmt`。
2. 继续后续 SQL 语句的消息链接。

### 8.14.2 禁用消息链接

可以选择禁用消息链接。

在禁用消息链接之前，请考虑以下情况：

- 一些 SQL 语句需要立即回复。如果禁用消息链接，请在限制的 SQL 语句完成后重新启动 OPTMSG 功能。
- 如果执行调试。可以在试图确定每个 SQL 语句应答时禁用 OPTMSG 功能。
- 如果启用 OPTMSG，该消息会在数据库服务器中列队，但是不会发送进行处理。考虑在程序中的最后一条 SQL 语句之前禁用消息链接，以确保数据库服务器在应用程序退出之前处理所有消息。
- 如果禁用消息链接，必须在需要它的 SQL 语句之后立即重置 SQL\_INFX\_ATTR\_OPTMSG 属性，来避免意外链接。

以下示例显示如何禁用消息链接，通过在 DELETE 语句之后放置 SQL\_INFX\_ATTR\_OPTMSG 属性。如果将该属性放在删除语句后，则驱动程序可以在下一条 SQL 语句之前时，刷新所有排队的消息：

```
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 1);
SQLExecDirect(hstmt, (unsigned char *)
"delete from customer", SQL_NTS);
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 0);
SQLExecDirect(hstmt, (unsigned char *)
"create index ix1 on customer (zipcode)", SQL_NTS);
```

意外消息链接会使其很难确定哪一个链接的语句失败。

在 CREATE INDEX 语句中，驱动程序将 DELETE 和 CREATE INDEX 语句发送到数据库服务器。

### 8.14.3 优化消息传输的错误

当启用 OPTMSG 功能时，GBase 8s ODBC 不会对任何连接的语句执行错误处理。

如果您不确定某个特定语句是否会产生错误，则在代码中包含错误处理语句，并不要为此语句启用消息链接。

当链接语句中发生错误后，数据库服务器停止后续语句的执行。例如，在以下代码片段中，它试图链接五条 INSERT 语句：

```
SQLExecDirect(hstmt, "create table tab1 (col1 INTEGER)", SQL_NTS);
/* enable message chaining */
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 1);
/* these two INSERT statements execute successfully */
SQLExecDirect(hstmt, "insert into tab1 values (1)", SQL_NTS);
SQLExecDirect(hstmt, "insert into tab1 values (2)", SQL_NTS);
/* this INSERT statement generates an error because the data
* in the VALUES clause is not compatible with the column type */
SQLExecDirect(hstmt, "insert into tab1 values ('a')", SQL_NTS);
/* these two INSERT statements never execute */
```

```
SQLExecDirect(hstmt, "insert into tab1 values (3)", SQL_NTS);
SQLExecDirect(hstmt, "insert into tab1 values (4)", SQL_NTS);
/* disable message chaining */
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 0);
/* commit work */
rc = SQLEndTran (SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
if (rc != SQL_SUCCESS)
```

在此示例中，会发生以下操作：

- 驱动程序将这五个 INSERT 语句和 COMMIT WORK 语句发送到数据库服务器执行。
- 数据库将 1 和 2 的 col1 值插入到 tab1 表中。
- 第三条 INSERT 语句产生错误，因此数据服务器不会执行后续的 INSERT 语句或 COMMIT WORK 语句。
- 当队列到达 SQLEndTran 函数，驱动程序刷新消息队列。
- SQLEndTran 函数，它是链接语句中的最后一条语句，返回失败的 INSERT 语句的错误。

如果需要保存数据库服务器插入到 col1 的值，必须自己提交它们。

## 9 错误消息

本主题描述 GBase 8s ODBC Driver 错误消息。

本主题提供以下信息：

- 诊断 SQLSTATE 值
- 映射到 GBase 8s 错误消息的 SQLSTATE 值

- GBase 8s ODBC Driver 错误消息映射到特定的 SQLSTATE 值
- 有关错误消息的描述，请使用 finderr 实用程序。

## 9.1 诊断 SQLSTATE 值

每个 GBase 8s ODBC Driver 函数可以返回对应于 GBase 8s 错误码的 SQLSTATE 值。函数可以返回从实现特定的情况中产生的额外的 SQLSTATE 值。SQLError 返回由 GLS 和 SQL Access Group SQL CAE 规范（1992）定义的 SQLSTATE 值。

SQLSTATE 值是由两个字符类值和三个字符的子类值组成的字符串。类值 01 表示警告，并伴随一个 SQL\_SUCCESS\_WITH\_INFO 返回码。01 以外的类值（类 IM 除外）表示错误，并伴随一个 SQL\_ERROR 返回码。IM 类表示由 GBase 8s ODBC Driver 实现派生的警告和错误。任何类中的子类值 000 都是给定类中实现定义的条件。ANSI SQL-92 定义了类和子类值的分配。

## 9.2 将 SQLSTATE 值映射到 GBase 8s 错误消息

查看 GBase 8s ODBC Driver 可以返回的 SQLSTATE 值。

下表映射 GBase 8s ODBC Driver 可以返回的 SQLSTATE 值。

SQL\_SUCCESS 的返回值通常表示函数执行成功，尽管 SQLSTATE 00000 也表示成功。

SQLSTATE	错误消息	从这里返回	
01 000	General warning	所有的 GBase 8s ODBC Driver 函数，除了： SQLAllocEnv                  SQLError	
01002	Disconnect error	SQLDisconnect	
01004	Data truncated	SQLBrowseConnect	SQLColAttributes
		SQLDataSources	SQLDescribeCol
		SQLDriverConnect	SQLDrivers
		SQLExecDirect	SQLExecute
		SQLExtendedFetch	SQLFetch
		SQLGetCursorName	SQLGetData
		SQLGetInfo	SQLNativeSql
		SQLPutData	SQLSetPos
01006	Privilege not revoked	SQLExecDirect	SQLExecute
01S00	Invalid connection string attribute	SQLBrowseConnect	SQLDriverConnect
01S01	Error in row	SQLExtendedFetch	SQLSetPos
01S02	Option value changed	SQLSetConnectOption	SQLSetStmtOption

SQLSTATE	错误消息	从这里返回	
01S03	No rows updated or deleted	SQLExecDirect SQLSetPos	SQLExecute
01S04	More than one row updated or deleted	SQLExecDirect SQLSetPos	SQLExecute
07001	Wrong number of parameters	SQLExecDirect	SQLExecute
07006	Restricted data type attribute violation	SQLBindParameter SQLFetchSQLGetData	SQLExtendedFetch
08001	Unable to connect to data source	SQLBrowseConnect SQLDriverConnect	SQLConnect
08002	Connection in use	SQLBrowseConnect SQLDriverConnect	SQLConnect SQLSetConnectOption
08003	Connection not open	SQLAllocStmt SQLGetConnectOption SQLNativeSql SQLTransact	SQLDisconnect SQLGetInfo SQLSetConnectOption
08004	Data source rejected establishment of connection	SQLBrowseConnect SQLDriverConnect	SQLConnect
08007	Connection failure during transaction	SQLTransact	
08S01	Communication link failure	SQLBrowseConnect SQLColumns SQLDriverConnect SQLExecute SQLFetch SQLFreeConnect SQLGetTypeInfo SQLPrepare SQLProcedureColumns SQLPutDataSQL SQLSetStmtOption SQLStatistics SQLTables	SQLColumnPrivileges SQLConnect SQLExecDirect SQLExtendedFetch SQLForeignKeys SQLGetData SQLParamData SQLPrimaryKeys SQLProcedures SetConnectOption SQLSpecialColumns SQLTablePrivileges
21S01	Insert value list does not match column list	SQLExecDirect	SQLPrepare
21S02	Degree of derived table does not match column list	SQLExecDirect	SQLPrepare SQLSetPos
22001	String data right truncation	SQLPutData	

SQLSTATE	错误消息	从这里返回	
22003	Numeric value out of range	SQLExecDirect SQLExtendedFetch SQLGetData SQLPutData	SQLExecute SQLFetch SQLGetInfo SQLSetPos
22005	Error in assignment	SQLExecDirect SQLExtendedFetch SQLGetData SQLPutData	SQLExecute SQLFetch SQLPrepare SQLSetPos
22008	Datetime field overflow	SQLExecDirect SQLExtendedFetch SQLFetchSQLGetData SQLPutDataSQLSetPos	SQLExecute
22012	Division by zero	SQLExecDirect SQLExtendedFetch	SQLExecute SQLFetchSQLGetData
22026	String data, length mismatch	SQLParamData	
23000	Integrity constraint violation	SQLExecDirect SQLSetPos	SQLExecute
24000	Invalid cursor state	SQLColAttributes SQLColumns SQLExecDirect SQLExtendedFetch SQLForeignKeys SQLGetStmtOption SQLPrepare SQLProcedureColumns SQLSetCursorName SQLSetStmtOption SQLStatistics SQLTables	SQLColumnPrivileges SQLDescribeCol SQLExecute SQLFetch SQLGetData SQLGetTypeInfo SQLPrimaryKeys SQLProcedures SQLSetPos SQLSpecialColumns SQLTablePrivileges
25000	Invalid transaction state	SQLDisconnect	
28000	Invalid authorization specification	SQLBrowseConnect QLDriverConnect	SQLConnectS
34000	Invalid cursor name	SQLExecDirect SQLSetCursorName	SQLPrepare
37000	Syntax error or access violation	SQLExecDirect SQLPrepare	SQLNativeSql
3C000	Duplicate cursor name	SQLSetCursorName	
40001	Serialization failure	SQLExecDirect SQLExtendedFetch	SQLExecute SQLFetch

SQLSTATE	错误消息	从这里返回	
42000	Syntax error or access violation	SQLExecDirect SQLPrepare	SQLExecute SQLSetPos
70100	Operation aborted	SQLCancel	
IM001	Driver does not support this function	所有的 ODBC 函数，除了： SQLAllocConnect      SQLAllocEnv SQLDataSources      SQLDrivers SQLError              SQLFreeConnect SQLFreeEnv            SQLGetFunctions	
IM002	Data source name not found and no default driver specified	SQLBrowseConnect SQLConnect SQLDriverConnect	
IM003	Specified driver could not be loaded	SQLBrowseConnect SQLDriverConnect	SQLConnect
IM004	Driver's SQLAllocEnv failed	SQLBrowseConnect SQLDriverConnect	SQLConnect
IM005	Driver's SQLAllocConnect failed	SQLBrowseConnect SQLDriverConnect	SQLConnect
IM006	Driver's SQLSetConnectOption failed	SQLBrowseConnect SQLDriverConnect	SQLConnect
IM007	No data source or driver specified; dialog prohibited	SQLDriverConnect	
IM008	Dialog failed	SQLDriverConnect	
IM009	Unable to load translation shared library (DLL)	SQLBrowseConnect SQLDriverConnect	SQLConnect SQLSetConnectOption
IM010	Data source name too long	SQLBrowseConnect	SQLDriverConnect
IM011	Driver name too long	SQLBrowseConnect	SQLDriverConnect
IM012	DRIVER keyword syntax error	SQLBrowseConnect	SQLDriverConnect
IM013	Trace file error	All ODBC functions.	
S0001	Base table or view already exists	SQLExecDirectSQLPrepare	
S0002	Base table not found	SQLExecDirectSQLPrepare	
S0011	Index already exists	SQLExecDirectSQLPrepare	
S0012	Index not found	SQLExecDirectSQLPrepare	
S0021	Column already exists	SQLExecDirectSQLPrepare	
S0022	Column not found	SQLExecDirectSQLPrepare	

SQLSTATE	错误消息	从这里返回	
S0023	No default for column	SQLSetPos	
S1000	General error	所有的 ODBC 函数	
S1001	Memory allocation failure	所有的 ODBC 函数，除了： SQLAllocEnv            SQLError SQLFreeConnect        SQLFreeEnv	
S1002	Invalid column number	SQLBindCol	SQLColAttributes
		SQLDescribeCol	SQLExtendedFetchS
		QLFetch	SQLGetData
S1003	Program type out of range	SQLBindCol	SQLBindParameter
		SQLGetData	
S1004	SQL data type out of range	SQLBindParameter	SQLGetTypeInfo
S1008	Operation canceled	所有可以执行异步的 ODBC 函数： SQLColAttributes        SQLColumnPrivileges SQLColumns              SQLDescribeCol SQLDescribeParam        SQLExecDirect SQLExecute                SQLExtendedFetch SQLFetch                  SQLForeignKeys SQLGetData                SQLGetTypeInfo SQLMoreResults            SQLNumParams SQLNumResultCols        SQLParamData SQLPrepare                SQLPrimaryKeys SQLProcedureColumns     SQLProcedures SQLPutData                SQLSetPos SQLSpecialColumns        SQLStatistics SQLTablePrivileges        SQLTables	
S1009	Invalid argument value	SQLAllocConnect	SQLAllocStmt
		SQLBindCol	SQLBindParameter
		SQLExecDirect	SQLForeignKeys
		SQLGetData	SQLGetInfo
		SQLNativeSql	SQLPrepare
		SQLPutData	SQLSetConnectOption
		SQLSetCursorName	SQLSetPos
		SQLSetStmtOption	
S1010	Function sequence error	SQLBindCol	SQLBindParameter
		SQLColAttributes	SQLColumnPrivileges
		SQLColumns	SQLDescribeCol
		SQLDisconnect	SQLExecDirect
		SQLExecute	SQLExtendedFetch
		SQLFetch	SQLForeignKeys
		SQLFreeConnect	SQLFreeEnv



SQLSTATE	错误消息	从这里返回	
		SQLFreeStmt	SQLGetConnectOption
		SQLGetCursorName	SQLGetData
		SQLGetFunctions	SQLGetStmtOption
		SQLGetTypeInfo	SQLMoreResults
		SQLNumParams	SQLNumResultColsS
		QLParamData	SQLParamOptions
		SQLPrepare	SQLPrimaryKeys
		SQLProcedureColumns	SQLProcedures
		SQLPutData	SQLRowCount
		SQLSetConnectOption	SQLSetCursorName
		SQLSetPos	SQLSetScrollOptions
		SQLSetStmtOption	SQLSpecialColumns
		SQLStatistics	SQLTablePrivileges
		SQLTables	SQLTransact
S1011	Operation invalid at this time	SQLGetStmtOption	SQLSetConnectOption
		SQLSetStmtOption	
S1012	Invalid transaction operation code specified	SQLTransact	
S1015	No cursor name available	SQLGetCursorName	
S1090	Invalid string or buffer length	SQLBindCol	SQLBindParameter
		SQLBrowseConnect	SQLColAttributes
		SQLColumnPrivileges	SQLColumns
		SQLConnect	SQLDataSources
		SQLDescribeCol	SQLDriverConnect
		SQLDrivers	SQLExecDirect
		SQLExecute	SQLForeignKeys
		SQLGetCursorName	SQLGetData
		SQLGetInfo	SQLNativeSql
		SQLPrepare	SQLPrimaryKeys
		SQLProcedureColumns	SQLProcedures
		SQLPutData	SQLSetCursorName
		SQLSetPos	SQLSpecialColumns
		SQLStatistics	SQLTablePrivileges
		SQLTables	
S1091	Descriptor type out of range	SQLColAttributes	
S1092	Option type out of range	SQLFreeStmt	SQLGetConnectOption
		SQLGetStmtOption	SQLSetConnectOption
		SQLSetStmtOption	

SQLSTATE	错误消息	从这里返回
S1093	Invalid parameter number	SQLBindParameter
S1094	Invalid scale value	SQLBindParameter
S1095	Function type out of range	SQLGetFunctions
S1096	Information type out of range	SQLGetInfo
S1097	Column type out of range	SQLSpecialColumns
S1098	Scope type out of range	SQLSpecialColumns
S1099	Nullable type out of range	SQLSpecialColumns
S1100	Uniqueness option type out of range	SQLStatistics
S1101	Accuracy option type out of range	SQLStatistics
S1103	Direction option out of range	SQLDataSources SQLDrivers
S1104	Invalid precision value	SQLBindParameter
S1105	Invalid parameter type	SQLBindParameter
S1106	Fetch type out of range	SQLExtendedFetch
S1107	Row value out of range	SQLExtendedFetch SQLParamOptions SQLSetScrollOptions SQLSetPos
S1108	Concurrency option out of range	SQLSetScrollOptions
S1109	Invalid cursor position	SQLExecute SQLGetData SQLSetPos SQLExecDirect SQLGetStmtOption
S1110	Invalid driver completion	SQLDriverConnect
S1111	Invalid bookmark value	SQLExtendedFetch
S1C00	Driver not capable	SQLBindCol SQLColAttributes SQLColumns SQLExecute SQLFetch SQLBindParameter SQLColumnPrivileges SQLExecDirect SQLExtendedFetch SQLForeignKeys

SQLSTATE	错误消息	从这里返回	
		SQLGetData	SQLTablePrivileges
		SQLGetInfo	SQLGetStmtOption
		SQLGetTypeInfo	SQLPrepare
		SQLPrimaryKeys	SQLTransact
		SQLProcedures	SQLSetConnectOption
		SQLSetPosSQL	SQLTables
		SQLSetStmtOption	SQLSpecialColumns
		SQLStatistics	SQLTransact
		SetScrollOptions	
		SQLProcedureColumns	
		SQLGetConnectOption	
S1T00	Time-out expired	SQLBrowseConnect	SQLColAttributes
		SQLColumnPrivileges	SQLColumns
		SQLConnect	SQLDescribeCol
		SQLDriverConnect	SQLExecDirect
		SQLExecute	SQLExtendedFetch
		SQLFetch	SQLForeignKeys
		SQLGetData	SQLGetInfo
		SQLGetTypeInfo	SQLMoreResults
		SQLNumParams	SQLNumResultCols
		SQLParamData	SQLPrepare
		SQLPrimaryKeys	SQLProcedureColumns
		SQLProcedures	SQLPutData
		SQLSetPos	SQLSpecialColumns
		SQLStatistics	SQLTablePrivileges
		SQLTables	

### 9.3 将 GBase 8s 错误消息映射到 SQLSTATE 值

本节的余下部分描述了 GBase 8s ODBC Driver 函数的诊断 SQLSTATE 值。

每个 SQLSTATE 值的返回码是 SQL\_ERROR，否则是表示其它信息的描述。当函数返回 SQL\_SUCCESS\_WITH\_INFO 或 SQL\_ERROR，可以调用 SQLError 获取 SQLSTATE 值。

#### 9.3.1 已弃用的和新的 GBase 8s ODBC Driver API

在 Version 4.10 中，许多 ODBC API 已被弃用，它们的函数都转移到新的 API 中。

只更改了名称；没有更改功能。下表列出了弃用的和新的 API。

表 1. 弃用的和新的 ODBC API

弃用的 ODBC API	新的 ODBC API
SQLAllocConnect	SQLAllocHandle
SQLAllocEnv	SQLAllocHandle
SQLAllocStmt	SQLAllocHandle
SQLColAttributes	SQLColAttribute
SQLError	SQLGetDiagRec
SQLExtendedFetch	SQLFetch, SQLFetchScroll
SQLFreeConnect	SQLFreeHandle
SQLFreeEnv	SQLFreeHandle
SQLFreeStmt	SQLFreeHandle
SQLGetConnectOption	SQLGetConnectAttr
SQLGetStmtOption	SQLGetStmtAttr
SQLSetConnectOption	SQLSetConnectAttr
SQLSetPos	SQLBulkOperations
SQLSetStmtOption	SQLSetStmtAttr
SQLTransact	SQLEndTran

9.3.2 SQLAllocConnect（仅限核心级别）

本表描述了 SQLAllocConnect 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value

9.3.3 SQLAllocEnv（仅限核心级别）

SQLAllocEnv 为环境句柄，并初始化驱动程序调用级别接口，以供应用程序使用。

应用程序必须在调用其它驱动程序函数之前调用 SQLAllocEnv。

驱动程序在调用 SQLAllocEnv 之后不能直接返回 SQLSTATE 值，因为没有有效的句柄在调用 SQLError。

存在两个级别的 `SQLAllocEnv` 函数，一个在驱动程序管理器中（如果使用的话），一个在驱动程序中。在应用程序调用 `SQLConnect`、`SQLBrowseConnect` 或 `SQLDriverConnect` 之前，驱动程序管理器不会调用驱动程序级别的函数。如果在驱动程序级别 `SQLAllocEnv` 函数中发生了错误，则驱动程序管理器层 `SQLConnect`、`SQLBrowseConnect` 或 `SQLDriverConnect` 函数返回 `SQL_ERROR`。随后对使用 `henv`、`SQL_NULL_HDBC` 和 `SQL_NULL_HSTMT` 的 `SQLError` 调用返回 `SQLSTATE IM004`（驱动程序 `SQLAllocEnv` 函数失败），在以下驱动程序的错误之后：

- `SQLSTATE S1000`（一般错误）
- GBase 8s ODBC Driver `SQLSTATE` 值，它的范围是从 `S1000` 到 `S19ZZ`。

例如，`SQLSTATE S1001`（内存分配失败）表示从驱动程序管理器到驱动程序级别 `SQLAllocEnv` 的调用返回 `SQL_ERROR`，并且驱动程序管理器中的 `henv` 被设置为 `SQL_NULL_HENV`。

9. 3. 4 **SQLAllocStmt**（仅限核心级别）

`SQLAllocStmt` 为使用 `hdbc` 指定的连接的语句句柄分配内存。

应用程序必须在提交 `SQL` 语句之前调用 `SQLAllocStmt`。

下表描述了 `SQLAllocStmt` 的 `SQLSTATE` 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08003	-11017	Connection not open
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
08S01	-11301	A protocol error has been detected. Current connection is closed.

9. 3. 5 **SQLBindCol**（仅限核心级别）

`SQLBindCol` 为结果集中的列分配存储和 GBase 8s ODBC Driver C 数据类型。

`SQLBindCol` 按如下方式分配存储：

- 一个存储缓冲区，用于接收一列数据的内容
- 存储缓冲区的长度
- 一个存储位置，用于接收访存操作返回的数据列的实际长度
- 将 GBase 8s `SQL` 数据类型转换为 GBase 8s ODBC driver C 数据类型

下表描述了 `SQLBindCol` 的 `SQLSTATE` 和错误值。

SQLSTATE	错误值	错误消息
----------	-----	------

SQLSTATE	错误值	错误消息
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1003	-11063	Program type out of range
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable

**重要：** 应用程序可以调用 `SQLBindCol` 将列绑定到新的存储位置，而不管数据是否已经被存取。新的绑定替换旧的绑定列以及其它绑定列。新的绑定不会应用到已经访存的数据；它会在下次调用 `SQLFetch`、`SQLExtendedFetch` 或 `SQLSetPos` 时发生作用。

### 9.3.6 SQLBindParameter（仅限一级）

`SQLBindParameter` 将缓冲区绑定到 SQL 语句中的参数标记。

下表描述了 `SQLBindParameter` 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
07006	-11013	Restricted data type attribute violation
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1003	-11063	Program type out of range
S1004	-11064	SQL data type out of range
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1093	-11074	Invalid parameter number
S1094	-11075	Invalid scale value
S1104	-11084	Invalid precision value
S1105	-11085	Invalid parameter type
S1C00	-11092	Driver not capable

### 9.3.7 SQLBrowseConnect（仅限二级）

SQLBrowseConnect 支持发现和枚举连接到数据源所需的属性和属性值的迭代方法。

每次调用 SQLBrowseConnect 都会返回连续的属性和属性值。枚举所有的级别，完成到数据源的连接，并返回新连接到数据源的返回码的 SQLBrowseConnect 字符串。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
01S00	-11005	Invalid connection string attribute
08001	-11015	Unable to connect to data source
08002	-11016	Connection in use
08S01	-11020	Communication-link failure
28000	-11033	Invalid authorization specification
IM002	-11041	Data source not found and no default driver specified
IM003	-11042	Specified driver could not be loaded
IM004	-11043	Driver's SQLAllocEnv failed
IM005	-11044	Driver's SQLAllocConnect failed
IM006	-11045	Driver's SQLSetConnectOption failed
IM009	-11048	Unable to load translation shared library (DLL)
IM010	-11049	Data-source name too long
IM011	-11050	Driver name too long
IM012	-11051	DRIVER keyword syntax error
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11303	Input connection string too large
S1000	-11317	Invalid connectdatabase value specified
S1000	-11318	Invalid vmbcharlenexact value specified
S1000	-11320	Syntax error

SQLSTATE	错误值	错误消息
S1000	-11323	The statement contained an escape clause not supported by this database driver

9.3.8 SQLCancel（仅限核心级别）

SQLCancel 取消 hstmt 或查询的处理。

下表描述了此该函数的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01S05	-11010	Cancel treated as FreeStmt/Close.
70100	-11039	Operation aborted
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
08S01	-11301	A protocol error has been detected. Current connection is closed.

9.3.9 SQLColAttributes（仅限核心级别）

SQLColAttributes 返回结果集章列的描述符消息。

它不能用于返回标记列的信息（列 0）。描述符消息作为字符串，32 位描述符相关值或整数返回。

下表描述了 SQLColAttributes 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1091	-11072	Descriptor type out of range



SQLSTATE	错误值	错误消息
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired

SQLColAttributes 可以返回 SQLPrepare 或 SQLExecute 在 SQLPrepare 之后，SQLExecute 之前调用的任何 SQLSTATE。具体取决于数据源何时评估与 hstmt 关联的 SQL 语句。

9. 3. 10 SQLColumnPrivileges （仅限二级）

SQLColumnPrivileges 返回指定表的列和相关权限的列表。驱动程序将返回信息作为在指定的 hstmt 上设置的结果。

下表描述了 SQLColumnPrivileges 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

9. 3. 11 SQLColumns （仅限一级）

SQLColumns 返回指定表的列名。驱动程序将此信息作为指定 hstmt 上的结果集返回。

下表描述了SQLColumns 的 SQLSTATE 和错误消息。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

9. 3. 12 SQLConnect（仅限核心级别）

SQLConnect 加载驱动程序并建立到数据源的连接。

连接句柄引用存储有关连接的所有信息，包括状态、事务状态和错误消息。

下表描述了 SQLConnect 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08001	-11015	Unable to connect to data source
08002	-11016	Connection in use
08S01	-11020	Communication-link failure
28000	-11033	Invalid authorization specification
IM002	-11041	Data source not found and no default driver specified

SQLSTATE	错误值	错误消息
IM003	-11042	Specified driver could not be loaded
IM004	-11043	Driver's SQLAllocEnv failed
IM005	-11044	Driver's SQLAllocConnect failed
IM006	-11045	Driver's SQLSetConnectOption failed
IM009	-11048	Unable to load translation shared library (DLL)
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired
S1000	-11302	Insufficient connection information was supplied
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

9. 3. 13 SQLDataSources （仅限二级）

SQLDataSources 列出数据源名称。

下表描述了 SQLDataSources 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1103	-11083	Direction option out of range

9. 3. 14 SQLDescribeCol （仅限核心级别）

SQLDescribeCol 返回结果集中一列的详细信息（列名、类型、精度、小数位数和它是否具有 NULL 值）。

它不能用于返回有关标记列（列 0）的信息。

下表描述了 SQLDescribeCol 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
----------	-----	------

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired

当在 SQLPrepare 之后 SQLExecute 之前调用 SQLDescribeCol 时，SQLDescribeCol 可以返回 SQLPrepare 或 SQLExecute 返回的任何 SQLSTATE。具体取决于数据源何时评估与 hstmt 关联的 SQL 语句。

9. 3. 15 SQLDisconnect

SQLDisconnect 关闭与指定连接句柄关联的连接。

下表描述了 SQLDisconnect 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01002	-11002	Disconnect error
08003	-11017	Connection not open
25000	-11032	Invalid transaction state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
08S01	-11301	A protocol error has been detected. Current connection is closed.

用法

如果应用程序在 SQLBrowseConnect 返回 SQL\_NEED\_DATA 之后调用 SQLDisconnect，则在返回不同的返回码之前，驱动程序取消连接浏览过程并返回 hdbc 到未连接状态。

如果应用程序在未完成的事务与连接句柄关联时调用 `SQLDisconnect`，则驱动程序返回 `SQLSTATE 25000`（无效的事务状态），指示该事务没有变更，连接仍就打开。未完成的事务是没有提交或回滚的 `SQLTransact`。

如果应用程序在是否与连接相关联 `hstmt` 之前调用 `SQLDisconnect`，则驱动程序在它从数据源断开连接后释放剩余的 `hstmt`。然而，如果一个或多个与连接关联的 `hstmts` 仍在异步执行，则 `SQLDisconnect` 返回 `SQL_ERROR`，其中 `SQLSTATE` 的值为 `S1010`（函数序列错误）。

### 9.3.16 SQLDriverConnect（仅限一级）

`SQLDriverConnect` 等同于 `SQLConnect`。

它支持需要更多连接信息的数据源，而不是只需要 `SQLConnect` 对话框中的三个参数的数据源，这三个参数提示用户所有连接信息和未定义数据源名称的数据源。

`SQLDriverConnect` 提供以下连接选项：

- 可以使用包含数据源，一个或多个用户 ID，一个或多个密码和数据源需要的其他信息的连接字符串建立连接。
- 可以使用部分连接字符串建立连接，或者不使用其他信息。在这种情况下，GBase 8s ODBC Driver 可以提示用户提供连接信息。

建立连接后，`SQLDriverConnect` 连接字符串完成。应用程序可以使用此字符串进行后续的连接请求。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
01S00	-11005	Invalid connection string attribute
08001	-11015	Unable to connect to data source
08002	-11016	Connection in use
08S01	-11020	Communication-link failure
28000	-11033	Invalid authorization specification
IM002	-11041	Data source not found and no default driver specified
IM003	-11042	Specified driver could not be loaded
IM004	-11043	Driver's SQLAllocEnv failed
IM005	-11044	Driver's SQLAllocConnect failed
IM006	-11045	Driver's SQLSetConnectOption failed
IM007	-11046	No data source or driver specified; dialog prohibited
IM008	-11047	Dialog failed
IM009	-11048	Unable to load translation shared library

SQLSTATE	错误值	错误消息
IM010	-11049	Data-source name too long
IM011	-11050	Driver name too long
IM012	-11051	DRIVER keyword syntax error
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1110	-11090	Invalid driver completion
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11302	Insufficient connection information was supplied
S1000	-11303	Input connection string too large
S1000	-11317	Invalid connectdatabase value specified
S1000	-11318	Invalid vmbcharlenexact value specified
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

9. 3. 17 SQLDrivers （仅限一级）

SQLDrivers 列出驱动程序描述和驱动程序属性关键字。

下表描述了SQLDrivers 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1103	-11083	Direction option out of range

9. 3. 18 SQLError （仅限核心级别）

SQLError 返回错误或状态信息。

SQLError 不会传送它本身的错误值。当 SQLError 检索到任何错误消息，会返回 SQL\_NO\_DATA\_FOUND（sqlstate 等于 00000）。如果 SQLError 由于任何可以返回 SQL\_ERROR 的原因而无法访问错误，则 SQLError 返回 SQL\_ERROR，但是不会发布任何错误值。如果错误消息的缓冲区太小，则 SQLError 返回 SQL\_SUCCESS\_WITH\_INFO，但是仍不会返回 SQLError 的 SQLSTATE 值。

为了确定在错误消息中是否发生截断，应用程序可以将 cbErrorMsgMax 与写入 pcbErrorMsg 的消息文本的实际长度进行比较。

### 9.3.19 SQLExecDirect（仅限核心级别）

如果语句中存在任何参数，则 SQLExecDirect 通过使用参数标记变量的当前值来执行预备的语句。

SQLExecDirect 是提交一次性执行的 SQL 语句的最快方法。

下表描述了 SQLExecDirect 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
01006	-11004	Privilege not revoked
01S03	-11008	No rows updated or deleted
01S04	-11009	More than one row updated or deleted
07001	-11012	Wrong number of parameters
07S01	-11014	Invalid use of default parameter
08S01	-11020	Communication-link failure
21S01	-11021	Insert value list does not match column list
21S02	-11022	Degree of derived table does not match column list
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
23000	-11030	Integrity-constraint violation
24000	-11031	Invalid cursor state
34000	-11034	Invalid cursor name
37000	-11035	Syntax error or access violation
40001	-11037	Serialization failure
42000	-11038	Syntax error or access violation

SQLSTATE	错误值	错误消息
S0001	-11053	Base table or view already exists
S0002	-11054	Base table not found
S0011	-11055	Index already exists
S0012	-11056	Index not found
S0021	-11057	Column already exists
S0022	-11058	Column not found
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

### 9. 3. 20 SQLExecute（仅限核心级别）

如果语句中有任何参数标记，则 SQLExecute 通过使用当前参数标记变量的值来执行准备好的语句。

下表描述了 SQLExecute 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
01006	-11004	Privilege not revoked
01S03	-11008	No rows updated or deleted



SQLSTATE	错误值	错误消息
01S04	-11009	More than one row updated or deleted
07001	-11012	Wrong number of parameters
07S01	-11014	Invalid use of default parameter.
08S01	-11020	Communication-link failure
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
23000	-11030	Integrity constraint violation
24000	-11031	Invalid cursor state
40001	-11037	Serialization failure
42000	-11038	Syntax error or access violation
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

SQLExecute 可以返回 SQLPrepare 基于数据源评估与 hstmt 关联的 SQL 语句时返回的 SQLSTATE。

### 9. 3. 21 SQLExtendedFetch （仅限二级）

SQLExtendedFetch 扩展 SQLFetch 的功能。

SQLExtendedFetch 通过以下方式扩展功能：

- 为每个绑定列返回数组形式的行集数据（一行或多行）。
- 根据滚动类型参数的设置滚动结果。

SQLExtendedFetch 与 SQLSetStmtOption 一起使用。

要一次向前获取一行数据，应用程序会调用 SQLFetch。

下表描述了 SQLExtendedFetch 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
01S01	-11006	Error in row
07006	-11013	Restricted data type attribute violation
08S01	-11020	Communication-link failure
22002	-11024	Indicator value required but not supplied
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
24000	-11031	Invalid cursor state
40001	-11037	Serialization failure
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1106	-11086	Fetch type out of range
S1107	-11087	Row value out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11307	In SQLExtendedFetch, only SQL_FETCH_NEXT is supported for SQL_SCROLL_Forward_only cursors

如果发生与整个行集相关的错误，例如 SQLSTATE S1T00（超时），则驱动程序返回 SQL\_ERROR 和适当的 SQLSTATE。行集缓冲区的的内容未定义，游标位置不变。

如果发生与单个行相关的错误，则驱动程序执行以下操作：

- 将行 rgfRowStatus 数组中的元素设置为 SQL\_ROW\_ERROR
- 在错误队列中发布 SQLSTATE 01S01（行中的错误）
- 在错误队列中的 SQLSTATE 01S01（行中的错误）之后，发布零个或多个附加的 SQLSTATE 值

驱动程序处理错误或警告之后，它继续行集中剩余的操作，并返回 SQL\_SUCCESS\_WITH\_INFO。因此，对于与单个行相关的每个错误，错误队列包含 SQLSTATE 01S01（行中的错误），后跟零个或多个附加的 SQLSTATE 值。

驱动程序处理该错误之后，它访存行集中的剩余的行，并返回 SQL\_SUCCESS\_WITH\_INFO。因此，对于返回错误的每一行，错误队列包含 SQLSTATE 01S01（行中的错误），后跟零个或多个附加的 SQLSTATE 值。

如果行集包含已经访存的行，则当首次访存行时，驱动程序不需要返回错误的 SQLSTATE 值。但是，对于最初返回错误的每一行，它需要返回 SQLSTATE 01S01（行中的错误），和 SQL\_SUCCESS\_WITH\_INFO。例如，维护高速缓存的静态游标可能会缓存行状态信息（因此，它可以确定哪一行包含错误），但是不能缓存与错误相关的 SQLSTATE。

错误行不会影响相关游标的移动。例如，假定结果集大小为 100，行集大小为 10。如果当前行集是从 11 到 20 的行，并且 11 行的 rgfRowStatus 数组中的元素是 SQL\_ROW\_ERROR，则使用 SQL\_FETCH\_NEXT 访存类型调用 SQLExtendedFetch 仍然返回 21 到 30 行。

如果驱动程序返回任何警告，例如 SQLSTATE 01004（数据截断），则在返回适用于整个行或行集中位置行的警告，然后返回特定行的错误信息。它返回关于这些行的任何其它错误信息的特定行的警告。

9. 3. 22 SQLFetch（仅限核心级别）

SQLFetch 访存结果集的一行数据。

驱动程序返回使用 SQLBindCol 绑定到存储位置的所有列的数据。

下表描述了 SQLFetch 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
07006	-11013	Restricted data-type attribute violation
08S01	-11020	Communication-link failure
22002	-11024	Indicator value required but not supplied
22003	-11025	Numeric value out of range

SQLSTATE	错误值	错误消息
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
24000	-11031	Invalid cursor state
40001	-11037	Serialization failure
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired

### 9. 3. 23 SQLForeignKeys（仅限二级）

SQLForeignKeys 返回外键列表。

SQLForeignKeys 返回以下任一项：

- 指定表中外键列表（指定表中的指向其它表中主键的列）
- 其它表中引用指定表中主键的外键列表

驱动程序将返回每个列表作为在指定的 hstmt 上设置的结果。

下表描述了 SQLForeignKeys 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication link failure
24000	-11031	Invalid cursor state
IM001	-11040	Driver does not support this function
S1000	-11060	General error
S1001	-11061	Memory allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function sequence error
S1090	-11071	Invalid string or buffer length

SQLSTATE	错误值	错误消息
S1C00	-11092	Driver not capable
S1T00	-11094	Timeout expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

9. 3. 24 **SQLFreeConnect**（仅限核心级别）

SQLFreeConnect 释放连接句柄并释放所有与句柄相关联的内存。

下表描述了 SQLFreeConnect 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
S1000	-11060	General error
S1010	-11067	Function-sequence error

9. 3. 25 **SQLFreeEnv**（仅限核心级别）

SQLFreeEnv 释放环境句柄和与环境句柄相关联的内存。

下表描述了 SQLFreeEnv 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
S1000	-11060	General error
S1010	-11067	Function-sequence error

9. 3. 26 **SQLFreeStmt**（仅限核心级别）

SQLFreeStmt 停止与特定 *hstmt* 关联的句柄，关闭与 *hstmt* 关联的打开游标，丢弃暂挂的结果，以及（可选）释放所有与语从句柄关联的资源。

下表描述了 SQLFreeStmt 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1092	-11073	Option type out of range

9. 3. 27 SQLGetConnectOption（仅限一级）

SQLGetConnectOption 返回当前连接选项的设置。

下表描述了 SQLGetConnectOption 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08003	-11017	Connection not open
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1092	-11073	Option type out of range
S1C00	-11092	Driver not capable

9. 3. 28 SQLGetCursorName（仅限核心级别）

SQLGetCursorName 返回与指定的 hstmt 相关联的游标名称。

下表描述了 SQLGetCursorName 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1015	-11070	No cursor name available
S1090	-11071	Invalid string or buffer length

9. 3. 29 SQLGetData（仅限一级）

SQLGetData 返回当前行中单个未绑定列的结果数据。

应用程序在调用 SQLGetData 之前,必须调用 SQLFetch 或 SQLExtendedFetch 和(可选) SQLSetPos 来定位一行数据的游标。对某些列可以使用 SQLBindCol, 对相同行中的其它列使用 SQLGetData。该函数可以用于从字符、二进制或数据源特定的数据类型(例如,来自 SQL\_LONGVARBINARY 或 SQL\_LONGVARCHAR 列的数据)的列中检索字符或二进制数据。

下表描述了 SQLGetData 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
07006	-11013	Restricted data- type attribute violation
08S01	-11020	Communication-link failure
22002	-11024	Indicator value required but not supplied
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime-field overflow
22012	-11028	Division by zero
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1003	-11063	Program type out of range
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

### 9. 3. 30 SQLGetFunctions (仅限一级)

SQLGetFunctions 返回有关驱动程序是否支持指定函数的信息。

下表描述了 SQLGetFunctions 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-1101	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1095	-11076	Function type out of range

9. 3. 31 SQLGetInfo（仅限一级）

SQLGetInfo 返回与 hdbc 关联的驱动程序和数据源的一般信息。

下表描述了 SQLGetInfo 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
08003	-11017	Connection not open
22003	-11025	Numeric value out of range
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1090	-11071	Invalid string or buffer length
S1096	-11077	Information type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

9. 3. 32 SQLGetStmtOption（仅限一级）

SQLGetStmtOption 返回当前语句选项的设置。

下表描述了 SQLGetStmtOption 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error



SQLSTATE	错误值	错误消息
S1011	-11068	Operation invalid at this time
S1092	-11073	Option type out of range
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable

9. 3. 33 SQLGetTypeInfo（仅限一级）

SQLGetTypeInfo 返回有关数据源支持的数据类型的信息。

驱动程序以 SQL 结果集的格式返回信息。

下表描述了 SQLGetTypeInfo 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1004	-11064	SQL data type out of range
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11305	SQLGetTypeInfo supported for FORWARD_ONLY cursors
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

9. 3. 34 SQLMoreResults（仅限二级）

SQLMoreResults 确定在包含 SELECT 、UPDATE 、INSERT 或 DELETE 语句的 hstmt 上是否有更多结果可用。如果是，则初始化这些结果的处理。

下表描述了 SQLMoreResults 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

9. 3. 35 SQLNativeSql（仅限二级）

SQLNativeSql 返回驱动程序翻译的 SQL 字符串。

下表描述了SQLNativeSql 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
08003	-11017	Connection not open
37000	-11035	Syntax error or access violation
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1090	-11071	Invalid string or buffer length
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

用法

以下示例显示 SQLNativeSql 返回包含刻度函数 LENGTH 的输入 SQL 字符串的信息：

SELECT {fn LENGTH(NAME)} FROM EMPLOYEE

GBase 8s 可能返回以下翻译的 SQL 字符串：

SELECT length(NAME) FROM EMPLOYEE

9. 3. 36 SQLNumParams（仅限二级）

SQLNumParams 返回 SQL 语句中参数的数量。

下表描述了SQLNumParams 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired

9. 3. 37 SQLNumResultCols（仅限核心级别）

SQLNumResultCols 返回结果集中的列数。

下表描述了 SQLNumResultCols 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired

SQLNumResultCols 可以返回在 SQLPrepare 之后，SQLExecute 之前调用 SQLNumResultCols 时，SQLPrepare 或 SQLExecute 返回的任何 SQLSTATE ，这取决于数据源何时评估与 hstmt 关联的 SQL 语句。

9. 3. 38 SQLParamData（仅限一级）

当执行语句时，同时使用 SQLParamData 和 SQLPutData 来提供参数数据。

下表描述了 SQLParamData 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
22026	-11029	String data, length mismatch
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error

SQLSTATE	错误值	错误消息
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.

如果在发送 SQL 语句中的参数数据时调用 SQLParamData, 则会返回被调用来执行语句的函数 (SQLExecute 或 SQLExecDirect) 所返回的任何 SQLSTATE。如果在为使用 SQLSetPos 更新或添加的列发送数据时调用它, 则它返回 SQLSetPos 返回的任何 SQLSTATE。

## 9.4 SQLParamOptions（仅限核心和二级）

SQLParamOptions 允许应用程序为由 SQLBindParameter 分配的一组参数指定多个值。

为一组参数指定多个值的函数对用于批量插入和其它需要数据源多次使用各种参数值处理相同 SQL 语句的工作很有用。例如, 应用程序可以为与 INSERT 语句关联的参数集指定三个值, 然后再次执行 INSERT 语句来执行三次插入操作。

下表列出了 SQLParamOptions 返回的 SQLSTATE 值, 并解释了该函数内容中的每个值: 在驱动程序管理器返回的每个 SQLSTATE 的描述之前都包含注释 (DM)。除非另有说明, 否则与每个 SQLSTATE 值关联的返回码都是 SQL\_ERROR。

SQLSTATE	错误值	错误消息
01000		General warning
S1000		General error
S1001		Memory-allocation failure
S1010		Function-sequence error
S1107		Row value out of range

### 9.4.1 SQLPrepare

SQLPrepare 准备要执行的 SQL 字符串。

下表描述了 SQLPrepare 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
21S01	-11021	Insert value list does not match column list
21S02	-11022	Degree of derived table does not match column list

SQLSTATE	错误值	错误消息
22005	-11026	Error in assignment
24000	-11031	Invalid cursor state
34000	-11034	Invalid cursor name
37000	-11035	Syntax error or access violation
42000	-11038	Syntax error or access violation
S0001	-11053	Base table or view already exists
S0002	-11054	Base table not found
S0011	-11055	Index already exists
S0012	-11056	Index not found
S0021	-11057	Column already exists
S0022	-11058	Column not found
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

#### 9. 4. 2 SQLPrimaryKeys（仅限二级）

SQLPrimaryKeys 返回组成表的主键的列的名称。

驱动程序将此信息返回为结果集。该函数不支持在单个调用中返回多个表的主键。

下表描述了 SQLPrimaryKeys 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
----------	-----	------

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

#### 9.4.3 SQLProcedureColumns（仅限二级）

SQLProcedureColumns 返回输入和输出参数的列表，和组成特定过程的结果集的列。

驱动程序将返回的信息作为特定 hstmt 的结果集。

下表描述了 SQLProcedureColumns 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication link failure
24000	-11031	Invalid cursor state
IM001	-11040	Driver does not support this function
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function sequence error
S1090	-11071	Invalid string or buffer length

SQLSTATE	错误值	错误消息
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

#### 9.4.4 SQLProcedures（仅限二级）

SQLProcedures 返回特定数据源中过程名称的列表，

Procedure 是一个通用术语，用于描述可执行对象，或可以使用输入和输出参数启动的命名实体，并且可以返回类似于 SELECT 语句返回结果的结果集。

下表描述了 SQLProcedures 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error

SQLSTATE	错误值	错误消息
S1000	-11323	The statement contained an escape clause not supported by this database driver

9. 4. 5 SQLPutData（仅限一级）

SQLPutData 允许应用程序在执行时将参数或列的数据发送到驱动程序。

该函数可以将字符或二进制数据值以字符、二进制或数据源指定的数据类型（如 SQL\_LONGVARBINARY 或 SQL\_LONGVARCHAR 的参数）的形式发送到列。

下表描述了 SQLPutData 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01004	-11003	Data truncated
07S01	-11014	Invalid use of default parameter
08S01	-11020	Communication-link failure
22001	-11023	String data right truncation
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime-field overflow
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired

**重要：** 应用程序可以使用 SQLPutData 将字符 C 数据的各个部分发送到具有字符、二进制或数据源指定的数据类型的列，或者将二进制 C 数据发送到具有字符、二进制或数据源指定的数据类型的列。如果在任何其他条件下多次调用 SQLPutData，则会返回 SQL\_ERROR 和 SQLSTATE 22003（数值超出范围）。

9. 4. 6 SQLRowCount（仅限核心级别）

SQLRowCount 通过在 SQLSetPos 中的 SQL\_UPDATE、SQL\_ADD 或 SQL\_DELETE 操作返回受 UPDATE、INSERT 或 DELETE语句影响的行数。



下表描述了 SQLRowCount 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error

9. 4. 7 SQLSetConnectOption（仅限一级）

SQLSetConnectOption 设置管理连接方面的选项。

下表描述了 SQLSetConnectOption 的 SQLSTATE 和错误消息。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01S02	-11007	Option value changed
08002	-11016	Connection in use
08003	-11017	Connection not open
08S01	-11020	Communication-link failure
IM009	-11048	Unable to load translation shared library (DLL)
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1011	-11068	Operation invalid at this time
S1092	-11073	Option type out of range
S1C00	-11092	Driver not capable
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

当 fOption 是语句选项时, SQLSetConnectOption 可以返回 SQLSetStmtOption 返回的任何 SQLSTATE。

9. 4. 8 SQLSetCursorName（仅限核心级别）

SQLSetCursorName 将游标描述与活动的 hstmt 关联。

如果应用程序不能调用 `SQLSetCursorName`，那么驱动程序会根据 `SQL` 语句处理的需要生成游标名称。

下表描述了 `SQLSetCursorName` 的 `SQLSTATE` 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
24000	-11031	Invalid cursor state
34000	-11034	Invalid cursor name
3C000	-11036	Duplicate cursor name
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length

9. 4. 9 `SQLSetStmtOption`（仅限一级）

`SQLSetStmtOption` 设置与 `hstmt` 相关的选项。

要为所有与特定 `hdbc` 关联的语句设置选项，应用程序可以调用 `SQLSetConnectOption`。

下表描述了 `SQLSetStmtOption` 的 `SQLSTATE` 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
01S02	-11007	Option value changed
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1011	-11068	Operation invalid at this time
S1092	-11073	Option type out of range
S1C00	-11092	Driver not capable

9. 4. 10 `SQLSpecialColumns`（仅限一级）

`SQLSpecialColumns` 检索有关列的信息。

`SQLSpecialColumns` 检索有关指定表的以下信息：

- 唯一标识表中的一行的最佳列集
- 通过事务更新行中的值时，自动更新的列

下表描述了 SQLSpecialColumns 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1097	-11078	Column type out of range
S1098	-11079	Scope type out of range
S1099	-11080	Nullable type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

9. 4. 11 SQLStatistics（仅限一级）

SQLStatistics 检索有关单个表的统计信息和与该表相关联的索引信息。

驱动程序将此消息作为结果集返回。

下表描述了 SQLStatistics 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure

SQLSTATE	错误值	错误消息
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory- allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1100	-11081	Uniqueness option type out of range
S1101	-11082	Accuracy option type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

#### 9. 4. 12 SQLTablePrivileges（仅限二级）

SQLTablePrivileges 返回表和与每个表相关的特权的列表。

驱动程序将此信息作为指定 hstmt 上的结果集。

下表描述了 SQLTablePrivileges 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired

SQLSTATE	错误值	错误消息
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

#### 9. 4. 13 SQLTables（仅限一级）

SQLTables 返回存储在指定数据源中表的名称。

驱动程序将此信息作为结果集返回。

下表描述了 SQLTables 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

9. 4. 14 SQLTransact（仅限核心级别）

SQLTransact 为与连接关联的 hstmts 上的所有活动操作请求提交或回滚。

SQLTransact 还可以请求对与 henv 关联的所有连接执行提交或回滚操作。

下表描述了 SQLTransact 的 SQLSTATE 和错误值。

SQLSTATE	错误值	错误消息
01000	-11001	General warning
08003	-11017	Connection not open
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1012	-11069	Invalid transaction operation code specified
S1C00	-11092	Driver not capable
08S01	-11301	A protocol error has been detected. Current connection is closed.

10 Unicode

本章提供 Unicode 标准的简要概述并描述如何在 ODBC 应用程序中使用它。

10.1 Unicode 概述

Unicode 是字符编码标准，它提供了每种主要语言中每个字符的使用方法。

在 Unicode 标准中，每个字符被分配了一个唯一的数字值和名称。这些值在跨多个平台的应用程序之间是一致的。

尽管 Unicode 提供了一种使用多种语言表示文本的一种方式，但是也有不同的版本，为每个字符提供不同的数据大小。

10.1.1 Unicode 版本

尽管 Unicode 提供了一种使用多种语言表示文本的一种方式，但是也有不同的版本，为每个字符提供不同的数据大小。

以下列表描述了在 GBase 8s ODBC 应用程序中支持的版本。

- UCS-2

ISO 编码标准，将 Unicode 字符映射到每个 2 字节。UCS-2 是 Windows(TM) 上通用的编码标准。

GBase AIX 平台的 GBase 8s ODBC Driver 支持 UCS-2 编码。Windows(TM) 的 GBase 8s ODBC Driver 仅支持 UCS-2。

- UCS-4

ISO 编码标准，将 Unicode 字符映射到每个 4 字节。

UNIX(TM) 平台上，GBase 8s ODBC Driver 支持 UCS-4。

- UTF-8

基于单字节（8 位）的编码标准。UTF-8 定义了一种将所有 Unicode 字符转换为可变长度（1 - 4）字节编码的机制。

所有的 UNIX(TM) 应用程序的 GBase 8s ODBC Driver 使用 UTF-8 编码，连接 Data Direct（以前称为 Merant）驱动程序管理器，

ASCII 和 UTF-8 之下，7 位 ASCII 字符句具有相同的编码。这样做的好处是 UTF-8 可以与大量现有的软件一起使用而不需要大量的修改。

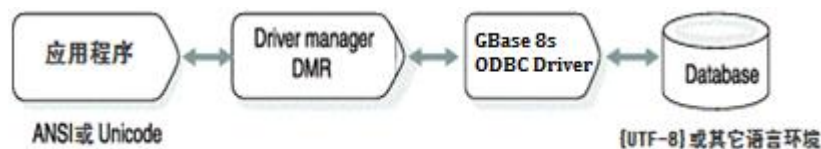
**重要：** 在使用 Unicode 的应用程序中，驱动程序执行从 Unicode 到数据库语言环境的代码集转换工作。UTF-8 是唯一可以设置为客户端语言环境的 Unicode 代码集类型。

## 10.2 ODBC 应用程序中的 Unicode

查看典型的 ODBC 应用程序架构。

下图显示了具有驱动管理器和 GBase 8s ODBC Driver 的典型的 ODBC 应用程序架构。

图: 典型的 ODBC 应用程序架构



在此场景中，如果应用程序启用 Unicode 的 API，则必须将其连接到启用 Unicode 的 GBase 8s ODBC Driver（3.8 版本或更高），来确保数据没有丢失。如果应用程序调用 ANSI ODBC API，则应用程序可以链接到启用 Unicode 的驱动程序或 ANSI 驱动程序。

GBase 8s ODBC Driver 继续支持 GBase 8s GLS。因此，在字符缓冲区中获取的所有数据都可以在客户端语言环境代码集中获取。只有使用宽字符缓冲区获取的数据才使用 Unicode。

在 Windows(TM) 上，如果 ODBC 驱动程序没有启用 Unicode，则 ODBC Driver Manager 会将所有的 Unicode API 函数调用映射到 ANSI ODBC API。

如果 ODBC 驱动程序启用了 Unicode，则 Windows(TM) ODBC Driver Manager (4.0 版本或更高) 将所有的 ANSI ODBC API 映射到 Unicode ODBC API。用于 UNIX(TM) 的 Data Direct (之前称为 Merant) 驱动程序管理器也使用这种方式。

**重要：** CSDK Version 2.70 中有两个 ODBC 驱动程序。一个只有 ANSI API (称为 ANSI ODBC Driver Version 3.34)，另一个有 ANSI 和 UNICODE API (称为 Unicode ODBC Driver Version 3.80)。对于 CSDK 2.80 或更高版本，只有一个支持 ANSI 和 UNICODE API 的 ODBC 驱动程序。

**重要：** UNIX 平台的 GBase 8s Driver Manager Replacement (DMR) 不会在 Unicode 和 ANSI API 之间进行映射。

## 10.3 ODBC 应用程序中的 Unicode

本节提供有关在 GBase 8s ODBC 应用程序中编译和配置 Unicode 的详细信息。

### 10.3.1 配置

因为 GBase 8s ODBC Driver 在 UNIX(TM) 平台上支持不同类型的 Unicode，因此应用程序使用的 Unicode 类型必须在 `odbc.ini` 文件的 ODBC 部分中指明。

在 ODBC 部分中指示 Unicode 的类型，如下所示：

```
[ODBC]
.
.
.
UNICODE=UCS-4
```

**重要：** 启用 Unicode 的应用程序必须在 `odbc.ini` 文件中指示 Unicode 的类型。如果未在 `odbc.ini` 中设置 Unicode 参数，则缺省类型为 UCS-4。

要求所有的 UNIX(TM) ODBC 应用程序必须如下设置 `odbc.ini` 文件中的 Unicode 类型：

- UNIX(TM) 上 (包括 AIX 64 位) 的 ANSI ODBC 应用程序必须设置 `UNICODE=UCS-4`
- GBase AIX 32 位上的 ANSI ODBC 应用程序必须设置 `UNICODE=UCS-2`
- 使用 Data Direct (之前称为 Merant) ODBC 驱动程序管理器的 ANSI ODBC 应用程序不会在文件中指示不是 UTF-8 的 Unicode 类型。

下表提供了 `odbc.ini` 设置的概述：

平台	驱动程序管理器	odbc.ini 设置
AIX	Data Direct	UTF-8
AIX 32 - 位	DMR or none	UCS-2
AIX 64 - 位	Data Direct	UTF-8
UNIX(TM)	Data Direct	UTF-8
UNIX(TM)	DMR 或无	UCS-4



平台	驱动程序管理器	odbc.ini 设置
Windows(TM)	Windows(TM) ODBC Driver Manager	N/A

**重要：**

如果下列条件都满足，则设置会自动重置，且不会发出警告或错误消息：

- 该应用程序是一个 ANSI 应用程序。
- 正在与 DMR 链接或没有链接。
- odbc.ini 文件中的 Unicode 设置与表中显示的值不匹配。

## 10.4 支持的 Unicode 函数

GBase 8s ODBC Driver 支持接受指向字符串或 SQLPOINTER 参数指针的所有函数的 ANSI 和 Unicode 版本。

以下列表描述了支持的两种类型的函数：

**ODBC “A” 函数**

一般的 ODBC 函数，接受单字节（ASCII）数据作为所有字符/字符串参数的输入。

**ODBC “W” 函数**

接受 “宽字符” 作为所有的字符/字符串参数的输入的 Unicode 函数。

ODBC 规范定义了具有 wchar\_t 数据类型的三个函数。该数据类型是标准 C 库宽字符数据类型。

GBase 8s ODBC Driver 支持下列 Unicode “wide” 函数：

- SQLColAttributeW
- SQLColAttributesW
- SQLConnectW
- SQLDescribeColW
- SQLErrorW
- SQLExecDirectW
- SQLGetConnectAttrW
- SQLGetCursorNameW
- SQLSetDescFieldW
- SQLGetDescFieldW
- SQLGetDescRecW
- SQLGetDiagFieldW
- SQLGetDiagRecW
- SQLPrepareW
- SQLSetConnectAttrW
- SQLSetCursorNameW
- SQLColumnsW

- SQLGetConnectOptionW
- SQLGetTypeInfoW
- SQLSetConnectOptionW
- SQLSpecialColumnsW
- SQLStatisticsW
- SQLTablesW
- SQLDataSourcesW
- SQLDriverConnectW
- SQLBrowseConnectW
- SQLColumnPrivilegesW
- SQLGetStmtAttrW
- SQLSetStmtAttrW
- SQLForeignKeysW
- SQLNativeSqlW
- SQLPrimaryKeysW
- SQLProcedureColumnsW
- SQLProceduresW
- SQLTablePrivilegesW
- SQLDriversW

从 Version 4.10 起, SQLGetDiagRecW 函数 BufferLength 参数定义为: *MessageText* 缓冲区的长度, 以字符为单位。

**GBASE<sup>®</sup>**

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



微信二维码

