SUNDB数据库管理系统 V5.0-开发者手册

Developer Manual

目录

1. Database Connection 1

1.1 特征 1

1.2 选择执行成员 2

1.3 Global Session 4

1.4 约束事项 6

1.5 设置 8

2. ODBC 9

2.1 SUNDB ODBC Driver概述 9

2.2 数据源构成 13

2.3 GLOBAL CONNECTION 36

2.4 目录函数 48

2.5 非标准数据类型 54

2.6 ODBC API References 69

2.7 XA API References 604

3. JDBC 631

3.1 概要 631

3.1 功能详情 638

3.2 JDBC API References 695

4. Embedded SQL 970

4.1 Precompiler 970

4.2 Embedded SQL 986

4.3 Advanced Topic 1178

4.4 Embedded SQL Reference 1243

5. PDO 1281

5.1 PDO概要 1281

5.2 安装/构成 1281

5.3 使用 1285

5.4 示例 1285

6. PyDBC 1304

6.1 SUNDB PyDBC 1304

6.2 API Reference 1309

6.3 Exception 1331

6.4 Data Type 1333

7. Ruby 1338

7.1 概要 1338

7.2 安装 1338

7.3 示例 1340

7.4 ActiveRecord的使用示例 1346

8. Hibernate 1351

8.1 概要 1351

8.2 联动 1351

8.3 示例 1354

1.Database Connection

1.1 特征

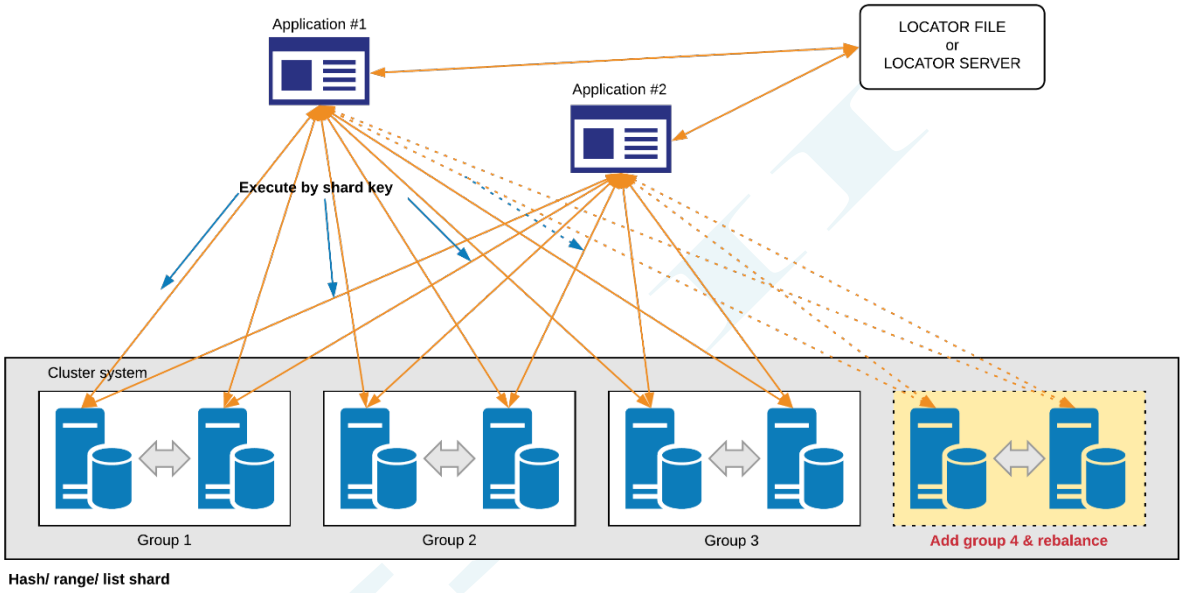


Figure 1-1 GLOBAL CONNECTION

Global connection功能是考虑到数据locality的事务性能优化方案

通常connection连接一个成员而global connection连接所有成员使用global connection的应用程序执行查询访问的数据最多的成员的查询从而提高性能

Global connection可用于hashrangelist sharding等所有方式此时不需要变更应用程序

执行online scale-out时用户不需要追加考虑新的节点应用程序可自动访问新节点运行此节点

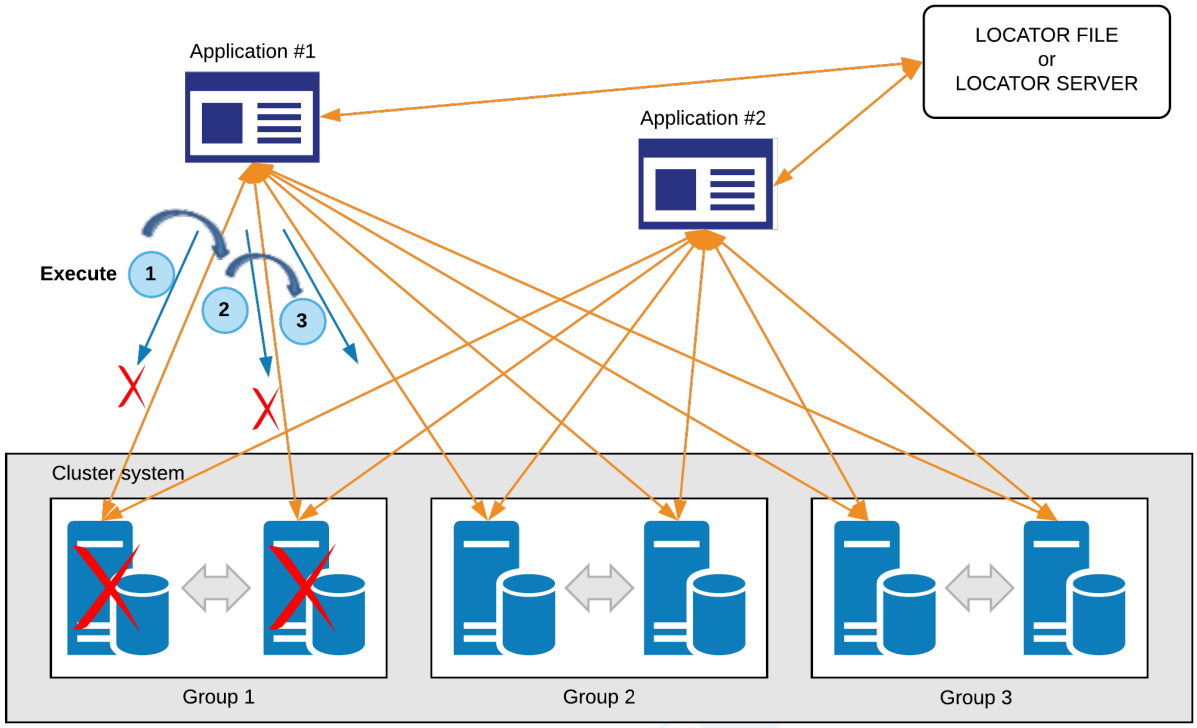


Figure 1-2 GLOBAL CONNECTION HA (high availability)

执行SQL时所选节点发生故障时通过其他组的其他节点执行SQL所选的组的所有节点发生故障时通过其他组执行SQL

恢复发生故障的节点时在online状态下自动重新访问该节点不仅如此用户也可以使用以下语句使应用程序重新执行访问

```
ALTER SYSTEM RECONNECT GLOBAL CONNECTION
```

1.2 选择执行成员

执行成员按照事务为单位进行选择没有事务时首次执行DML查询时按照sharding key选择组组内

的执行成员取决于LOCALITY_MEMBER_POLICY参数之后在COMMIT或ROLLBACK之前执行的所有查询在所选成员中执行如果首次查询未选择根据sharding key的合适的组时根据LOCALITY_GROUP_POLICY 参数决定组

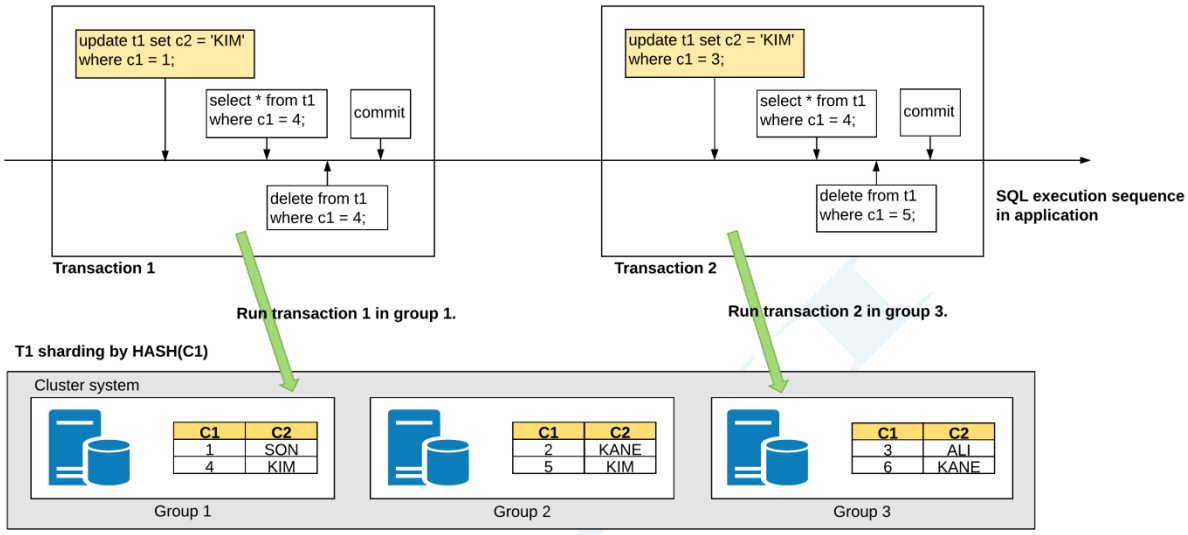


Figure 1-3 选择成员

上图中transaction1的UPDATE查询根据sharding key在group1执行因此后续COMMIT 之间的所有查询均在group 1执行Transaction2的UPDATE查询在group3执行即使后续查询不适合在group 3执行但在COMMIT之前所有查询均在group3执行

在没有事务的情况下只读专用查询(SELECT)与其他查询相同根据sharding key选择执行成员但后续执行的查询未必在上一次选择的成员中执行

即包含在事务的所有查询仅在一个成员中执行与事务无关的查询按照查询为单位选择成员

1.3 Global Session

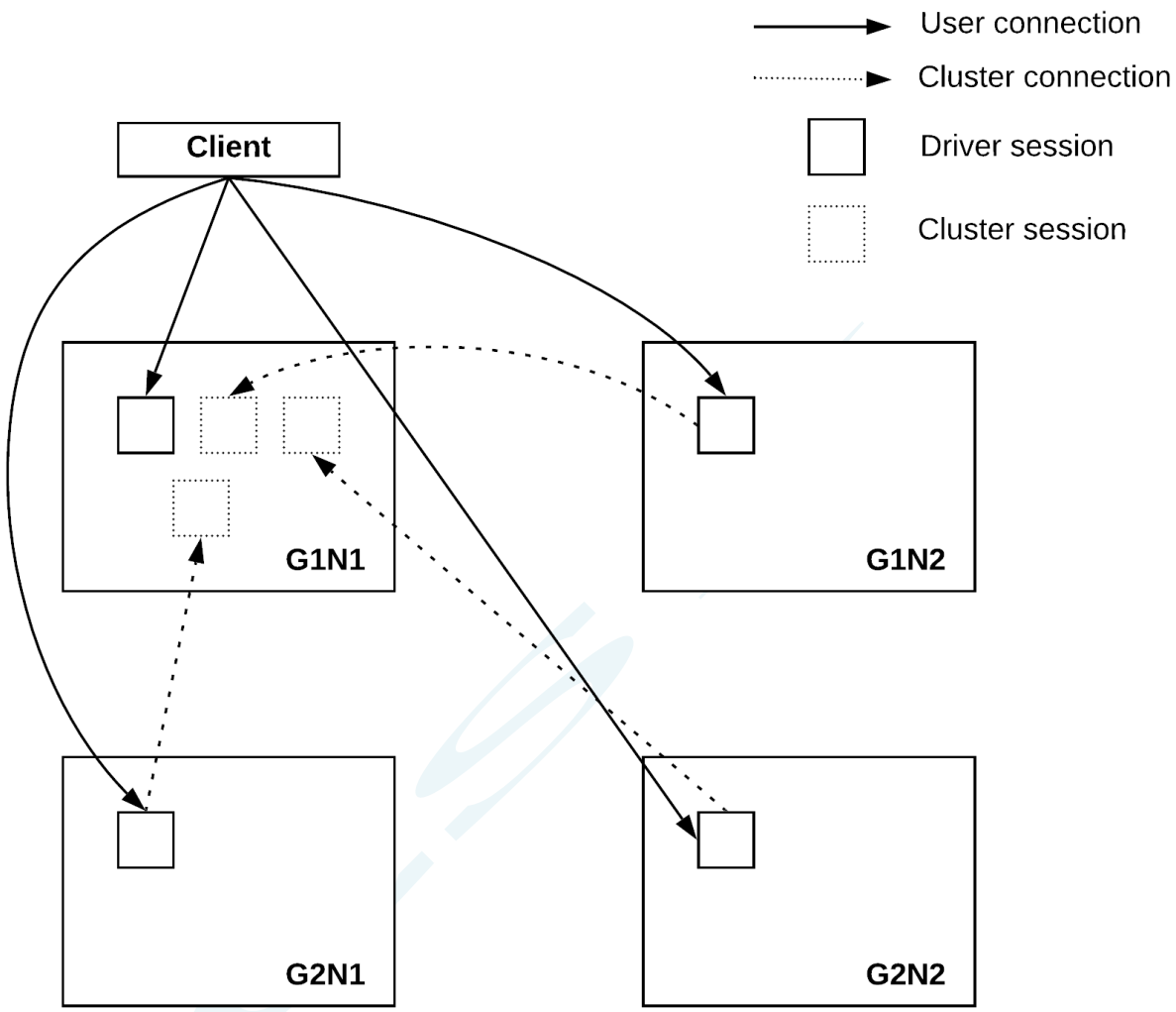


Figure 1-4 从global connection衍生的cluster session

与应用程序直接连接的会话叫做driver session从driver session到其他成员的会话叫做cluster sessionGlobal connection在所有成员中创建driver session根据所需向其他成员创建cluster session由于global connection的特征可在一个成员中创建多个cluster session

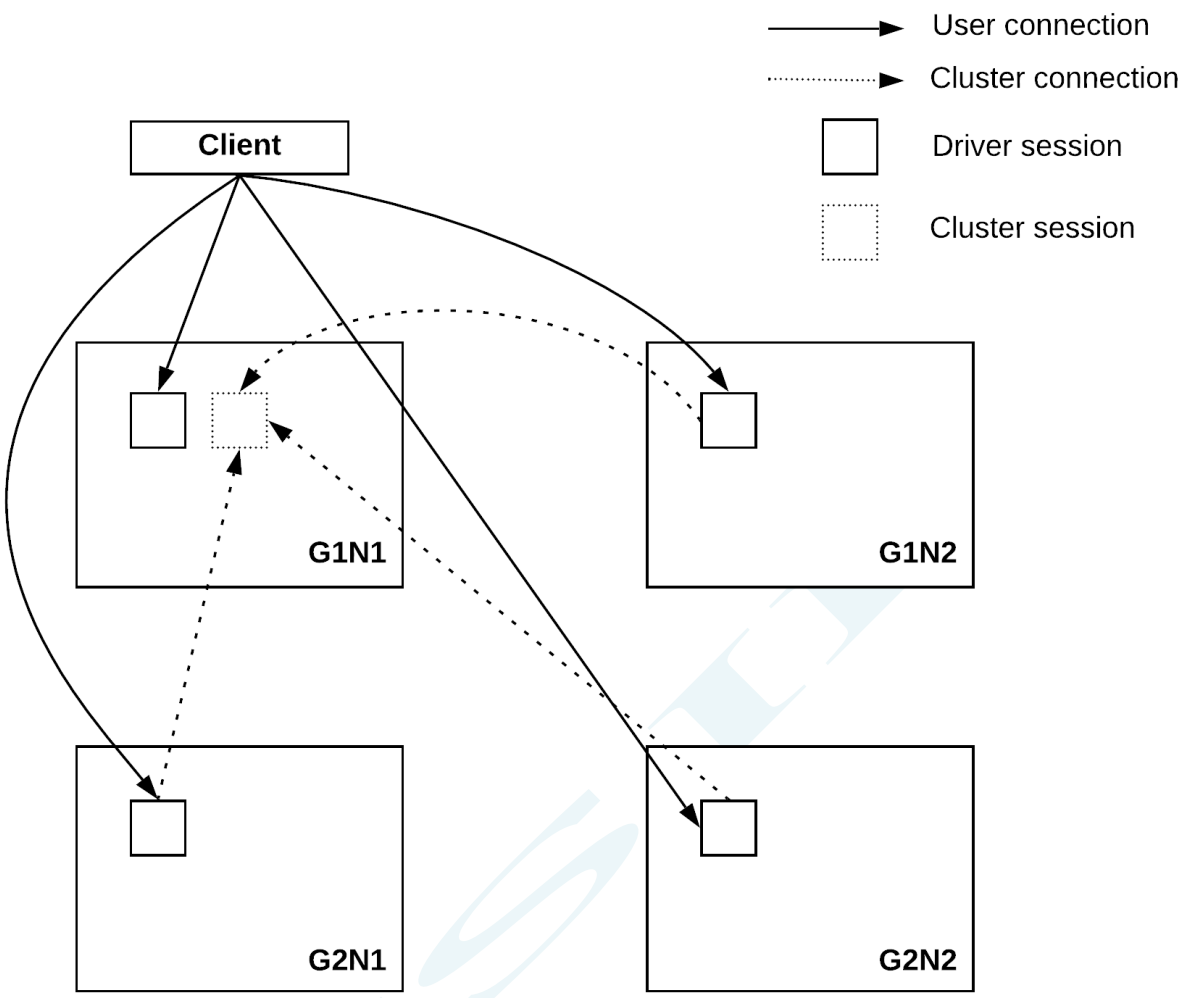


Figure 1-5 Global session

Global session是在global connection创建的cluster session共享一个会话并由此提高资源效率的功能Global connection不使用global session时根据组和成员的增加而增加cluster session相反使用global session时即使增加组和成员也不会增加cluster session

Global session功能仅可在global connection使用普通connection无法使用

1.4 约束事项

使用global connection时SQL语句无法使用拥有session dependenccy的对象（session dependent object）或语句（session dependent clause）或函数（session dependent function）

另外访问多个集群节点的SQL存在于一个事务中时事务中的SQL共同使用第一个SQL的sharding key选择的集群节点

Global connection中查询仅在以prepare execute执行时考虑数据locality以direct execute执行时在任意节点执行

Session Dependent Object

在SQL语句使用session dependent object时不支持global connection

- Global temporary table

Session Dependent Clause

在SQL语句使用session dependent clause时不支持global connection

- 所有@domain 相关语句

Session Dependent Function和Pseudo Column

在SQL语句使用session dependent信息时不支持global connection

- CURRVAL(sequence), sequence.CURRVAL
- UUID()
- VERSION()
- SESSION_ID()
- SESSION_SERIAL()
- USER_ID()
- LAST_IDENTITY_VALUE()
- STATEMENT_VIEW_SCN()
- STATEMENT_VIEW_SCN_GCN()
- STATEMENT_VIEW_SCN_DCN()
- STATEMENT_VIEW_SCN_LCN()
- LOCAL_GROUP_ID()
- LOCAL_MEMBER_ID()
- LOCAL_GROUP_NAME()
- LOCAL_MEMBER_NAME()
- CLUSTER_GROUP_ID
- CLUSTER_GROUP_NAME
- CLUSTER_MEMBER_ID
- CLUSTER_MEMBER_NAME
- CLUSTER_SHARD_ID

使用Global Session功能的情况

SQL语句不支持Data Definition Language (DDL)

1.5 设置

详细内容参考如下

- **ODBC Global Connection**
- **JDBC Global Connection**

2.ODBC

2.1 SUNDB ODBC Driver概述

SUNDB ODBC Driver的概念

ODBC（开放数据库互连Open Database Connectivity）是数据库API（Application Programming Interface）的配置Microsoft ODBC 3.0版本基于International Standards Organization/ International Electromechanical Commission (ISO/ IEC)和X/open的分级调用接口（CLI）的推荐配置ODBC使用C libarary函数支持SQL语句应用程序通过调用该函数实现ODBC功能

ODBC架构由执行以下功能的4个要素组成

组成要素	功能
应用程序	调用与ODBC数据源通信的ODBC函数传递SQL语句后处理结果集
驱动管理器	管理应用程序以及应用程序中使用的所有ODBC驱动程序之间的通信
驱动	处理应用程序中的所有ODBC调用连接数据源并从应用程序将SQL语句传递至数据源后向应用程序返回结果必要时驱动将应用程序传递的ODBC SQL转换为数据源使用的基本SQL
数据源	包含驱动访问数据库数据时所需的所有信息

使用ODBC应用程序可以执行以下操作

- 连接数据源
- 向数据源传输SQL语句
- 在数据源处理SQL语句的结果
- 错误及信息处理
- 断开与数据源的连接

ODBC构成要素概要

包含驱动管理器的SUNDB ODBC Driver

以下为系统包含驱动管理器的软件架构此时应用程序需要链接到驱动管理器库

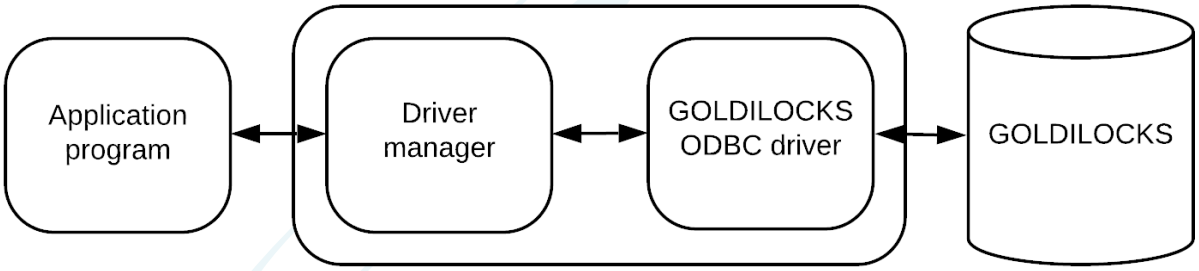


Figure 2-1 包含驱动管理器的SUNDB ODBC driver

不包含驱动管理器的SUNDB ODBC Driver

下图为系统不包含驱动管理器并使用SUNDB ODBC driver的架构此时应用程序需要直接链接到SUNDB ODBC driver库

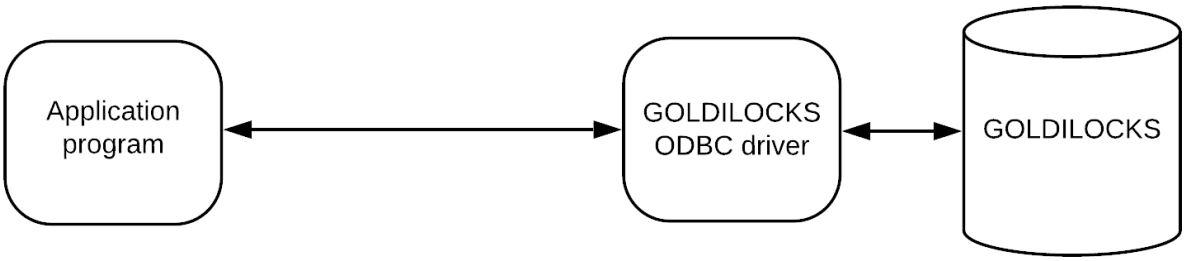


Figure 2-2 不包含驱动管理器的SUNDB ODBC driver

SUNDB ODBC Driver的使用

头文件（header file）

执行SUNDB ODBC driver需要有$SUNDB_HOME/include中设置的sundb.h文件此文件定义SUNDB ODBC driver的常数和类型提供SUNDB ODBC driver函数的原型

库（Libraray）

不使用驱动管理器的应用程序需要链接SUNDB ODBC driver库的静态或共享文件

UNIX

文件名	说明
libsundb.a	包含DA与CS的库的静态版本
libsundba.a	DA专用库的静态版本
libsundbas.so	DA专用库的共享版本

文件名	说明
libsundbc.a	CS专用库的静态版本
libsundbcs-ul32.so	将SQLLEN识别为4字节的64Bit CS 专用库的共享版本
libsundbcs-ul64.so	将SQLLEN识别为8字节的64Bit CS 专用库的共享版本
libsundbcs.so	32Bit CS 专用库的共享版本
libsundbs.so	包含DA与CS的库的共享版本

Table 2-1 SUNDB UNIX ODBC driver库文件

Windows

SUNDB Windows ODBC driver库只提供CS库文件

文件名	说明
sundbcs-ul64.dll	将SQLLEN识别为8字节的64Bit CS 专用库的共享版本
sundbcs.dll	32Bit CS 专用库的共享版本
sundbsetup32.dll	设置32Bit ODBC driver manager的库
sundbsetup64.dll	设置64Bit ODBC driver manager的库

Table 2-2 SUNDB Windows ODBC driver库文件

2.2 数据源构成

UNIX中的DSN设置

odbcinst.ini文件

odbcinst.ini是已安装的ODBC驱动程序的配置文件

- unixODBC

```
% odbcinst -j

unixODBC 2.3.4

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

SYSTEM DATA SOURCES: /etc/odbc.ini

FILE DATA SOURCES..: /etc/ODBCDataSources

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

SQLULEN Size.......: 8

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

SQLSETPOSIROW Size.: 8
```

- iODBC

```
% iodbc-config --odbcinstini

/etc/odbcinst.ini
```

ODBC Driver配置

odbcinst.ini文件的ODBC驱动程序配置部分描述驱动程序属性值和目录每个安装的驱动程序的驱动程序名称下有注册信息部分

```
[driver_name]
Description = driver_description
Driver = driver_library_path
Setup = setup_library_path
FileUsage = file_usage
```

下表描述了驱动程序配置部分中的关键字

关键字	说明
Description	说明驱动程序的字符串
Driver	驱动library路径
Setup	安装library路径
FileUsage	显示基于文件的驱动在DSN中直接处理文件的方法的文字

以下为查看SUNDB ODBC驱动配置相关信息的示例

```
[SUNDB ODBC Driver]
Description= SUNDB ODBC Driver
Driver = /home/sundb/home/lib/libsundbcs-ul64.so
Setup = /home/sundb/home/lib/libsundbcs-ul64.so
```

```
FileUsage = 0
```

odbc.ini文件

odbc.ini文件是应用程序连接的DSN的配置文件分为用户DSN与系统DSN通常用户DSN文件为~/.odbc.ini文件系统DSN文件为 /etc/odbc.ini文件

- unixODBC

```
% odbcinst -j
unixODBC 2.3.4
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /home/sundb/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

- iODBC

```
% iodbc-config --odbcini
/etc/odbc.ini
```

数据源配置

odbc.ini文件的数据源配置部分中说明DSN

```
[data_source_name]

Driver = driver_name

PROTOCOL = {DA | TCP | IPC}

CS_MODE = {default | dedicated | shared}

HOST = host_address

PORT = port_no

PREFER_IPV6 = {0 | 1}

CHARSET = {SQL_ASCII | UTF8 | UHC | GB18030}

TCP_NODELAY = {0 | 1}

ALTERNATE_SERVERS = (HOST=ADDRESS1:PORT=PORT1,HOST=ADDRESS2:PORT=PORT2)

CONNECTION_RETRY_COUNT = retry_count

CONNECTION_RETRY_DELAY = retry_delay

FAILOVER_TYPE = {CONNECTION | SESSION}

FAILOVER_GRANULARITY = {0 | 1 | 2}

FAILOVER_ROUTING_POLICY = {0 | 1}

DATE_FORMAT = date_format_string

TIME_FORMAT = time_format_string

TIME_WITH_TIME_ZONE_FORMAT = timetz_format_string

TIMESTAMP_FORMAT = timestamp_format_string

TIMESTAMP_WITH_TIME_ZONE_FORMAT = timestamptz_format_string

CHAR_LENGTH_UNITS = {BYTE | OCTETS | CHAR | CHARACTERS}

ENABLE_SQLDESCRIBEPARAM = {0 | 1}

ENABLE_SQLBINDPARAMETER_CONSISTENCY_CHECK = {0 | 1}

USE_TARGETTYPE = {0 | 1 | 2}

LOCATOR_DSN = locator_dsn_name
```

```
LOCATOR_SERVICE = locator_service_name

LOCALITY_AWARE_TRANSACTION = {0 | 1}

LOCALITY_GROUP_POLICY = {0 | 1 | 2}

LOCALITY_GROUP_PATH = group_name1, group_name2, group_name3

LOCALITY_MEMBER_POLICY = {0 | 1 | 2 | 3 | 4}

LOCALITY_MEMBER_PATH = member_name1,member_name2, member_name3

DB_HOME = database_home_path

PACKET_COMPRESSION_THRESHOLD = packet_compression_threshold

USE_GLOBAL_SESSION = {0 | 1}

LOGIN_TIMEOUT = login_timeout

TRACE = {0 | 1}

TRACEFILE = file_path_name

TRACE_POLICY={DEFAULT | ERROR}

INCLUDE_SYNONYMS = {0 | 1}

DOT_NET_FOR_ODBC = {0 | 1}


[locator_dsn_name]

FILE = location_file_name

HOST = IP address(v4)

PORT = locator_port

CONNECTION_TIMEOUT = second

ALTERNATE_LOCATORS = (HOST=ADDRESS1:PORT=PORT1,HOST=ADDRESS2:PORT=PORT2)
```

下表为数据源配置部分的关键字

属性	说明
data_source_name	数据源section中指定的数据源
Driver	odbcinst.ini中安装的驱动名
PROTOCOL	与服务器连接的方式 • DA: 无额外的通讯直接连接 • TCP: 使用TCP socket进行通讯 • IPC: 使用共享内存进行通讯并且需要与server在相同设备上第一次连接时为了传输IPC信息会使用TCP连接所以需要设置HOST和PORT仅可使用Dedicated模式进行连接
CS_MODE	设置以dedicated模式访问还是以shared模式连接 如果不使用此设置模式取决于listener的configuration(DEFAULT_CS_MODE)
HOST	HOST IP地址或名称
PORT	连接端口号
PREFER_IPV6	HOST参数为HOST名时IP地址中优先IPv6
TCP_NODELAY	socket TCP_NODELAY选项
UID	用户ID

属性	说明
PWD	用户密码
CHARSET	客户端字符集
ALTERNATE_SERVERS	发生failover时尝试连接的服务器目录每个服务器以逗号","区分 不使用failover功能时不设置ALTERNATE_SERVERS
CONNECTION_RETRY_COUNT	连接失败时尝试连接服务器的次数
CONNECTION_RETRY_DELAY	连接失败时尝试连接的服务器的时间间隔（单位：秒）
FAILOVER_TYPE	• CONNECTION：连接失败时连接至ALTERNATE_SERVERS • SESSION：连接失败或处理statement过程中连接断开时连接至ALTERNATE_SERVERS后恢复statement如果断开连接时没有进行中的事务则failover后执行进行中的statement

属性	说明
FAILOVER_GRANULARITY	• 0：进行failover的过程中发生错误也继续进行failover • 1：进行failover的过程中发生除SQLExeceute()SQLExecDirect()外的错误时failover将失败 • 2：进行failover的过程中发生错误时failover将失败
DATE_FORMAT	DATE类型字符串
TIME_FORMAT	TIME类型字符串
TIME_WITH_TIME_ZONE_FORMAT	TIME WITH TIME ZONE类型字符串
TIMESTAMP_FORMAT	TIMESTAMP类型字符串
TIMESTAMP_WITH_TIME_ZONE_FORMAT	TIMESTAMP WITH TIME ZONE类型字符串
CHAR_LENGTH_UNITS	SQLBindParameter() 中ParameterType为SQL_CHARSQL_VARCHAR时的ColumnSize的单位 • BYTE, OCTETS：以byte为单位 • CHAR, CHARACTERS：以字符为单位

属性	说明
ENABLE_SQLDESCRIBEPARAM	决定是否执行SQLDescribeParam() • 0：不支持SQLDescribeParam() • 1：对所有parameter返回SQL_VARCHAR
ENABLE_SQLBINDPARAMETER_CONSISTENCY_CHECK	决定是否检查SQLBindParameter()的ColumnSize和DecimalDigits • 0：不检查ColumnSize和DecimalDigits • 1：检查ColumnSize和DecimalDigits
USE_TARGETTYPE	设置以通信接收column类型时同时接收的类型信息 • 0：仅接收column类型 • 1：接收column类型与column名称 • 2：接收column类型与column的所有信息
LOCATOR_DSN	指定Location信息的Data Source Name（DSN）

属性	说明
LOCATOR_SERVICE	从Service hintglocator获取访问信息
LOCALITY_AWARE_TRANSACTION	是否使用GLOBAL CONNECTION • 0：不使用GLOBAL CONNECTION • 1：使用GLOBAL CONNECTION
LOCALITY_GROUP_POLICY	使用GLOBAL CONNECTION时没有可选择的群组或可以选择两个以上的群组时设置群组选择方法 • 0：任意选择 • 1：按照顺序选择LOCALITY_GROUP_PATH设置中的群组无法使用LOCALITY_GROUP_PATH的所有群组时选择任意群组 • 2：按照顺序选择每次按照驱动中连接的群组顺序选择
LOCALITY_GROUP_PATH	使用GLOBAL CONNECTION时可选择的群组不是1个时指定选择的群组的目录各群组以逗号（）区分 例：G1G2G3

属性	说明
LOCALITY_MEMBER_POLICY	决定使用GLOBAL CONNECTION时所选的群组中的成员的选择方法 • 0: DML: MASTER / SELECT: MASTER • 1: DML: ANY / SELECT : ANY • 2: DML: MASTER / SELECT: ANY • 3: DML: MASTER / SELECT: SLAVE • 4: 按照LOCALITY_MEMBER_PATH设置中的成员的顺序进行选择无法使用LOCALITY_MEMBER_PATH中的所有成员时选择所选群组的MASTER
LOCALITY_MEMBER_PATH	指定在使用GLOBAL CONNECTION时所选的群组中要使用的成员的目录各成员以逗号（）区分 例：G1N1, G2N1, G3N1, G1N2, G2N2, G3N2
DB_HOME	设置数据库的home目录默认值使用$SUNDB_HOME的环境变量
PACKET_COMPRESSION_THRESHOLD	发送至服务器的通信数据的大小大于PACKET_COMPRESSION_THRESHOLD时压缩通信数据设定值的范围为32 ~ 2113929216

属性	说明
USE_GLOBAL_SESSION	是否使用GLOBAL SESSION • 0：不使用GLOBAL SESSION • 1：使用GLOBAL SESSION
LOGIN_TIMEOUT	等待完成login请求的时间（秒）
TRACE	是否在ODBC API使用trace • 0：不使用trace • 1：使用trace
TRACEFILE	Trace文件名输入在相对路径时执行程序的当前目录成为标准默认值为'odbc_trace.log'
TRACE_POLICY	Trace策略 默认值为DEFAULT • DEFAULT：同时记录函数参数和结果 • ERROR：函数失败时记录日志

属性	说明
INCLUDE_SYNONYMS	决定是否在SQLGetColumns()中包含synonym个体. • 0: 不包含synonym个体 • 1: 包含synonym个体
DOT_NET_FOR_ODBC	决定是否将ODBC用于.NET Framework用途 • 0: 不改变用途 • 1: SQLGetDescField()和SQLColAttribute()中将SQL_DESC_BASE_COLUMN_NAME, SQL_DESC_NAME属性替换为SQL_DESC_LABEL

Table 2-3 数据源配置部分的关键字

关键字	说明
FILE	Location file name
HOST	glocator ip address
PORT	glocator port number
CONNECTION_TIMEOUT	Connection timeout with glocator (second)

关键字	说明
ALTERNATE_LOCATORS	未收到glocator的响应时使用ALTERNATE_LOCATORS获取访问信息

Table 2-4 Location

> Note:
>
> * LOCATOR_DSN中已设置FILE与HOSTPORT属性时优先适用FILE属性FILE相关详细内容参考**Location File**
>
> * 通过LOCATOR_SERVICE属性可访问LOCATOR_SERVICE所属的服务器所访问的服务器以外的服务器为ALTERNATE_SERVERS
> + 未设置FAILOVER_TYPE则FAILOVER_TYPE为session
> + 未设置FAILOVER_GRANULARITY则FAILOVER_GRANULARITY为1
> + 详细内容参考**glocator**与 **gloctl**

SUNDB的DSN配置如下

```
[SUNDB]
Driver = SUNDB ODBC Driver
PROTOCOL = TCP
CS_MODE = SHARED
HOST = 192.168.0.10
PORT = 22581
CHARSET = UTF8
TCP_NODELAY = 1
```

```
ALTERNATE_SERVERS =

(HOST=192.168.0.11:PORT=22581,HOST=192.168.0.12:PORT=22581)

CONNECTION_RETRY_COUNT = 3

CONNECTION_RETRY_DELAY = 1

FAILOVER_TYPE = SESSION

FAILOVER_GRANULARITY = 0

FAILOVER_ROUTING_POLICY = 0

DATE_FORMAT = YYYY-MM-DD

TIME_FORMAT = HH24:MI:SS.FF6

TIME_WITH_TIME_ZONE_FORMAT = HH24:MI:SS.FF6 TZH:TZM

TIMESTAMP_FORMAT = YYYY-MM-DD HH24:MI:SS.FF6

TIMESTAMP_WITH_TIME_ZONE_FORMAT = YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM

CHAR_LENGTH_UNITS = CHARACTERS

ENABLE_SQLDESCRIBEPARAM = 1

ENABLE_SQLBINDPARAMETER_CONSISTENCY_CHECK = 1


USE_TARGETTYPE = 0

INCLUDE_SYNONYMS = 0


PACKET_COMPRESSION_THRESHOLD = 2113929216


LOCALITY_AWARE_TRANSACTION = 0

LOCALITY_GROUP_POLICY = 0

LOCALITY_GROUP_PATH = G1,G2,G3

LOCALITY_MEMBER_POLICY = 0
```

```
LOCALITY_MEMBER_PATH = G1N1,G2N1,G3N1,G1N2,G2N2,G3N2


USE_GLOBAL_SESSION = 0


LOGIN_TIMEOUT = 0


LOCATOR_DSN = LOCATOR

LOCATOR_SERVICE = S1


TRACE = 1

TRACEFILE = /home/test/log/mytrace.log


DOT_NET_FOR_ODBC = 0


[LOCATOR]

FILE = /home/test/.location.ini

HOST = 127.0.0.1

PORT = 42581

ALTERNATE_LOCATORS=(HOST=127.0.0.1:PORT=42582,HOST=127.0.0.1:PORT=42583)
```

Windows中的DSN设置

Windows中可通过ODBC数据源管理器添加或设置DSN

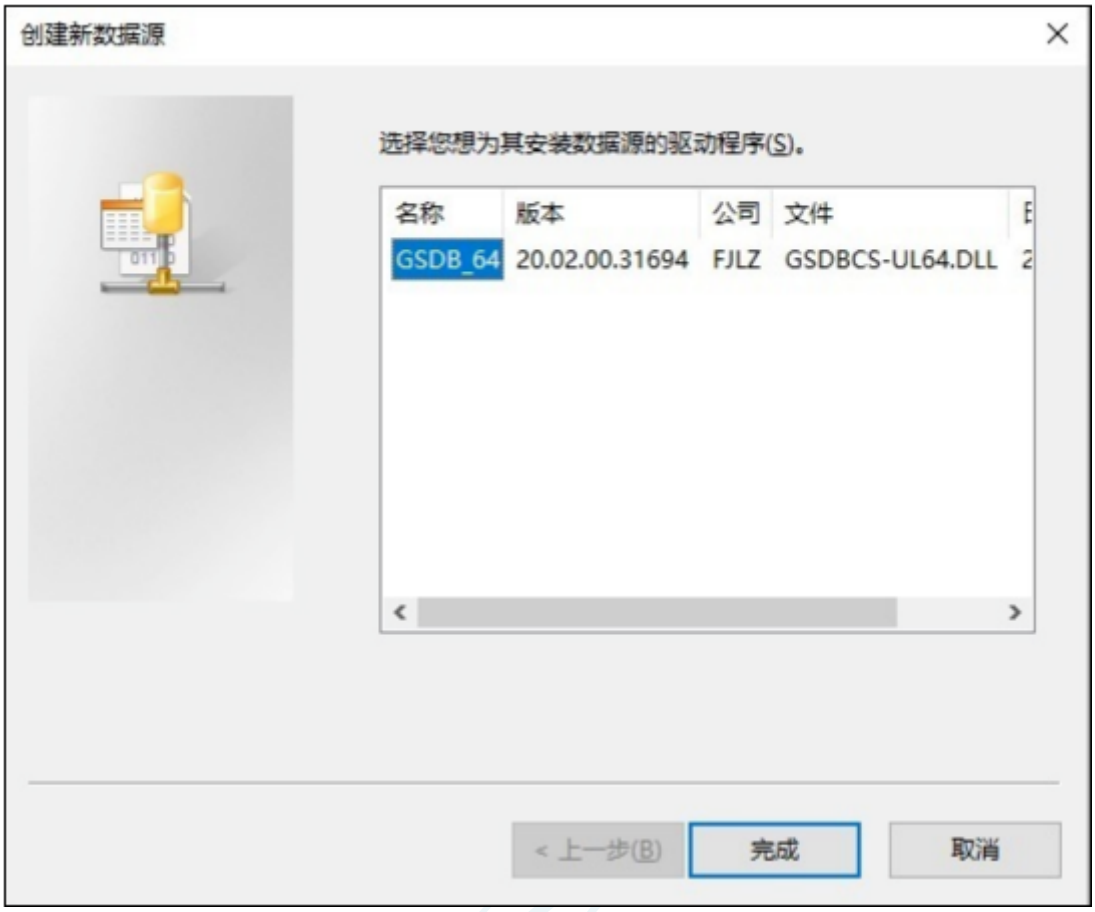


Figure 2-3 Creating new data source

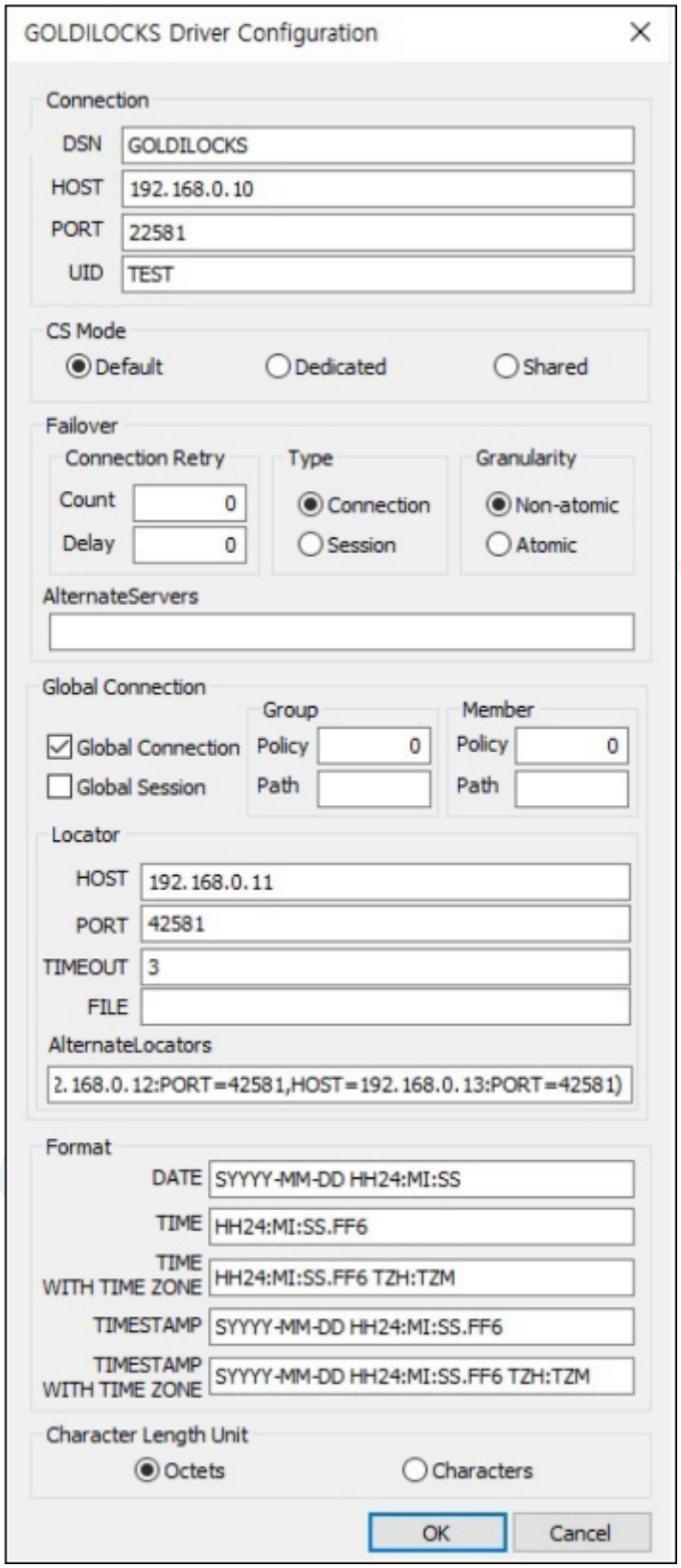


Figure 2-4 ODBC driver configuration

各项详细说明如下

关键字	说明
DSN	数据源名称
HOST	HOST IP地址或名称
PORT	连接端口号
UID	用户ID
CS_MODE	设置以dedicated模式连接或以shared模式连接 未设置时default模式取决于listener的 configuration(DEFAULT_CS_MODE)
ALTERNATE_SERVERS	发生failover时尝试连接的服务器列表以逗号（）区分 不使用failover功能时ALTERNATE_SERVERS设置为空白
CONNECTION_RETRY_COUNT	连接失败时尝试重连服务器的次数
CONNECTION_RETRY_DELAY	连接失败时尝试重连的服务器时间间隔（单位：秒）
FAILOVER_TYPE	• CONNECTION：连接失败时连接至ALTERNATE_SERVERS • SESSION：连接失败或语句执行中连接中断时连接至ALTERNATE_SERVERS后恢复语句断开连接时如没有进行中的事务failover后执行进行中的statement

关键字	说明
FAILOVER_GRANULARITY	• Non-atomic：failover过程中即使报错也继续执行failover • Atomic：failover过程中报错则failover失败
DATE_FORMAT	DATE类型字符串
TIME_FORMAT	TIME类型字符串
TIME_WITH_TIME_ZONE_FORMAT	TIME WITH TIME ZONE类型字符串
TIMESTAMP_FORMAT	TIMESTAMP类型字符串
TIMESTAMP_WITH_TIME_ZONE_FORMAT	TIMESTAMP WITH TIME ZONE类型字符串
CHAR_LENGTH_UNITS	SQLBindParameter()中ParameterType为SQL_CHAR时的ColumnSize单位 • BYTE, OCTETS：以byte为单位 • CHAR, CHARACTERS：以字符为单位
LOCALITY_AWARE_TRANSACTION	GLOBAL CONNECTION 使用与否 • 0：不使用GLOBAL CONNECTION • 1：使用GLOBAL CONNECTION

关键字	说明
USE_GLOBAL_SESSION	GLOBAL SESSION使用与否 • 0：不使用GLOBAL SESSION • 1：使用GLOBAL SESSION
LOCALITY_GROUP_POLICY	设置可使用GLOBAL CONNECTION时没有可选择的group或有两个以上可选择的group时选择哪个group • 0：任意选择 • 1：依次选择LOCALITY_GROUP_PATH设置中的group无法使用所有LOCALITY_GROUP_PATH 中的group时选择任意group • 2：依次选择按照每次连接驱动的group顺序选择
LOCALITY_GROUP_PATH	使用GLOBAL CONNECTION时可选择的group不是一个时指定选择的group的目录各个group以逗号（）区分 例：G1G2G3

关键字	说明
LOCALITY_MEMBER_POLICY	使用GLOBAL CONNECTION时决定选择被选择地group中的成员的方法 • 0：DML: MASTER / SELECT: MASTER • 1: DML: ANY / SELECT: ANY • 2: DML: MASTER / SELECT: ANY • 3: DML: MASTER / SELECT: SLAVE • 4：依次选择LOCALITY_MEMBER_PATH设置中的成员无法使用所有LOCALITY_MEMBER_PATH中的成员时选择被选择的group的MASTER
LOCALITY_MEMBER_PATH	使用GLOBAL CONNECTION时在所选的group中指定要使用的成员的目录各个成员以逗号（）区分 例：G1N1,G2N1,G3N1,G1N2,G2N2,G3N2
LOCATOR_HOST	glocator ip address
LOCATOR_PORT	glocator port number
LOCATOR_CONNECTION_TIMEOUT	Connection timeout with glocator (second)
ALTERNATE_LOCATORS	无法从glocator获得响应时使用ALTERNATE_LOCATORS获取访问信息

关键字	说明
TRACE	是否在ODBC API使用trace • 0：不使用trace • 1：使用trace
TRACEFILE	Trace文件名输入到相对路径时执行程序的当前目录成为 标准默认值为'odbc_trace.log'
TRACE_POLICY	Trace策略 默认值为DEFAULT • DEFAULT：同时记录函数参数和结果 • ERROR：函数失败时记录日志
DOT_NET_FOR_ODBC	决定是否将ODBC用于.NET Framework用途 • 0: 不改变用途 • 1: SQLGetDescField()和SQLColAttribute()中将 SQL_DESC_BASE_COLUMN_NAME, SQL_DESC_NAME属性替换为SQL_DESC_LABEL

Table 2-5 DSN配置关键字

2.3 GLOBAL CONNECTION

集群环境支持应用程序可以选择符合处理查询的节点后执行的GLOBAL CONNECTION功能

设置

仅在PROTOCOL为TCP时才可使用GLOBAL CONNECTION此时需要在设置LOCALITY_AWARE_TRANSACTION属性值的同时设置LOCATOR文件或LOCATOR服务器使用global session时需要将USE_GLOBAL_SESSION属性值设置为1

- 使用DSN时的.odbc.ini

```
[SUNDB]
PROTOCOL = TCP
HOST = 192.168.0.1
PORT = 22581
UID = TEST
PWD = test
LOCALITY_AWARE_TRANSACTION = 1
LOCATOR_DSN = LOCATOR

[LOCATOR]
FILE = /home/sundb/.location.ini
```

- 使用连接字符串时

```
SQLDriverConnect( dbc,
                  NULL,


(SQLCHAR*)"PROTOCOL=TCP;HOST=192.168.0.1;PORT=22581;UID=TEST;PWD=test;LOCA
LITY_AWARE_TRANSACTION=1;LOCATOR_HOST=192.168.0.2;LOCATOR_PORT=42581",
                  SQL_NTS,
                  NULL,
                  0,
                  NULL,
                  SQL_DRIVER_NOPROMPT );
```

GLOBAL CONNECTION的处理过程

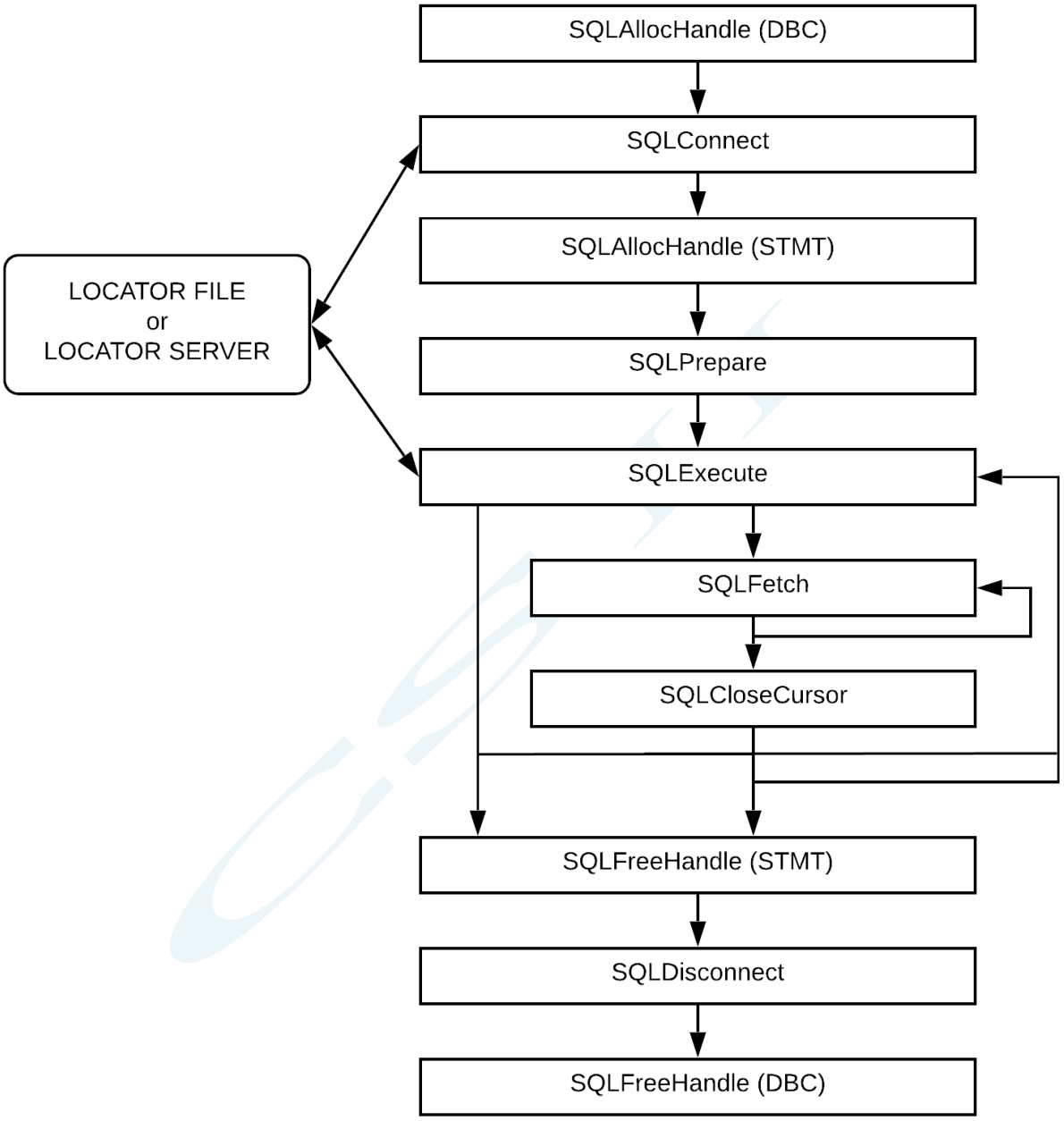


Figure 2-5 Basic steps of applying GLOBAL CONNECTION

1. SQLAllocHandle (DBC)

   分配connection handle

2. SQLConnect

   连接用户输入服务器信息的服务器并获取集群系统的信息后通过LOCATOR文件或LOCATOR服务器建立集群系统信息并连接集群系统的所有节点

3. SQLAllocHandle(STMT)

   向连接的所有节点分配各个statement

4. SQLPrepare

   在连接的所有节点准备执行SQL

5. SQLExecute

   如未建立集群系统信息应用程序可通过LOCATOR文件或LOCATOR服务器建立集群系统信息并连接集群系统的所有节点

   在集群添加节点后访问新的节点时在该节点同样生成其他节点的所有statement并准备执行SQL

如已建立sharding key信息应用程序使用sharding key信息选择合适的节点后执行查询

如未建立sharding key信息应用程序从任意服务器建立该SQL的sharding key信息后选择合适的节点并执行查询

所选节点发生故障时除该节点外再重新选择合适的节点后执行查询

SQLExecute后sharding信息发生变更则删除已建立的sharding key信息

SQLExecute后添加/删除集群节点等集群系统信息发生变更则删除已建立的集群系统信息

6. SQLFetch

   从执行SQL的节点获取数据

7. SQLCloseCursor

   在执行SQL的节点关闭游标

8. SQLFreeHandle（STMT)

   在所有已连接的节点上解除statement

9. SQLDisconnect

   解除与所有节点的连接

10. SQLFreeHandle（DBC）

    解除connection handle

GLOBAL CONNECTION异常处理

使用GLOBAL CONNECTION时运行中所选节点发生故障时根据事务发生及SELECT进行与否如下进行操作

- 没有事务的情况

没有事务的情况下所选节点发生故障时在ODBC内部选择其他节点并执行对应查询虽然所选节点发生了故障但由其他节点正常执行因此不向用户报错

- 有事务或SELECT进行时

发生事务或SQLFetch()进行时所选节点发生故障时ODBC无法再进行当前操作因此返回19068(Retry the transactional operations)错误发生19068错误时用户需要重新执行对应事务或SELECT

```
if( !SQL_SUCCEEDED(SQLPrepare( sStmt,
                               (SQLCHAR*)"INSERT INTO T1 VALUES ( ? )",
                               SQL_NTS )) )
{
    goto stmt_error;
```

```
}


trans_retry:


if( !SQL_SUCCEEDED(SQLExecute( sStmt )) )
{
    SQLGetDiagRec( SQL_HANDLE_STMT,
                   sStmt,
                   1,
                   sSQLState,
                   &sNativeError,
                   sMessageText,
                   sizeof(sMessageText),
                   &sTextLength );


    if( sNativeError == 19068 )
    {
        goto trans_retry;
    }


    goto stmt_error;
}



if( !SQL_SUCCEEDED(SQLPrepare( sStmt,
```

```
                            (SQLCHAR*)"SELECT * FROM T1 WHERE C1 = ?",
                            SQL_NTS )) )
{
    goto stmt_error;
}

trans_begin:

sReturn = SQLExecute( sStmt );

if( sRetrun == SQL_ERROR )
{
    SQLGetDiagRec( SQL_HANDLE_STMT,
                   sStmt,
                   1,
                   sSQLState,
                   &sNativeError,
                   sMessageText,
                   sizeof(sMessageText),
                   &sTextLength );

    if( sNativeError == 19068 )
    {
        goto trans_retry;
    }
```

```
    goto stmt_error;
}

while( 1 )
{
    sReturn = SQLFetch( sStmt );

    if( sReturn == SQL_NO_DATA )
    {
        SQLCloseCursor( sStmt );
        break;
    }
    else if( sReturn == SQL_ERROR )
    {
        SQLGetDiagRec( SQL_HANDLE_STMT,
                       sStmt,
                       1,
                       sSQLState,
                       &sNativeError,
                       sMessageText,
                       sizeof(sMessageText),
                       &sTextLength );

        if( sNativeError == 19068 )
```

```
        {
            goto trans_retry;
        }

        goto stmt_error;
    }

    ...
}
```

- COMMIT或ROLLBACK事务时

COMMIT事务时所选节点发生故障时ODBC通过其他节点查看事务是否在所选节点发生故障之前COMMIT如果所选节点发生了故障但事务正常COMMIT时不返回错误而且在事务未COMMIT的情况下所选节点发生故障时返回19068(Retry the transactional operations)错误发生19068错误时用户要重新执行该事务

ROLLBACK事务时如果所选节点发生故障ODBC不报错因为由于节点故障该事务已经被ROLLBACK

```
if( !SQL_SUCCEEDED(SQLSetConnectAttr( sDbc,
                                      SQL_AUTOCOMMIT,
                                      (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
                                      0 ) ))
{
    goto dbc_error;
```

```
}


if( !SQL_SUCCEEDED(SQLPrepare( sStmt,
                               (SQLCHAR*)"INSERT INTO T1 VALUES ( ? )",
                               SQL_NTS )) )
{
    goto stmt_error;
}


trans_retry:


if( !SQL_SUCCEEDED(SQLExecute( sStmt )) )
{
    SQLGetDiagRec( SQL_HANDLE_STMT,
                   sStmt,
                   1,
                   sSQLState,
                   &sNativeError,
                   sMessageText,
                   sizeof(sMessageText),
                   &sTextLength );


    if( sNativeError == 19068 )
    {
        goto trans_retry;
```

```
    }

    goto stmt_error;
}

if( !SQL_SUCCEEDED(SQLEndTran( SQL_HANDLE_DBC,
                               sDbc,
                               SQL_COMMIT)) )
{
    SQLGetDiagRec( SQL_HANDLE_DBC,
                   sDbc,
                   1,
                   sSQLState,
                   &sNativeError,
                   sMessageText,
                   sizeof(sMessageText),
                   &sTextLength );

    if( sNativeError == 19068 )
    {
        goto trans_retry;
    }

    goto stmt_error;
}
```

GLOBAL CONNECTION 约束事项

- 为了选择符合查询的节点需要使用SQLPrepare()和SQLExecute()

使用GLOBAL CONNECTION查找符合查询的节点时需要使用SQLPrepare()和SQLExecute()SQLExecDirect()中没有查找符合查询的节点的信息因此选择节点取决于LOCALITY_GROUP_POLICY和LOCALITY_MEMBER_POLICY 属性

- COMMIT或ROLLBACK事务时要使用SQLEndTran()

使用GLOBAL CONNECTION时通过SQL语句执行COMMIT或ROLLBACK则无法检测到事务状态变化COMMIT或ROLLBACK事务时必须使用SQLEndTran()

- Global session 不支持SQL语句中的Data Definition Language (DDL)

2.4 目录函数

所有数据库均拥有数据存储方式的结构体例如一个简单的sales order数据库有如下结构体此时ID column用于连接各表

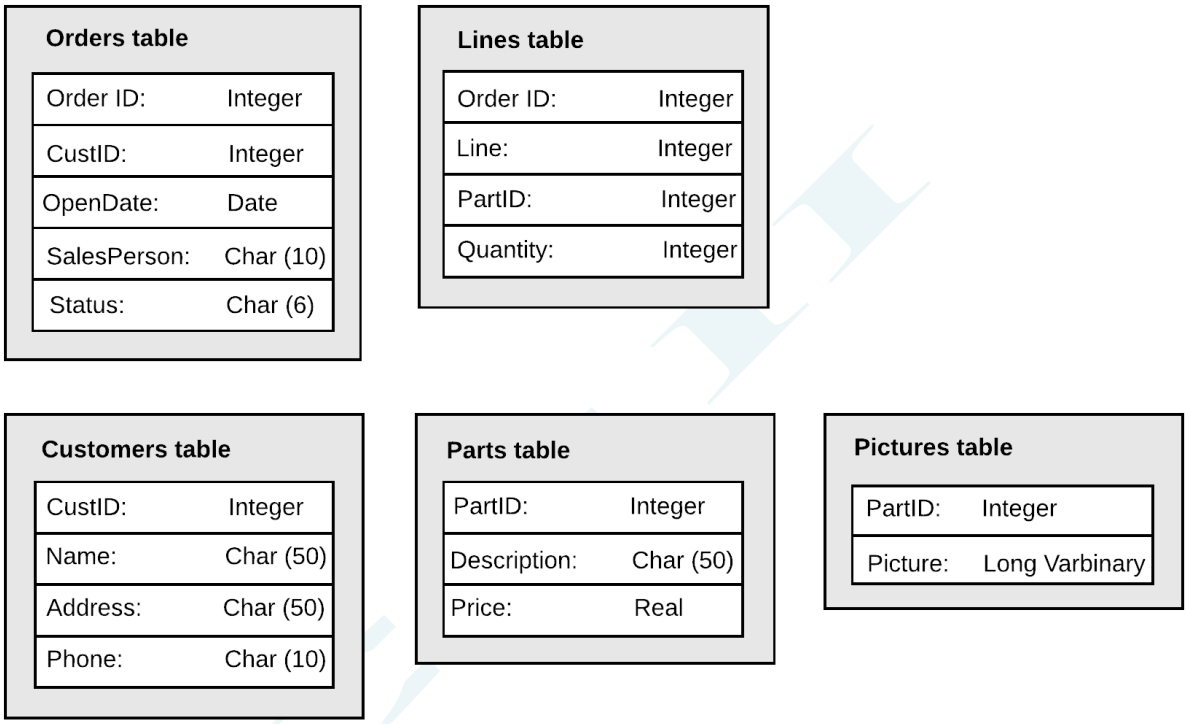


Figure 2-6 Sales orders

此结构体与权限等其他信息一起存储于称为数据库目录的系统表的set中即数据字典

应用程序可通过调用目录函数发现此结构体目录函数在结果集中返回信息通常通过对目录的表执行SELECT 命令实现

目录数据的使用

应用程序以多种方法使用目录数据,以下为几种通用的使用方法

执行时组成SQL命令语

订单输入系统等垂直型应用程序通常拥有写死的SQL语句应用程序使用的表与column是提前固定的应用程序访问这些表例如订单输入应用程序中为了在系统添加新的订单通常有一个媒介变量化的命令

通过ODBC回收数据的电子表格程序等一般应用程序是基于用户输入内容在执行时间点组成SQL命令此类应用可向用户请求使用表与column的格式此时如果在应用程序显示用户所选的表或column目录将会更加便于用户使用为了建立这样的目录应用程序调用SQLTables及SQLColumns目录函数

开发过程中组成SQL命令语

通常的应用程序开发环境在开发程序的过程中允许开发人员生成数据库query之后把这些query写死在应用程序中

在这种环境下开发人员可以使用SQLTables及SQLColumns创建目录这种环境使用SQLPrimaryKeys和SQLForeignKeys自动查询并显示所选的表之间关系也可以通过SQLStatistics查找并强调索引字段使开发人员更加有效地创建query

游标构成

提供滚动游标的应用程序驱动程序中间件使用SQLSpecialColumns查找唯一能识别行的列应用程序可以对回收的各行建立包含这些列值的keyset 应用程序会使用这些值向后滚动行并回收行的

最新数据

ODBC中的目录函数

ODBC包含以下目录函数

函数	说明
SQLTables	返回数据源中的catalogschematable或table形式的目录
SQLColumns	返回一个以上的表的列目录
SQLStatistics	返回一个表的统计目录同时返回连接该表的索引目录
SQLSpecialColumns	返回一个表中唯一识别行的列目录也返回其表中的列目录并自动更新
SQLPrimaryKeys	返回一个表的组成primary key的列目录
SQLForeignKeys	返回一个表的foreign key目录或参照此表的其他表的foreign key目录
SQLTablePrivileges	返回一个以上与表相关的权限的目录
SQLColumnPrivileges	返回一个表的一个以上的与列相关权限的目录
SQLProcedures	驱动程序不支持
SQLProcedureColumns	驱动程序不支持
SQLGetTypeInfo	返回数据源支持的SQL数据类型目录这些数据类型一般用于CREATE TABLE及ALTER TABLE命令

Table 2-6 ODBC中的目录函数

目录函数的数据返回

目录函数像结果集一样返回数据该结果集并非不同于其他任何结果集该结果集通常写死在驱动中或由存储于数据源procedure中的媒介变量化的SELECT命令语等提前定义好的命令生成

各目录函数结果集的说明在对应函数的参考项目中结果集可在最后选择的列包含包含除选定的列之外在驱动程序定义的列这种列相关说明可参考驱动程序用户指南

应用程序以结果集的最后为准绑定驱动程序定义的列并在驱动程序定义的列数计算为比所要求的列数之后产生的列数小的最后一个列数这可以减少在后续的ODBC版本或驱动程序中增加新的列时需要变更应用程序的麻烦为了使用此SCHEMA需要在原有的驱动定义的列之前添加新的驱动定义的列防止列的数字以结果集的最后为准发生变更

即使包含特殊文字也不会引用返回在结果集中的标识符
假设（驱动程序中定义并通过SQLGetInfo返回的）标识符引用文字为双引号（"）的Accounts Payable表包含名称为Customer Name的列则在SQLColumns对此列返回的行中TABLE_NAME列的值为Accounts Payable而不是"Accounts Payable"COLUMN_NAME列的值为Customer Name而不是"Customer Name"
应用程序通过如下命令语回收Accounts Payable表中的客户名

```
SELECT "Customer Name" FROM "Accounts Payable"
```

目录函数以基于用户名与密码的连接中的SQL-like认证模式为基础并仅向有权限的用户返回数据不适合此模式的各文件的密码保护由驱动程序定义

无法更新大部分由目录函数返回的结果集即使应用程序更新此结果集中的数据也无法变更数据库的结构体

目录函数的参数

pattern值参数

与SQLTables的TableName参数类似目录函数中的部分参数也接受检索pattern如果SQL_ATTR_METADATA_ID命令属性设置为SQL_FALSE则此参数可以输入查询pattern如果该属性设置为SQL_TRUE则无法输入检索pattern

检索pattern字符由如下特征

- 下划线(_)代表任意一个字符
- 百分号 (%)代表0个以上的任意字符
- 转义字符由驱动程序定义并用于包含百分号下划线转义字符转换字符出现在非特殊字符前时转义字符没有特殊意义出现在特殊字符前时转义字符为特殊字符例如"\a""视为"\" d与"a"两个文字但"\%"只是无特殊意义的一个"%"

SQLGetInfo中使用SQL_SEARCH_PATTERN_ESCAPE选项回收转义字符为了使可输入检索pattern的参数引用其原有字符需要位于下划线百分号或转义字符之前

检索pattern	说明
%A%	包含A的所有标识符
ABC_	以ABC开始的所有4个字符
ABC_	假设转义字符为反斜杠(\)标识符为ABC_
\\%	假设转义字符为反斜杠(\)标识符以反斜杠(\)开始

Caution:

对可以使用检索pattern的参数需要格外注意使用转义检索pattern字符特别是经常用作标识符的下划线(_)

应用程序经常犯一个错误即将从一个目录函数回收的值直接发送至其他目录函数的检索pattern参数

例如应用程序从SQLTables 的结果集回收表名为MY_TABLE的表并为了查询此表的列目录而把MY_TABLE发送至SQLColumns时应用程序将返回MY_TABLEMY1TABLEMY2TABLE等对应检索pattern MY_TABLE的所有表的列而不是MY_TABLE的列

Note:

ODBC 2.x驱动程序不支持在SQLTables的CatalogName参数中使用检索patternODBC 3.x驱动程序把SQL_ATTR_ODBC_VERSION环境属性设置为SQL_OV_ODBC3时此参数可输入检索pattern如果此属性设置为SQL_OV_ODBC2则不可输入检索pattern

在检索pattern参数中使用NULL指针（pointer）意味着不强制检索其参数NULL指针与检索pattern%（任何字符）意义相同但长度为0的检索pattern与空字符("")匹配

2.5 非标准数据类型

本章将说明ODBC标准中未定义的SUNDB数据类型

C数据类型

C type identifier	ODBC C typedef	C type
SQL_C_BOOLEAN	SQLBOOLEAN	unsigned char
SQL_C_LONGVARBINARY	SQL_LONG_VARIABLE_LENGTH_STRUCT	[a]
SQL_C_LONGVARCHAR	SQL_LONG_VARIABLE_LENGTH_STRUCT	[a]
SQL_C_REFCURSOR	SQLHSTMT	void *
SQL_C_TYPE_TIME_WITH_TIMEZONE	SQL_TIME_WITH_TIMEZONE_STRUCT	[b]
SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE	SQL_TIMESTAMP_WITH_TIMEZONE_STRUCT	[c]
SQL_C_WLONGVARCHAR	SQL_WLONG_VARIABLE_LENGTH_STRUCT	[d]

Table 2-7 SUNDB C数据类型

- [a]

```
typedef struct tagLONG_VARIABLE_LENGTH_STRUCT
{
```

```
    SQLBIGINT   len;
    SQLCHAR   * arr;
} LONG_VARIABLE_LENGTH_STRUCT;
```

- [b]

```
typedef struct tagTIME_WITH_TIMEZONE_STRUCT
{
   SQLUSMALLINT hour;
   SQLUSMALLINT minute;
   SQLUSMALLINT second;
   SQLUINTEGER  fraction;
   SQLSMALLINT  timezone_hour;
   SQLSMALLINT  timezone_minute;
} TIME_WITH_TIMEZONE_STRUCT;
```

- [c]

```
typedef struct tagTIMESTAMP_WITH_TIMEZONE_STRUCT
{
   SQLSMALLINT  year;
   SQLUSMALLINT month;
   SQLUSMALLINT day;
   SQLUSMALLINT hour;
   SQLUSMALLINT minute;
   SQLUSMALLINT second;
```

```
    SQLUINTEGER  fraction;
    SQLSMALLINT  timezone_hour;
    SQLSMALLINT  timezone_minute;
} TIMESTAMP_WITH_TIMEZONE_STRUCT;
```

- [d]

```
typedef struct tagWLONG_VARIABLE_LENGTH_STRUCT
{
    SQLBIGINT   len;
    SQLWCHAR  * arr;
} WLONG_VARIABLE_LENGTH_STRUCT;
```

SQL_C_BOOLEAN

用于描述boolean值

- SQL_TRUE
- SQL_FALSE

```
SQLBOOLEAN boolean   = SQL_TRUE;
SQLLEN     indicator = 0;


SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_BOOLEAN,
```

```
                SQL_BOOLEAN,
                0,
                0,
                &boolean,
                0,
                &indicator );
```

SQL_C_LONGVARBINARY

它具有SQLCHAR缓冲区的大小和SQLCHAR缓冲区的指针

SQLCHAR缓冲区具有二进制数据

二进制数据的实际长度通过indicator传输

```
SQL_LONG_VARIABLE_LENGTH_STRUCT   longvarbinary;
SQLLEN                            indicator;
SQLCHAR                          * buffer;


buffer = (SQLCHAR*)malloc(128);


longvarbinary.arr = buffer;
longvarbinary.len = 128;


buffer[0] = 0x1;
buffer[1] = 0x2;
buffer[2] = 0x3;
```

```
indicator = 3;


SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_LONGVARBINARY,
                  SQL_VARBINARY,
                  128,
                  0,
                  &longvarbinary,
                  0,
                  &indicator );
```

SQL_C_LONGVARCHAR

它具有SQLCHAR缓冲区的大小和SQLCHAR缓冲区的指针

SQLCHAR缓冲区具有字符串数据

字符串数据的实际长度通过indicator传输

```
SQL_LONG_VARIABLE_LENGTH_STRUCT   longvarchar;
SQLLEN                            indicator;
SQLCHAR                         * buffer;


buffer = (SQLCHAR*)malloc(128);
```

```
longvarchar.arr = buffer;
longvarchar.len = 128;

buffer[0] = 'A';
buffer[1] = 'B';
buffer[2] = 'C';

indicator = 3;

SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_LONGVARCHAR,
                  SQL_VARCHAR,
                  128,
                  0,
                  &longvarchar,
                  0,
                  &indicator );
```

SQL_C_REFCURSOR

可以通过分配的SQLHSTMT使用REF CURSOR

```
SQLHSTMT stmt    = SQL_NULL_HSTMT;
```

```
SQLHSTMT refstmt = SQL_NULL_HSTMT;
SQLLEN   refind  = 0;
SQLCHAR  job[10];
SQLLEN   jobind  = 0;

SQLAllocHandle( SQL_HANDLE_STMT,
                dbc,
                &stmt );

SQLAllocHandle( SQL_HANDLE_STMT,
                dbc,
                &refstmt );

SQLPrepare( stmt,
            (SQLCHAR*)"CALL EMP_BY_JOB_REFCURSOR(?, ?)",
            SQL_NTS );

SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_CHAR,
                  SQL_VARCHAR,
                  10,
                  0,
                  job,
```

```
                sizeof(job),
                &jobind );

SQLBindParameter( stmt,
                2,
                SQL_PARAM_OUTPUT,
                SQL_C_REFCURSOR,
                SQL_REFCURSOR,
                0,
                0,
                refstmt,
                0,
                &refind );

strcpy( (char*)job, "SALESMAN" );
jobind = strlen((char*)job);

SQLExecute( stmt );

SQLFetch( refstmt );
```

SQL_C_TYPE_TIME_WITH_TIMEZONE

它可以描述有时区的时间

- hour

- 时
- minute
  - 分
- second
  - 秒
- fraction
  - 1/1,000,000,000秒
    - SUNDB仅可描述到1/1,000,000秒
    - 须将微秒转换为纳秒使用
- timezone_hour
  - 时区的时
- timezone_minute
  - 时区的分

```
SQL_TIME_WITH_TIMEZONE_STRUCT timetz;
SQLLEN                        indicator = 0;

timetz.hour           = 10;
timetz.minute         = 20;
timetz.second         = 30;
timetz.fraction       = 123456000;
timetz.timezone_hour   = 9;
timetz.timezone_minute = 0;

SQLBindParameter( stmt,
```

```
                1,
                SQL_PARAM_INPUT,
                SQL_C_TYPE_TIME_WITH_TIMEZONE,
                SQL_TYPE_TIME_WITH_TIMEZONE,
                0,
                6,
                &timetz,
                0,
                &indicator );
```

SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE

它可以描述有时区的时间戳

- year
  - 年
- month
  - 月
- day
  - 日
- hour
  - 时
- minute
  - 分
- second
  - 秒

- fraction
  - 1/1,000,000,000秒
    - SUNDB仅可描述到1/1,000,000秒
    - 须将微秒转换为纳秒使用
- timezone_hour
  - 时区的时
- timezone_minute
  - 时区的分

```
SQL_TIMESTAMP_WITH_TIMEZONE_STRUCT timestamptz;
SQLLEN                             indicator = 0;

timestamptz.year           = 2010;
timestamptz.month          = 9;
timestamptz.day            = 1;
timestamptz.hour           = 10;
timestamptz.minute         = 20;
timestamptz.second         = 30;
timestamptz.fraction       = 123456000;
timestamptz.timezone_hour   = 9;
timestamptz.timezone_minute = 0;

SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
```

```
                SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE,
                SQL_TYPE_TIMESTAMP_WITH_TIMEZONE,
                0,
                6,
                &timestamptz,
                0,
                &indicator );
```

SQL_C_WLONGVARCHAR

它具有SQLWCHAR缓冲区的大小和SQLWCHAR缓冲区的指针

SQLWCHAR缓冲区具有统一码字符串数据

统一码字符串数据的实际长度通过indicator传输

```
SQL_WLONG_VARIABLE_LENGTH_STRUCT  wlongvarchar;
SQLLEN                            indicator;
SQLWCHAR                         * buffer;


buffer = (SQLWCHAR*)malloc(128 * sizeof(SQLWCHAR));


wlongvarchar.arr = buffer;
wlongvarchar.len = 128;


buffer[0] = 'A';
buffer[1] = 'B';
```

```
buffer[2] = 'C';


indicator = 3 * sizeof(SQLWCHAR);


SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_WLONGVARCHAR,
                  SQL_VARCHAR,
                  128,
                  0,
                  &wlongvarchar,
                  0,
                  &indicator );
```

SQL数据类型

SQL type identifier	SUNDB data type	SUNDB type description
SQL_BOOLEAN	BOOLEAN	保存TRUE或FALSE值
SQL_REFCURSOR	REF CURSOR	PSM的cursor variable
SQL_TYPE_TIME_WITH_TIMEZONE	TIME(p) WITH TIME ZONE	包含时区的时间Precision p表示seconds precision

SQL type identifier	SUNDB data type	SUNDB type description
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	TIMESTAMP(p) WITH TIME ZONE	包含时区的时间戳Precision p 表示seconds precision
SQL_ROWID	ROWID	记录标识符

Table 2-8 SUNDB SQL数据类型

SQL_BOOLEAN

SUNDB的BOOLEAN类型

SQL_REFCURSOR

SUNDB PSM的REF CURSOR类型

SQL_TYPE_TIME_WITH_TIMEZONE

SUNDB的TIME(p) WITH TIME ZONE类型

fractional seconds precision的范围是0 ~ 6

SQL_TYPE_TIMESTAMP_WITH_TIMEZONE

SUNDB的TIMESTAMP(p) WITH TIME ZONE类型

fractional seconds precision的范围是0 ~ 6

SQL_ROWID

SUNDB的ROWID类型

```
SQLCHAR  rowid[24];
SQLLEN   indicator = 0;


strcpy( (char*)rowid, "AAAAAAAAYfCAACAAAADiAAA" );
indicator = strlen((char*)rowid);


SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_CHAR,
                  SQL_ROWID,
                  0,
                  0,
                  rowid,
                  0,
                  &indicator );
```

2.6 ODBC API References

SQLAllocConnect

兼容性

导入版本：ODBC 1.0

概要

ODBC 3.x中SQLAllocConnect函数被替代为SQLAllocHandle函数

详细内容参考**SQLAllocHandle** 函数

语句

```
SQLRETURN SQLAllocConnect(
    SQLHENV   EnvironmentHandle,
    SQLHDBC * ConnectionHandlePtr);
```

参数

EnvironmentHandle

【输入】环境句柄

ConnectionHandlePtr

【输出】新分配的连接句柄的指针

SQLAllocEnv

兼容性

导入版本：ODBC 1.0

概要

ODBC 3.x中SQLAllocEnv函数被替代为SQLAllocHandle函数

详细内容参考**SQLAllocHandle**函数

语句

```
SQLRETURN SQLAllocEnv(
    SQLHENV * EnvironmentHandlePtr);
```

参数

EnvironmentHandlePtr

【输出】将要新分配的环境句柄的指针

SQLAllocHandle

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 9

概要

SQLAllocHandle分配环境连接以及命令语句handle

语句

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

参数

HandleType

【输入】SQLAllocHandle分配的句柄类型应为

SQL_HANDLE_DBCSQL_HANDLE_ENVSQL_HANDLE_STMT中的一个

InputHandle

【输入】HandleType为SQL_HANDLE_ENV时为SQL_NULL_HANDLEHandleType为

SQL_HANDLE_DBC时应为环境句柄为SQL_HANDLE_STMT时应为连接句柄

OutputHandlePtr

【输出】新分配的句柄的指针

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR

检测

SQLSTATE	报错	说明
08003	Connection not open	未连接状态下HandleType参数为SQL_HANDLE_STMTSQL_HANDLE_DESC
HY000	General error	没有特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY009	Invalid use of null pointer	OutputHandlePtr参数为null指针
HY010	Function sequence error	HandleType 参数为SQL_HANDLE_DBC并未调用设置SQL_ODBC_VERSION属性的SQLSetEnvAttr

SQLSTATE	报错	说明
HY014	Limit on the number of handles exceeded	限制分配句柄数量
HY092	Invalid attribute/option identifier	HandleType参数不是 SQL_HANDLE_ENVSQL_HANDLE_DBCSQL_HANDLE_STMTSQL_HANDLE_DESC
IM001	Driver does not support this function	HandleType参数为SQL_HANDLE_DESC

说明

SQLAllocHandle用于分配环境连接命令语句描述符的句柄通过*OutputHandlePtr使用SQLAllocHandle时驱动程序将覆盖对应句柄的信息驱动程序管理器无法查看*OutputHandlePtr中的句柄是否已使用也无法查看被覆盖之前的信息

分配环境句柄

环境句柄提供连接句柄是否有效是否激活等全局信息

如要请求环境句柄应用程序调用HandleType为SQL_HANDLE_ENVInputHandle为SQL_NULL_HANDLE的SQLAllocHandle驱动程序分配存储环境信息的内存并通过*OutputHandle参数返回分配的句柄应用程序向需要环境句柄参数的调用发送*OutputHandle值

分配环境句柄后应用程序调用SQLSetEnvAttr设置SQL_ATTR_ODBC_VERSION属性如果为了分配连接句柄而调用SQLAllocHandle时未设置此属性则返回SQLSTATE HY010(Function sequence error)

分配连接句柄

连接句柄提供是否为有效的命令语句是否为已连接的描述符句柄及事务是否为打开状态等信息

为了请求连接句柄应用程序调用HandleType为SQL_HANDLE_DBC的SQLAllocHandleInputHandle参数设置为调用SQLAllocHandle后返回的环境句柄驱动程序分配存储连接信息的内存并通过*OutputHandle参数返回分配的句柄应用程序向需要连接句柄参数的调用发送*OutputHandle值

如果调用分配连接句柄的SQLAllocHandle前未设置SQL_ATTR_ODBC_VERSION环境属性则返回SQLSTATE HY010(function sequence error)

分配命令语句柄

命令语句柄提供错误信息游标名称及SQL语句处理状态等信息

为了请求命令语句柄应用程序连接数据源后传送SQL语句之前调用SQLAllocHandle此时HandleType应设置为SQL_HANDLE_STMTInputHandle应设置为调用SQLAllocHandle返回的连接句柄驱动程序分配存储命令语信息的内存并通过*OutputHandle参数返回分配的句柄应用程序向需要命令语句柄参数的调用发送*OutputHandle值

分配命令语句柄后驱动程序自动分配四个描述符集合将这些描述符句柄分配于SQL_ATTR_APP_ROW_DESCSQL_ATTR_APP_PARAM_DESC,SQL_ATTR_IMP_ROW_DESC及SQL_ATTR_IMP_PARAM_DESC属性即默认描述符分配

SQLAllocStmt

兼容性

导入版本：ODBC 1.0

概要

ODBC 3.x中SQLAllocStmt 函数被替代为SQLAllocHandle 函数

详细内容参考**SQLAllocHandle**函数

语句

```
SQLRETURN SQLAllocStmt(
    SQLHDBC    ConnectionHandle,
    SQLHSTMT * StatementHandlePtr);
```

参数

ConnectionHandle

【输入】连接句柄

StatementHandlePtr

【输出】新分配的语句句柄的指针

SQLBindCol

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLBindCol将应用程序的数据缓冲绑定到结果集的列

语句

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

参数

StatementHandle

【输入】是命令语句柄

ColumnNumber

【输入】要绑定的结果集的列号列号从1开始递增

TargetType

【输入】*TargetValuePtr 缓冲区的C数据类型的标识符使用SQLFetchSQLFetchScroll及SQLSetPos检索数据时驱动程序将数据转换成该类型

TargetType参数为interval数据类型时默认值为interval leading precision (2)interval seconds precision (6)分别设置于ARD的SQL_DESC_DATETIME_INTERVAL_PRECISION及SQL_DESC_PRECISION字段TargetType参数为SQL_C_NUMERIC时默认值为precision (38)scale (0)分别设置于ARD的SQL_DESC_PRECISION及SQL_DESC_SCALE字段如果默认precision及scale不合适则应用程序需要调用SQLSetDescField或SQLSetDescRec并明确设置对应标识符字段

TargetValuePtr

【延迟的输入/输出】 绑定到列的数据缓冲区指针SQLFetch及SQLFetchScroll将数据返回到该缓冲区

如果TargetValuePtr为空指针则驱动程序解除列的数据缓冲区绑定应用程序可通过SQL_UNBIND选项调用SQLFreeStmt并解除所有列的绑定应用程序可以将TargetValuePtr参数设置为空指针并调用SQLBindCol而解除所有列的绑定但StrLen_or_IndPtr参数为有效值时仍然保持该列的长度/指示缓冲区的绑定

BufferLength

【输入】*TargetValuePtr 缓冲区的字节单位长度

驱动程序返回字符或二进制数据等可变长度数据时使用BufferLength防止溢出*TargetValuePtr的边界驱动程序向*TargetValuePtr返回字符数据时也要注意计算null终止字符因此*TargetValuePtr应包含空终止字符的空间否则驱动程序会截断数据

当驱动程序返回整数或日期结构体等固定长度数据时驱动程序会判断缓冲区足以保存数

据而忽略BufferLength因此应用程序需要向固定长度数据分配足够大的缓冲区否则驱动程序会溢出缓冲区

StrLen_or_IndPtr

【延迟的输入/输出】 指绑定到列的长度指示符缓冲区的指针SQLFetch及SQLFetchScroll向该缓冲区返回值

SQLFetch及SQLFetchScroll向长度/指示符缓冲区返回可返回的数据长度

SQL_NO_TOTALSQL_NULL_DATA

如果指示符缓冲区与长度缓冲区为不同的缓冲区时指示缓冲区只能返回SQL_NULL_DATA而长度缓冲可返回所有的值

StrLen_or_IndPtr为空指针时不使用长度/指示值当获取null数据时会报错

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR.

检测

SQLSTATE	报错	说明
07006	Restricted data type attribute violation	ColumnNumber 参数为0TargetType 参数并非SQL_C_BOOKMARK或SQL_C_VARBOOKMARK
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY003	Invalid application buffer type	TargetType 参数值为无效的数据类型
HY010	Function sequence error	调用SQLExecuteSQLExecDirect 后返回SQL_NEED_DATA并在发送所有data-at-execution变量之前调用了此函数
HY090	Invalid string or buffer length	BufferLength 参数值小于0
HYC00	Optional feature not implemented	驱动程序不支持对应column的SQL数据类型与TargetType参数值以及转换ColumnNumber 参数值为0驱动程序不支持书签

说明

SQLBindCol用于在应用程序的数据缓冲区及长度/指示缓冲区绑定结果集的column应用程序为了获取数据调用SQLFetch和SQLFetchScroll时驱动程序向指定缓冲区返回绑定的column数据应用程序可以不绑定column并通过调用SQLGetData查询数据

绑定列

为了绑定列应用程序调用SQLBindCol传送列号类型地址和数据缓冲区的长度及长度/指示缓冲区地址

应用程序在SQLBindCol时绑定缓冲区但驱动程序在调用SQLFetch及SQLFetchScroll时访问缓冲区所以延迟使用其缓冲区因此应用程序要保证SQLBindCol中设置的指针有效直到返回数据为止如果应用程序像解除缓冲区一样将指针无效化后再调用则结果会发生错误

绑定有效至被新的绑定替代或解除列绑定或解除命令语

解除列绑定

仅解除一个列时在应用程序中将要解除的column的编号设置为ColumnNumber再把TargetValue设置为空指针并调用SQLBindCol即可如果ColumnNumber为已解除绑定的列号时则SQLBindCol继续返回SQL_SUCCESS

如果要释放所有列的绑定在应用程序将Option设置为SQL_UNBIND后调用SQLFreeStmt即可也可以将ARD的SQL_DESC_COUNT字段设置为0后解除所有列的绑定

重新绑定列

应用程序为了变更绑定可以执行以下两个操作中的一个

- 调用SQLBindCol后对已绑定的column指定新的绑定
- 调用SQLBindCol后在指定缓冲区地址添加offset详细内容参考下文的**绑定Offsets**

绑定Offsets

绑定Offset为参照（TargetValuePtr和StrLen_or_IndPtr中指定的）数据及长度/指示缓冲区之前在地址中添加的值

使用绑定Offset基本上与调用SQLBindCol后重新绑定clumn的功能相同不同的是调用新的SQLBindCol时指定数据及长度/指示缓冲区的新地址而绑定offset不变更地址仅在原地址添加offset应用程序可以随时指定新的offset此offset总是添加到默认绑定地址如果offset设置为0或把命令语属性设置为空指针驱动程序则使用默认绑定地址

默认绑定地址与offset之和必须为有效地址但要添加的offset地址不一定必须有效

绑定数组

行集的大小（SQL_ATTR_ROW_ARRAY_SIZE命令的属性值）大于1时应用程序绑定缓冲区数组而不是一个缓冲区

应用程序可以通过以下两种方式绑定数组

- 对各column绑定数组各数据结构（数组）包含一个column的数据参考**列式绑定**
- 定义可包含整个行数据的结构体绑定该结构体的数组各数据结构包含一条行数据参考**行式绑定**

每个缓冲区数组至少要包含与行集大小相同的元素

列式绑定

列式绑定中应用程序将独立的数据和长度/指示数组绑定到各column

为了使用列式绑定应用程序首先将SQL_ATTR_ROW_BIND_TYPE命令属性设置为SQL_BIND_BY_COLUMN（默认值）应用程序对要绑定的各个column执行以下操作

1. 分配数据缓冲区数组
2. 分配长度/指示缓冲区数组

> Note:
>
> 使用列式绑定时如果应用程序直接写入描述符则独立的数组可用于长度和指示符数据

3. 与以下参数一起调用SQLBindCol

- TargetType 为数据缓冲区数组的各参数的形式
- TargetValuePtr 为数据缓冲区数组的地址
- BufferLength 为数据缓冲区数组的各元素的大小当数据为固定长度数据时忽略BufferLength参数
- StrLen_or_IndPtr 为长度/指示符数组的地址

行式绑定

在行式绑定应用程序定义包含要绑定的各列的数据及长度/指示符缓冲区的结构体

使用行式绑定时应用程序执行以下操作

1. 定义可包含一条行数据（包含数据和长度/指示符缓冲区）的结构体并分配该结构体的数组

> Note:
>
> 使用行式绑定时如果应用程序直接写入描述符则独立的字段可用于长度和指示数据

2. SQL_ATTR_ROW_BIND_TYPE属性设置包含一条行数据的结构体大小或要绑定的结果column的缓冲区实例大小对应长度应包含已绑定的所有column的空间与结构体或缓冲区间隙绑定的column地址增加至指定长度时应保证指向下一行的同一column的起始位置ANSI C中通过sizeof 运算符保证
3. 对要绑定的各列与以下参数一起调用SQLBindCol

- TargetType 为要绑定到column的数据缓冲区成员的类型
- TargetValuePtr 为第一个数组参数的数据缓冲区成员的地址
- BufferLength 为数据缓冲区成员的大小
- StrLen_or_IndPtr 为要绑定的长度/指示符成员的地址

缓冲区地址

缓冲区地址为数据或长度/指示缓冲区的实际地址驱动程序在写入缓冲区之前（与回收数据相同）计算缓冲区的地址用以下公式进行计算此公式使用TargetValuePtr和StrLen_or_IndPtr参数中指定的地址绑定offset行编号

```
Bound Address + Binding Offset + ((Row Number -1) x Element Size )
```

如下定义公式的变量

变量	说明
Bound address	数据缓冲区的地址指定于SQLBindCol的TargetValuePtr参数中 长度/指示符缓冲区的地址指定于SQLBindCol的StrLen_of_IndPtr参数中 绑定地址为0时计算的地址并非为0也不返回数据值
Binding offset	使用行式绑定时值与SQL_ATTR_ROW_BIND_OFFSET_PTR命令属性一起存储于指定的地址中 使用列式绑定或SQL_ATTR_ROW_BIND_OFFSET_PTR命令属性为空指针时绑定Offset为0
Row number	1-based的行集中的行编号 回收单一行时行编号默认为1

变量	说明
Element size	绑定数组中的元素大小 使用列式绑定时对长度/指示符缓冲区是sizeof(SQLLEN) 对数据缓冲区如果为可变长度数据类型则Element Size为SQLBindCol 的BufferLength 参数值如果为固定长度数据类型则Element Size为数据类型的大小 使用行式绑定时数据及长度/指示缓冲区均为SQL_ATTR_ROW_BIND_TYPE命令属性值

描述符和SQLBindCol

下面说明SQLBindCol如何与描述符互动

> Caution:
> 明确分配命令语相关的ARD并与其他命令语有关时对一个命令语调用SQLBindCol会影响其他命令语由于SQLBindCol修改描述符因此对描述符的修改适用于此描述符相关的所有命令语如果非故意行为应在调用SQLBindCol之前应用程序需要解除其他命令语与此描述符之间的关联性

参数映射

概念上SQLBindCol按照以下顺序执行

1. 通过调用SQLGetStmtAttr获取ARD句柄
2. 通过调用SQLGetDescField获取SQL_DESC_COUNT字段的描述符当ColumnNumber参数值超

过SQL_DESC_COUNT的值时通过调用SQLSetDescField把SQL_DESC_COUNT值增加到ColumnNumber值

3. 通过多次调用SQLSetDescField设置ARD的下一个field值

- 将SQL_DESC_TYPE和SQL_DESC_CONCISE_TYPE设置为TargetType值
    - 除TargetType为datetime或interval子类型的隐含标识符中的一个外将SQL_DESC_TYPE设置为SQL_DATETIME或SQL_INTERVALSQL_DESC_CONCISE_TYPE设置为隐含标识符SQL_DESC_DATETIME_INTERVAL_CODE设置为对应datetime或interval子码
- 对TargetType适当设置一个上的SQL_DESC_LENGTHSQL_DESC_PRECISIONSQL_DESC_SCALE和SQL_DESC_DATETIME_INTERVAL_PRECISION
- SQL_DESC_OCTET_LENGTH字段设置为BufferLength值
- SQL_DESC_DATA_PTR字段设置为TargetValue值
- SQL_DESC_INDICATOR_PTR字段设置为StrLen_or_Ind值
- SQL_DESC_OCTET_LENGTH_PTR字段设置为StrLen_or_Ind值

StrLen_or_Ind参数参照的变量用于指示符（Indicator）和长度信息如果取回时列值为NULL则存储SQL_NULL_DATA否则存储此变量的数据长度

如果输入空指针取回（fetch）时列值为NULL时无法返回SQL_NULL_DATA因此取回运算失败

SQLBindCol失败时不再定义在ARD中要设置的描述符字段的内容ARD的SQL_DESC_COUNT字段不会发生变化

COUNT字段的默认初始化

SQLBindCol仅在ColumnNumber增加SQL_DESC_COUNT值的情况下将SQL_DESC_COUNT设置为ColumnNumber值TargetValuePtr值为空指针ColumnNumber值等于SQL_DESC_COUNT（解除最大绑定column的绑定时）时SQL_DESC_COUNT设置为剩余的绑定column中最大的编号

SQL_DEFAULT 的注意事项

为了成功检索列数据应用程序应准确设置应用程序缓冲区中的数据长度以及起点应用程序明确定义TargetType时容易发现应用程序的错误

但应用程序向TargetType定义SQL_DEFAULT时SQLBindCol 将变更源数据或将代码应用到其他列因此可能适用于与应用程序中的目标数据类型不同的数据类型列中此时应用程序并不能决定获取的column数据的起点或长度这可能会导致未上报的数据错误或内存冲突

SQLBindParameter

兼容性

导入版本：ODBC 2.0

遵从标准：ODBC

概要

SQLBindParameter在SQL语句的参数标记绑定缓冲区

语句

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

参数

StatementHandle

【输入】命令语句柄

ParameterNumber

【输入】从1开始递增的参数编号

InputOutputType

【输入】参数的类型

ValueType

【输入】参数的C数据类型

ParameterType

【输入】参数的SQL数据类型

ColumnSize

【输入】该参数标记的column或表达式的大小

DecimalDigits

【输入】该参数标记的column或表达式的小数点位数

ParameterValuePtr

【延迟的输入】参数的数据缓冲区指针

BufferLength

【输入/输出】ParameterValuePtr 缓冲区的字节长度

StrLen_or_IndPtr

【延迟的输入】参数的长度/指示符的的指针

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
07006	Restricted data type attribute violation	ValueType参数数据类型无法转换为ParameterType参数数据类型
07009	Invalid descriptor index	ParameterNumber参数值小于1
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY003	Invalid application buffer type	ValueType参数值为无效的C数据类型

SQLSTATE	报错	说明
HY004	Invalid SQL data type	ParameterType参数值为无效的SQL数据类型
HY009	Invalid argument value	ParameterValuePtr参数及StrLen_or_IndPtr 参数为空指针时 InputOutputType 参数不是SQL_PARAM_OUTPUT InputOutputType 参数为SQL_PARAM_OUTPUT时 ParameterValuePtr参数为空指针C类型为字符或二进制 BufferLength值大于0
HY010	Function sequence error	调用SQLExecuteSQLExecDirect 后返回SQL_NEED_DATA传送 所有data-at-execution 变量之前调用了该函数
HY021	Inconsistent descriptor information	完整性检查中描述符信息没有一致性
HY090	Invalid string or buffer length	BufferLength值小于0
HY104	Invalid precision or scale value	ColumnSize及DecimalDigits中指定的值超出了ParameterType 参数的SQL数据支持范围
HY105	Invalid parameter type	InputOutType 参数值无效

SQLSTATE	报错	说明
HYC00	Optional feature not implemented	驱动程序不支持ValueType参数值与ParameterType参数值的转换 ParameterType参数值有效但驱动程序不支持

说明

应用程序为了绑定SQL语句的每个参数标记而调用SQLBindParameter绑定有效至应用程序再次调用SQLBindParameter或以SQL_RESET_PARAMS选项调用SQLFreeStmt或调用SQLSetDescField后将APD的SQL_DESC_COUNT头部字段设置为0为止

ParameterNumber 参数

调用SQLBindParameter时如果ParameterNumber大于SQL_DESC_COUNT的值则为了把SQL_DESC_COUNT值增加至ParameterNumber而调用SQLSetDescField

InputOutputType 参数

InputOutputType参数指定参数的类型此参数设置IPD的SQL_DESC_PARAMETER_TYPE字段

InputOutputType参数为以下值中的一个

- SQL_PARAM_INPUT：显示除procedure或SELECT INTO语句外的SQL语句参数例如INSERT INTO Employee VALUES (?, ?, ?) 的参数为输入参数
    - 执行命令时驱动程序传送参数数据此时*ParameterValuePtr缓冲区应包含有效的输入值或*StrLen_or_IndPtr缓冲区应包含

SQL_NULL_DATASQL_DATA_AT_EXECSQL_LEN_DATA_AT_EXEC Macro的结果

- SQL_PARAM_INPUT_OUTPUT：显示procedure的输入/输出参数
  - 执行命令时驱动程序传送参数数据此时*ParameterValuePtr缓冲区应包含有效的输入值或*StrLen_or_IndPtr缓冲区应包含SQL_NULL_DATASQL_DATA_AT_EXECSQL_LEN_DATA_AT_EXEC Macro的结果
  - 执行命令后驱动程序向应用程序的参数返回数据如果无法向输入/输出参数返回值则驱动程序在*StrLen_or_IndPtr缓冲区设置SQL_NULL_DATA
- SQL_PARAM_OUTPUT：参数显示procedure的返回值或procedureSELECT INTO语句的输出参数例如SELECT ID INTO ? FROM Employee WHERE NAME = 'Paul'为返回ID的输出参数
  - 执行命令后应用程序的ParameterValuePtr与StrLen_or_IndPtr参数不为空指针时驱动程序向参数返回数据否则丢弃输出结果
  - 如果无法向输出参数返回值则驱动程序在*StrLen_or_IndPtr 缓冲区设置SQL_NULL_DATA

ValueType 参数

ValueType参数指定参数的C数据类型此参数设置APD的SQL_DESC_TYPESQL_DESC_CONCISE_TYPESQL_DESC_DATETIME_INTERVAL_CODE字段值

ValueType参数为interval数据类型中的一个时

- APD的ParameterNumber记录的SQL_DESC_TYPE字段设置为SQL_INTERVAL
- SQL_DESC_CONCISE_TYPE字段设置为concise interval数据类型
- SQL_DESC_DATETIME_INTERVAL_CODE字段设置为特定interval数据的sub code
- interval leading precision的默认值为(2)interval seconds precision的默认值为(6)并各自设置于APD的SQL_DESC_DATETIME_INTERVAL_PRECISION和SQL_DESC_PRECISION字段

- 如果默认precision和scale不合适时应用程序应调用SQLSetDescField或SQLSetDescRec后明确设置对应的描述符字段

ValueType 参数为datetime数据类型中的一个时

- APD的ParameterNumber 记录的SQL_DESC_TYPE字段设置为SQL_DATETIME
- SQL_DESC_CONCISE_TYPE字段设置为concise date C 数据类型
- SQL_DESC_DATETIME_INTERVAL_CODE字段设置为特定datetime数据的sub code

ValueType 参数为SQL_C_NUMERIC时

- precision的默认值为(38)scale的默认值为(0)并各自设置于APD的SQL_DESC_PRECISION及SQL_DESC_SCALE字段
- 如果默认precision和scale不合适时应用程序应调用SQLSetDescField或SQLSetDescRec后明确设置对应描述符字段

ParameterType 参数

ParameterType参数指定参数的SQL数据类型此参数设置IPD的SQL_DESC_TYPESQL_DESC_CONCISE_TYPESQL_DESC_DATETIME_INTERVAL_CODE字段值

ParameterType参数为datetime 数据类型中的一个时

- IPD的SQL_DESC_TYPE字段设置为SQL_DATETIME
- SQL_DESC_CONCISE_TYPE字段设置为concise datetime SQL 数据类型
- SQL_DESC_DATETIME_INTERVAL_CODE字段设置为对应datetime的sub code

ParameterType参数为interval 数据类型中的一个时

- IPD的SQL_DESC_TYPE字段设置为SQL_INTERVAL
- SQL_DESC_CONCISE_TYPE字段设置为concise SQL interval 数据类型
- SQL_DESC_DATETIME_INTERVAL_CODE字段设置为对应interval的sub code
- IPD的SQL_DESC_DATETIME_INTERVAL_PRECISION字段中设置interval leading precisionSQL_DESC_PRECISION字段中设置interval second precision
- 当SQL_DESC_DATETIME_INTERVAL_PRECISION及SQL_DESC_PRECISION的默认值不合适时应用程序将调用SQLSetDescField后设置

ParameterType 参数为SQL_NUMERIC时

- precision的默认值为(38)scale的默认值为(0)各自设置于IPD的SQL_DESC_PRECISION和SQL_DESC_SCALE字段
- 如果默认precision和scale不合适时应用程序应调用SQLSetDescField或SQLSetDescRec后明确设置对应描述符字段

ColumnSize 参数

ColumnSize参数为对应参数标记的column或表达式的大小此参数根据ParameterType参数的SQL数据类型设置IPD的不同字段

- ParameterType为SQL_CHARSQL_VARCHARSQL_LONGVARCHARSQL_BINARYSQL_VARBINARYSQL_LONGVARBINARYconcise SQL datetime或interval数据类型时在IPD的SQL_DESC_LENGTH字段中设置ColumnSize的值
- ParameterType为SQL_DECIMALSQL_NUMERICSQL_FLOATSQL_REALSQL_DOUBLE时在IPD的SQL_DESC_PRECISION字段中设置ColumnSize的值

- 其他数据类型的情况忽略ColumnSize参数

DecimalDigits 参数

- ParameterType为SQL_TYPE_TIMESQL_TYPE_TIMESTAMPSQL_INTERVAL_SECONDSQL_INTERVAL_DAY_TO_SECONDSQL_INTERVAL_HOUR_TO_SECONDSQL_INTERVAL_MINUTE_TO_SECOND时在IPD的SQL_DESC_PRECISION字段中设置DecimalDigits的值
- ParameterType为SQL_NUMERICSQL_DECIMAL时在IPD的SQL_DESC_SCALE字段中设置DecimalDigits的值
- 其他数据类型的情况忽略DecimalDigits参数

ParameterValuePtr 参数

调用SQLExecuteSQLExecDirect时ParameterValuePtr指参数的实际数据数据应为指定为ValueType的类型此参数设置APD的SQL_DESC_DATA_PTR字段

*StrLen_or_IndPtr为SQL_LEN_DATA_AT_EXEC(length) Macro的结果或SQL_DATA_AT_EXEC时ParameterValuePtr是与参数相关的应用程序定义的指针值执行SQLParamData时返回给应用程序例如ParameterValuePtr可以是并非用于参数编号数据指针应用程序绑定输入参数的结构体的指针等非0的token

InputOutputType参数为SQL_PARAM_INPUT_OUTPUT或SQL_PARAM_OUTPUT时ParameterValuePtr应为存储输出值的缓冲区指针

SQL_ATTR_PARAMSET_SIZE命令属性值大于1时ParameterValuePtr指数组单条SQL语句处理整个数组的输入或输入/输出参数的输入值返回输入/输出或输出参数输出值的数组

BufferLength 参数

对字符串和二进制C数据SQL_ATTR_PARAMSET_SIZE命令属性值为1时指定*ParameterValuePtr缓冲区的长度SQL_ATTR_PARAMSET_SIZE命令属性值大于1时指定*ParameterValuePtr数组元素的长度

对输入与输出BufferLength均用于决定*ParameterValuePtr数组的位置此参数设置APD的SQL_DESC_OCTET_LENGTH字段

对输入/输出或输出参数BufferLength用于决定是否截断输出

- 字符C数据的情况如果可返回的数据字节长度大于或等于BufferLength*ParameterValuePtr的数据被截断为BufferLength -1字节并以null结束
- 二进制C数据的情况如果可返回的数据字节长度大于或等于BufferLength*ParameterValuePtr的数据被截断为BufferLength字节
- 其他C数据类型的情况忽略BufferLength参数

StrLen_or_IndPtr 参数

调用SQLExecute或SQLExecDirect时StrLen_or_IndPtr参数包含以下中的一个（StrLen_or_IndPtr设置APD的SQL_DESC_OCTET_LENGTH_PTR和SQL_DESC_INDICATOR_PTR）

- 存储于*ParameterValuePtr中的参数的长度忽略除字符和二进制C数据外的所有
- SQL_NTS：参数值为以null终止的字符串
- SQL_NULL_DATA：参数值为NULL
- SQL_LEN_DATA_AT_EXEC(length)Macro的结果：执行SQLPutData时传送参数数据
    - 如果ParameterType参数为SQL_LONGVARBINARYSQL_LONGVARCHAR或长数据类型

SQLGetInfo的SQL_NEED_LONG_DATA_LEN信息返回Y时length为参数传送的数据字节数

- SQLGetInfo的SQL_NEED_LONG_DATA_LEN信息为N时length不能为负数并忽略此值
- 例如多次调用SQLPutData传送10,000字节的SQL_LONGVARCAHR参数数据时*StrLen_or_IndPtr设置为SQL_LEN_DATA_AT_EXEC(10000)即可

- SQL_DATA_AT_EXEC：执行SQLPutData时传送参数数据

StrLen_or_IndPtr为空指针时驱动程序认为所有输入参数值并非空值字符及二进制数据以null终止InputOutputType为SQL_PARAM_OUTPUTParameterValuePtr和StrLen_or_IndPtr均为空指针时驱动程序将丢弃输出值

InputOutputType参数为SQL_PARAM_INPUT_OUTPUTSQL_PARAM_OUTPUT时StrLen_or_IndPtr指SQL_NULL_DATA*ParameterValuePtr可返回的字节数（除字符数据的null终止字节）SQL_NO_TOTAL（无法判断返回的字节数的情况）

SQL_ATTR_PARAMSET_SIZE命令属性值大于1时StrLen_or_IndPtr为SQLLEN值的数组

参数值的传送

应用程序可通过*ParameterValuePtr缓冲区或多次调用SQLPutData传送参数值通过SQLPutData传送的参数称为data-at-execution参数其通常用于传送SQL_LONGVARBINARY和SQL_LONGVARCHAR参数的数据也能与其他参数混用

为了传送参数值应用程序需执行以下步骤

1. 为了绑定参数值(ParameterValuePtr参数)和长度/指示符(StrLen_or_IndPtr参数)的缓冲区对每个参数调用SQLBindParameter对于data-at-execution参数ParameterValuePtr为参数编号或数据指针等应用程序定义的指针值此值在后续再返回而且可用于识别参数

2. 在*ParameterValuePtr和*StrLen_or_IndPtr缓冲区中设置输入及输入/输出参数

- 普通参数的情况应用程序向*ParameterValuePtr缓冲区输入参数值在*StrLen_or_IndPtr缓冲区输入该值的长度
- data-at-execution参数的情况应用程序在*StrLen_or_IndPtr缓冲区输入SQL_LEN_DATA_AT_EXEC(length) Macro(调用ODBC 2.0驱动时)的结果

3. 调用SQLExecute或SQLExecDirect执行SQL命令

- 如果没有data-at-execution参数则结束进程
- 有data-at-execution参数则函数将返回SQL_NEED_DATA

4. 为了调用SQLParamData后处理的第一个data-at-execution参数回收SQLBindParameter的ParameterValuePtr参数指定的应用程序定义值SQLParamData返回SQL_NEED_DATA

Note:

Data-at-execution参数类似于data-at-execution列但SQLParamData返回的值不同

Data-at-execution参数为命令与SQLExecDirect或SQLExecute同时执行时通过SQLPutData传送的SQL命令的参数其绑定到SQLBindParameter

SQLParamData返回的值为向SQLBindParameter的ParameterValuePtr参数传送的指针值data-at-execution列是更新或添加到SQLBulkOperations或通过SQLSetPos更新时传送的行集合的column其绑定到SQLBindColSQLParamData返回的值为要处理的*TargetValuePtr(调用SQLBindCol设置的) 缓冲区中的行地址

5. 多次调用SQLPutData传送参数的数据如果数据值大于SQLPutData的*ParameterValuePtr缓冲区中定义的值则需要调用一次以上传送以字符二进制或数据源指定数据形式的字符数据或二进制C数据时允许对同一个参数调用多次SQLPutData

6. 再次调用SQLParamData传递参数数据已全部传送的信号

- 如果有一个以上的data-at-execution参数SQLParamData返回SQL_NEED_DATA后处理应用程序定义的data-at-execution参数值应用程序重复处理第4与第5步骤
- 如果没有data-at-execution参数则结束进程当命令执行成功时SQLParamData返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO当命令执行失败时返回SQL_ERROR此时SQLParamData可由任何SQLSTATE返回也可以被用于执行命令的SQLExecDirect或SQLExecute函数返回
- 应用程序回收命令生成的所有结果集后输入/输出或输出参数的输出值才能用于*ParameterValuePtr和*StrLen_or_IndPtr缓冲区

调用SQLExecute或SQLExecDirect把命令设置为SQL_NEED_DATA状态此时应用程序只能与命令或命令相关的连接句柄一起调用SQLCancelSQLGetDiagFieldSQLGetDiagRecSQLGetFunctionsSQLParamData或SQLPutData

如果命令或命令相关的连接句柄调用非上述函数的其他函数时函数将返回SQLSTATE HY010(Function sequence error)SQLParamData或SQLPutData出错或SQLParamData返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO或取消命令时命令不再是SQL_NEED_DATA状态

如果驱动程序仍需要data-at-execution参数的数据时应用程序调用SQLCancel驱动程序将取消执行命令此时应用程序可再次调用SQLExecute或SQLExecDirect

参数数组的使用

应用程序使用以下两种方法准备参数标记和命令并传送参数数组

- 驱动程序依赖于数组处理能力进行处理此时参数与其命令语被视为一个单位Oracle就是支持数组处理能力的数据源之一

- 也有驱动程序生成SQL命令的批处理的方法每一个SQL命令以参数数组中的一个参数集合执行批处理参数数组不能用于UPDATE WHERE CURRENT OF命令

处理参数数组时每个参数集合使用一个或整个作为一个使用结果集合/行数SQLGetInfo中的SQL_PARAM_ARRAY_ROW_COUNTS选项表示行数是否可用于每个参数集合(SQL_PARC_BATCH)或仅可用于一个行数(SQL_PARC_NO_BATCH)

SQLGetInfo中的SQL_PARAM_ARRAY_SELECTS选项表示结果集是否可用于每个参数集合(SQL_PAS_BATCH)或仅可用于一个结果集合(SQL_PAS_NO_BATCH)如果驱动程序不允许result set-generating命令与参数数组一起执行则SQL_PARAM_ARRAY_SELECTS返回SQL_PAS_NO_SELECT
详细内容参考**SQLGetInfo**

为了支持参数数组SQL_ATTR_PARAMSET_SIZE命令属性指定参数值的数量如果字段大于1APD的SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTR和SQL_DESC_OCTET_LENGTH_PTR字段应指数组每个数组的元素等于SQL_ATTR_PARAMSET_SIZE的值

APD的SQL_DESC_ROWS_PROCESSED字段指包含包含错误的已完成处理的参数集合数量的缓冲区如完成处理的各个参数集合驱动程序在缓冲区中存储新的值空指针时不返回任何值
使用参数数组时即使通过函数设置返回SQL_ERROR也生成APD的SQL_DESC_ROWS_PROCESSED_PTR字段指向的值如果返回SQL_NEED_DATA则APD的SQL_DESC_ROWS_PROCESSED_PTR字段指向的值设置为要处理的参数值

列方向（Column-Wise）参数绑定

在列方向绑定中应用程序将分开的参数及长度/指示符数组绑定到各个参数

为了使用列方向绑定应用程序首先把SQL_ATTR_PARAM_BIND_TYPE命令属性设置为SQL_PARAM_BIND_BY_COLUMN（其为默认值）为了绑定各列应用程序执行如下步骤

1. 分配参数缓冲区数组
2. 分配长度/指示符缓冲区数组

> Note:
> 使用列方向绑定时如果应用程序直接写入描述符则分离的数组可用于长度与指示符数据

3. 与以下参数一起调用SQLBindParameter

- ValueType为参数缓冲区数组中的单个元素的C类型
- ParameterType为参数的SQL类型
- ParameterValuePtr为参数缓冲区数组的地址
- BufferLength为参数缓冲区数组中的单个元素的大小数据为固定长度数据类型时忽略BufferLength参数

行方向（Row-Wise）参数绑定

在行方向绑定中应用程序对要绑定的各个参数定义包含参数与长度/指示符缓冲区的结构体

为了使用行方向绑定应用程序需执行如下步骤

1. 定义维持（包含参数与长度/指示符缓冲区）一个参数集合的结构体并分配此结构体的数组

> Note:

> 使用行方向绑定时如果应用程序直接写入描述符则其他的字段可用于长度和指示符数据

2. SQL_ATTR_PARAM_BIND_TYPE命令属性设置包含单个参数集合的结构体的大小或要绑定的参数缓冲区的实例的大小该长度应包含绑定参数的所有空间而且该长度应包含用于绑定参数和结构体填充的所有空间或确保当绑定参数的地址按指定长度递增时结果必须指向下一个参数的开始使用ANSI C的sizeof运算符来保证此操作
3. 对绑定的各个参数与以下参数一起调用SQLBindParameter

- ValueType为要绑定到列的参数缓冲区成员的类型
- ParameterType为参数的SQL类型
- ParameterValuePtr为第一个数组元素中的参数缓冲区成员的地址
- BufferLength为参数缓冲区成员的大小
- StrLen_or_IndPtr为要绑定的长度/指示符成员的地址

错误信息

当驱动程序不执行批处理等参数数组时(SQL_PARAM_ARRAY_ROW_COUNTS选项与SQL_PARC_NO_BATCH相同)像执行单个命令一样处理错误信息

执行批处理时为了查看SQL语句的参数或参数数组的参数是否为SQLExecDirect或SQLExecute返回错误的原因可以使用IPD的SQL_DESC_ARRAY_STATUS_PTR头部字段此字段包含参数值的各行的状态信息如果此字段表示发生错误则检测数据结构体中的字段会显示失败的行与参数APD的SQL_DESC_ARRAY_SIZE头部字段定义数组中的column数量可以由SQL_ATTR_PARAMSET_SIZE命令属性设置此值

Note:

APD的SQL_DESC_ARRAY_STATUS_PTR头部字段用于忽略参数忽略参数的详细内容参考**忽略参数集合**

SQLExecute或SQLExecDirect返回SQL_ERROR时IPD的SQL_DESC_ARRAY_STATUS_PTR字段指向的数组中的元素包含SQL_PARAM_ERRORSQL_PARAM_SUCCESSSQL_PARAM_SUCCESS_WITH_INFOSQL_PARAM_UNUSED或SQL_PARAM_DIAG_UNAVAILABLE

为了数组中的各元素诊断数据结构体包含一个以上的状态记录结构体的SQL_DIAG_ROW_NUMBER字段显示引起错误的参数值的行编号如果能找到引起错误的参数行中的特定参数则参数编号将包含在SQL_DIAG_COLUNM_NUMBER字段中

由于SQLExecute或SQLExecDirect强行结束之前参数而引起错误并未使用参数时设置SQL_PARAM_UNUSED例如有50个参数执行引起取消SQLExecute或SQLExecDirect的第40个参数集合时发生错误则在从第41到50的参数状态数组中设置SQL_PARAM_UNUSED

当驱动程序将参数数组作为单个单元时输入SQL_PARAM_DIAG_UNAVAILABLE
因此不对每个参数生成错误信息

处理一个参数集合时部分错误会停止数组内后续的参数集合的处理其他错误不会影响后续的参数处理停止处理的错误由驱动程序定义如果未停止处理则处理数组中的所有参数并返回SQL_SUCCESS_WITH_INFO错误结果SQL_ATTR_PARAMS_PROCESSED_PTR定义的缓冲区设置为包含错误的已处理的整个参数集合数量

Note:

参数数组处理发生错误时ODBC在ODBC2.x与ODBC 3.x中的处理方式是不同的

ODBC 2.x中函数返回SQL_ERROR并结束处理SQLParamOptions的pirow参数指向的缓冲区包含错误行的数量

ODBC 3.x中函数返回SQL_SUCCESS_WITH_INFO可能结束处理或继续其处理继续处理时
SQL_ATTR_PARAMS_PROCESSED_PTR定义的缓冲区设置为包含错误的已处理的所有参数值这种变更可能会引起当前应用程序的故障

当SQLExecute或SQLExecDirect在结束参数数组中的所有参数集合处理之前返回SQL_ERROR或SQL_NEED_DATA等时状态数组将包含已处理的参数的状态IPD的SQL_DESC_ROWS_PROCESSED_PTR字段指向的位置包含参数数组中引起SQL_ERROR或SQL_NEED_DATA错误代码的行编号当参数的数组传送给SELECT命令时驱动程序定义是否可用状态数组值其可在已执行命令语或获取结果集等之后使用

忽略参数集合

APD的SQL_DESC_ARRAY_STATUS_PTR字段可用于显示SQL命令语中需要忽略的绑定参数集合执行过程中驱动程序为了忽略一个以上的参数集合应用程序需执行以下步骤

1. 调用SQLSetDescField使APD的SQL_DESC_ARRAY_STATUS_PTR头部字段指向包含状态信息的SQLUSMALLINT值的数组另外此字段可通过SQLSetStmtAttr中的Attribute参数的SQL_ATTR_PARAM_OPERATION_PTR进行设置其允许在不获取应用程序描述符句柄的情况下设置字段
2. 由APD的SQL_DESC_ARRAY_STATUS_PTR定义的数组元素设置为以下两个值中的一个

- SQL_PARAM_IGNORE：执行命令时排除此行
- SQL_PARAM_PROCEED：执行命令时包含此行

3. 调用SQLExecDirect或SQLExecute执行已准备的(prepared)命令APD的SQL_DESC_ARRAY_STATUS_PTR定义的数组遵守以下规则

- 指针默认设置为空
- 指针为NULL时像所有元素设置为SQL_ROW_PROCEED一样使用所有参数集合
- 元素设置为SQL_PARAM_PROCEED时也不保证运算一定使用特定参数集合
- SQL_PARAM_PROCEED在头文件中定义为0

应用程序可以设置APD的SQL_DESC_ARRAY_STATUS_PTR字段与IRD的SQL_DESC_ARRAY_STATUS_PTR字段可参照相同的数组其有利于将参数绑定到行数据参数可根据行数据的状态进行忽略

与SQL_PARAM_IGNORE一起以下状态码可使SQL命令忽略参数设置

- SQL_ROW_DELETED
- SQL_ROW_UPDATED
- SQL_ROW_ERROR

与SQL_PARAM_PROCEED一起以下状态码可使SQL命令处理参数设置

- SQL_ROW_SUCCESS
- SQL_ROW_SUCCESS_WITH_INFO
- SQL_ROW_ADDED

再次绑定参数

应用程序可以包含很多参数但拥有仅调用使用个别参数的SQLExecDirect或SQLExecute的缓冲区领域时参数的再次绑定非常有用缓冲区领域的剩余空间可通过offset修改原有绑定并用于设置下一个参数

APD的SQL_DESC_BIND_OFFSET_PTR头部字段指向绑定的offset字段并非NULL时驱动程序反向引用指针如果没有SQL_DESC_DATA_PTRSQL_INDICATOR_PTR的值并SQL_DESC_OCTET_LENGTH_PTR字段为NULL指针时执行过程中在描述符记录中的字段中添加反向引用值

offset在再次绑定后有效SQL_DESC_BIND_OFFSET_PTR字段为对offset的指针而不是offset本身因此当应用程序变更描述符字段时不需要调用SQLSetDescField或SQLSetDescRec也可以直接变更offset指针的默认值为NULL

ARD的SQL_DESC_BIND_OFFSET_PTR字段可调用SQLSetDescField或通过SQL_ATTR_PARAM_BIND_OFFSET_PTR在SQLSetStmtAttr的Attribute参数中设置offset绑定总是在SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTR及SQL_DESC_OCTET_LENGTH_PTR字段中直接添加值变更offset的值时每个描述符字段中继续添加新的值新的offset不会加上之前的offset值后添加

描述符

参数的绑定方式取决于APD和IPD的字段SQLBindParameter中的参数用于设置描述符字段应用程序不获取描述符句柄也可以调用SQLBindParameter因此使用SQLBindParameter更有效率但也可以使用SQLSetDescField函数设置字段

> Caution:
>
> 对一个命令语调用SQLBindParameter时会影响其他命令语在明确分配了命令语相关的ARD而且与其他命令语关联时会出现这种情况SQLBindParameter修改APD的字段因此修改会影响与描述符关联的所有命令语如果并非故意应用程序调用SQLBindParameter之前需要从其他命令语解除与此描述符的关联性

从概念上讲SQLBindParameter需执行以下过程

1. 调用SQLGetStmtAttr获取APD句柄
2. 调用SQLGetDescField获取APD的SQL_DESC_COUNT字段ColumnNumber值超过SQL_DESC_COUNT值时调用SQLSetDescField把SQL_DESC_COUNT值增加至ColumnNumber
3. 多次调用SQLSetDescField设置APD的下一个字段的值

- SQL_DESC_TYPE和SQL_DESC_CONCISE_TYPE设置为ValueType值
    - 如果ValueType为datetime或interval subtype的隐含标识符中的一个则将其排除将SQL_DESC_TYPE分别设置为SQL_DATETIME或SQL_INTERVALSQL_DESC_CONCISE_TYPE设置为隐含标识符SQL_DESC_DATETIME_INTERVAL_CODE对应datetime或interval subcode进行设置
- SQL_DESC_OCTET_LENGTH字段设置为BufferLength值
- SQL_DESC_DATA_PTR字段设置为ParameterValue值
- SQL_DESC_OCTET_LENGTH_PTR字段设置为StrLen_or_Ind值
- SQL_DESC_INDICATOR_PTR字段设置为StrLen_or_Ind值
- StrLen_or_Ind指定参数值的指示符信息与长度

4. 调用SQLGetStmtAttr获取IPD句柄
5. 调用SQLGetDescField获取IPD的SQL_DESC_COUNT字段如果ColumnNumber值超过SQL_DESC_COUNT值则调用SQLSetDescField把SQL_DESC_COUNT值增加至ColumnNumber

值

6. 多次调用SQLSetDescField设置IPD 的下一个字段的值

- SQL_DESC_TYPE和SQL_DESC_CONCISE_TYPE设置为ParameterType值
    - 如果ParameterType为datetime或interval subtype的隐含标识符中的一个则将其排除将SQL_DESC_TYPE分别设置为SQL_DATETIME或SQL_INTERVALSQL_DESC_CONCISE_TYPE设置为隐含标识符SQL_DESC_DATETIME_INTERVAL_CODE对应datetime或interval subcode进行设置
- 对ParameterType适当设置一个以上的SQL_DESC_LENGTHSQL_DESC_PRECISION和SQL_DESC_DATETIME_INTERVAL_PRECISION
- SQL_DESC_SCALE设置为DecimalDigits值

SQLBindParameter调用失败时不定义APD中要设置的描述符字段的内容不变更APD的SQL_DESC_COUNT字段此外不定义IPD中适当记录的SQL_DESC_LENGTHSQL_DESC_PRECISIONSQL_DESC_SCALE和SQL_DESC_TYPE字段不变更IPD的SQL_DESC_COUNT字段

SQLBrowseConnect

不支持

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLBrowseConnect发现连接数据源所需的属性与属性值支持重复性列出的方法

语句

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

SQLBulkOperations

不支持

兼容性

导入版本：ODBC 3.0

遵从标准：ODBC

概要

SQLBulkOperations执行批量插入以及使用书签的更新删除获取等批量书签操作

语句

```
SQLRETURN SQLBulkOperations(
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT   Operation);
```

SQLCancel

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLCancel取消处理中的命令语

语句

```
SQLRETURN SQLCancel(
    SQLHSTMT     StatementHandle);
```

参数

StatementHandle

【输入】命令语句柄

返回

SQL_SUCCESS, SQL_ERROR 或 SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	调用了一个异步执行连接到 StatementHandle 的句柄的函数 当调用 SQLCancel 函数时此异步函数仍在运行连接到StatementHandle的连接句柄上正在进行异步操作所以失败了
HYT00	Connection timeout expired	在连接到数据源之前登录时间超时此登录时间限制可通过SQLSetConnectAttr的SQL_ATTR_LOGIN_TIMEOUT进行设置

说明

SQLCancel可取消Statement的如下类型的操作

- 异步执行命令语句的函数
- 需要数据的命令语句的函数
- 在其他线程上执行命令语句的函数

对异步执行命令语句的函数或需要数据的命令语句调用SQLCancel进行取消操作会擦除诊断记录由SQLCancel发布自诊断记录 但是如果是为了取消其他线程上执行的命令语句而调用SQLCancel的话不会擦除函数的诊断记录也不会发布自诊断记录

为了取消其他线程上执行的命令语句可临时生成新的连接用于取消该命令语句

SQLCancelHandle

兼容性

导入版本：ODBC 3.8

遵从标准：无

概要

SQLCancelHandle取消命令语的处理

详情请参考**SQLCancel**

语句

```
SQLRETURN SQLCancelHandle(
    SQLSMALLINT  HandleType,
    SQLHANDLE    Handle);
```

参数

HandleType

【输入】句柄类型只支持SQL_HANDLE_STMT

Handle

【输入】必须是命令语句句柄

返回

SQL_SUCCESS, SQL_ERROR 或 SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	异步执行的命令语句关联函数被与句柄连接的命令语句句柄调用 此异步函数在SQLCancelHandle被调用时仍在执行句柄为命令语句时关联的连接句柄中正在执行异步操作所以失败了
HYT00	Connection timeout expired	在连接到数据源之前登录时间超时此登录时间限制可通过SQLSetConnectAttr的SQL_ATTR_LOGIN_TIMEOUT进行设置
IM001	Driver does not support function	不支持SQL_HANDLE_STMT以外的HandleType

SQLCloseCursor

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLCloseCursor关闭命令中已打开的游标并丢弃剩余的结果

语句

```
SQLRETURN SQLCloseCursor(
    SQLHSTMT     StatementHandle);
```

参数

StatementHandle

【输入】命令语句柄

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
24000	Invalid cursor state	命令中没有已打开的游标
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	调用SQLExecute, SQLExecDirect后返回了SQL_NEED_DATA在传送所有data-at-execution变量之前调用了此函数

说明

当没有已打开的游标时SQLCloseCursor返回SQLSTATE 24000(Invalid cursor state)调用SQLCloseCursor类似于以SQL_CLOSE选项调用SQLFreeStmt但当没有已打开的游标时SQLCloseCursor返回SQLSTATE 24000 (Invalid cursor state)但以SQL_CLOSE选项调用SQLFreeStmt不影响应用程序

SQLColAttribute

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLColAttribute返回结果集column的描述符信息描述符信息返回为字符串或整数值

语句

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

参数

StatementHandle

【输入】命令语句柄

ColumnNumber

【输入】IRD中要检索字段值的记录编号此值从1开始对应顺序递增的结果数据的column编号column可以以任何顺序描述

ColumnNumber中可以指定0号column除SQL_DESC_TYPE和SQL_DESC_OCTET_LENGTH外其他情况返回未定义的值

FieldIdentifier

【输入】描述符句柄其定义要在IRD中查询的字段（例：SQL_COLUMN_TABLE_NAME）

CharacterAttributePtr

【输出】IRD的ColumnNumber行的FieldIdentifier字段值为字符串时是返回字段值的缓冲区指针字段值并非字符串时不使用

CharacterAttributePtr为NULL时StringLengthPtr返回可返回的总字节数（不包含null终止字符 ）

BufferLength

【输入】FieldIdentifier为ODBC中定义的字段并CharacterAttributePtr指向字符串或二进制缓冲区时其为*CharacterAttributePtr的长度FieldIdentifier为ODBC定义的字段并*CharacterAttributePtr为整数时忽略此参数

StringLengthPtr

【输出】返回*CharacterAttributePtr中可返回的总字节数（不包括空终止字符 ）的指针字符数据的情况可返回的字节数大于或等于BufferLength时*CharacterAttributePtr的说明信息被截断为BufferLength - 1长度并被驱动程序以null终止

其他数据类型的情况忽略BufferLength的值

NumericAttributePtr

【输出】如果IRD的ColumnNumber行的FieldIdentifier字段值为SQL_DESC_COLUMN_LENGTH等数字时是返回字段值的缓冲区指针如果字段值并非数字时不使用

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
01004	String data, right truncated	*CharacterAttributePtr缓冲区大小不足以返回整个字符串因此字符串被截断没有被截断的字符串长度返回给*StringLengthPtr（函数返回SQL_SUCCESS_WITH_INFO）
07005	Prepared statement not a cursor-specification	命令未返回结果集FieldIdentifier不是SQL_DESC_COUNT没有可说明的column
07009	Invalid descriptor index	ColumnNumber参数值大于结果集的column数
HY000	General error	无特定SQLSTATE的错误

SQLSTATE	报错	说明
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	调用SQLPrepreSQLExecDirect目录函数之前调用了此函数 调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution变量之前调用了此函数
HY090	Invalid string or buffer length	*CharacterAttributePtr为字符串 BufferLength小于0但不是SQL_NTS
HY091	Invalid descriptor field identifier	FieldIdentifier参数值不是定义的值

说明

SQLColAttribute向*NumericAttributePtr或*CharacterAttributePtr返回信息整数信息为SQLLEN值返回至*NumericAttributePtr其他所有类型返回至*CharacterAttributePtr向*NumericAttributePtr返回信息时驱动程序忽略CharacterAttributePtrBufferLengthStringLengthPtr向*CharacterAttributePtr返回信息时驱动程序忽略NumericAttributePtr

SQLColAttribute从IRD的描述符字段中返回值用SQLColAttribute返回的FieldIdentifier值可通过恰当的IRD句柄调用SQLGetDescField获取

以下表格为用SQLColAttribute返回的描述符类型NumericAttributePtr类型为SQLLEN*

FieldIdentifier	返回的信息	说明
SQL_DESC_AUTO_UNIQUE_VALUE (ODBC 1.0)	NumericAttributePtr	• SQL_TRUE：自动增加的column • SQL_FALSE：不是自动增加的column或数字类型
SQL_DESC_BASE_COLUMN_NAME (ODBC 3.0)	CharacterAttributePtr	结果集column的默认column名不存在默认column名时（表达式 况）此变量包含空字符串 此信息从IRD的只读字段SQL_DESC_BASE_COLUMN_NAME记录
SQL_DESC_BASE_TABLE_NAME (ODBC 3.0)	CharacterAttributePtr	包含column的默认表名无法定义默认表名或不符合时此变量包 此信息从IRD的只读字段SQL_DESC_BASE_TABLE_NAME记录字
SQL_DESC_CASE_SENSITIVE (ODBC 1.0)	NumericAttributePtr	• SQL_TRUE：为了排列和对比Column区分大小写 • SQL_FALSE：为了排序和对比column不区分大小写或
SQL_DESC_CATALOG_NAME (ODBC 2.0)	CharacterAttributePtr	包含column的表的目录
SQL_DESC_CONCISE_TYPE (ODBC 1.0)	NumericAttributePtr	Concise 数据类型 datetime或interval数据的情况返回SQL_TYPE_TIMESQL_INTERV concise数据 此信息从IRD的SQL_DESC_CONCISE_TYPE记录字段中返回

FieldIdentifier	返回的信息	说明
SQL_DESC_COUNT (ODBC 1.0)	NumericAttributePtr	结果集可使用的column数结果集中没有列时返回0 忽略ColumnNumber 参数 此信息从IRD的SQL_DESC_COUNT头部字段返回
SQL_DESC_DISPLAY_SIZE (ODBC 1.0)	NumericAttributePtr	表示column所需的最大字符数
SQL_DESC_FIXED_PREC_SCALE (ODBC 1.0)	NumericAttributePtr	• SQL_TRUE：列为固定precision并拥有非零的scale • SQL_FALSE：列不是固定precision并拥有非零的scale
SQL_DESC_LABEL (ODBC 2.0)	CharacterAttributePtr	column标签或标题例如column名为EmpName的column可显示为 Name或别名 没有标签时返回column名 column未指定标签或名称时返回空字符串
SQL_DESC_LENGTH (ODBC 3.0)	NumericAttributePtr	字符串或二进制数据类型的最大或实际长度值固定长度的数据类 符长度可变长度的数据类型为实际字符长度该值始终不包含字符 终止字符 此信息从IRD的SQL_DESC_LENGTH记录字段返回
SQL_DESC_LITERAL_PREFIX (ODBC 3.0)	CharacterAttributePtr	此VARCHAR(128)记录字段包含驱动程序识别数据类型的前缀的 不适用前缀的数据类型包含空字符串

FieldIdentifier	返回的信息	说明
SQL_DESC_LITERAL_SUFFIX (ODBC 3.0)	CharacterAttributePtr	此VARCHAR(128)记录字段包含驱动程序识别数据类型的后缀的不适用后缀的数据类型包含空字符串
SQL_DESC_LOCAL_TYPE_NAME (ODBC 3.0)	CharacterAttributePtr	此VARCHAR(128)记录字段包含不同于数据类型的普通名称的数地化（母语）名称没有本地化名称时返回空字符串此字段仅用的字符集根据不同区域有所不同通常默认为服务器的字符集
SQL_DESC_NAME (ODBC 3.0)	CharacterAttributePtr	如果采用column别名则是column的别名否则返回列名两种情况SQL_DESC_UNNAMED设置为SQL_NAMED如果没有column名或字字符串SQL_DESC_UNNAMED设置为SQL_UNNAMED 此信息从IRD的SQL_DESC_NAME记录字段返回
SQL_DESC_NULLABLE (ODBC 3.0)	NumericAttributePtr	此信息从IRD的SQL_DESC_NULLABLE记录字段返回 • SQL_NULLABLE：column可以有空值 • SQL_NO_NULLS：column不能有空值 • SQL_NULLABLE_UNKNOWN：无法知道column是否允
SQL_DESC_NUM_PREC_RADIX (ODBC 3.0)	NumericAttributePtr	如果SQL_DESC_TYPE字段数据类型为approximate numeric数据SQL_DESC_PRECISION字段包含位数（bits）因此该字段包含2如SQL_DESC_TYPE字段数据类型为exact numeric数据类型由于SQL_DESC_PRECISION字段包含十进制数字因此该字段包含10此字的所有数据类型设置为0

FieldIdentifier	返回的信息	说明
SQL_DESC_OCTET_LENGTH (ODBC 3.0)	NumericAttributePtr	字符串或二进制数据类型的字节长度固定长度的字符或二进制类 字节长度可变长度字符或二进制类型时是最大字节长度该值不 字符 此信息从IRD的SQL_DESC_OCTET_LENGTH记录字段返回
SQL_DESC_PRECISION (ODBC 3.0)	NumericAttributePtr	可适用于数字数据类型的precision 与 SQL_TYPE_TIMESQL_TYPE_TIMESTAMP及表示时间间隔的所有 类型的情况该值为可适用的fractional seconds precision 此值从IRD的SQL_DESC_PRECISION记录字段返回
SQL_DESC_SCALE (ODBC 3.0)	NumericAttributePtr	可适用于数字数据类型的scale对于DECIMAL及NUMERIC 数据类 的scale不定义其他所有数据类型 此值从IRD的SQL_DESC_SCALE记录字段返回
SQL_DESC_SCHEMA_NAME (ODBC 2.0)	CharacterAttributePtr	包含column的表的schema

FieldIdentifier	返回的信息	说明
SQL_DESC_SEARCHABLE (ODBC 1.0)	NumericAttributePtr	• SQL_PRED_NONE：WHERE中无法使用column(与ODB SQL_UNSEARCHABLE相同) • SQL_PRED_CHAR：WHERE中可以使用column但只能 ODBC 2.x的SQL_LIKE_ONLY相同) • SQL_PRED_BASIC：WHERE子句中可使用除LIKE以外 运算符(与ODBC 2.x的SQL_EXCEPT_LIKE相同) • SQL_PRED_SEARCHABLE：column可与任何比较运算 于WHERE子句
SQL_DESC_TABLE_NAME (ODBC 2.0)	CharacterAttributePtr	包含column的表名 无法查看表名时返回空字符串
SQL_DESC_TYPE (ODBC 3.0)	NumericAttributePtr	指定SQL数据类型的数值 对于datetime及interval数据类型返回SQL_DATETIME或SQL_INT verbose数据类型 此信息从IRD的SQL_DESC_TYPE记录字段返回
SQL_DESC_TYPE_NAME (ODBC 1.0)	CharacterAttributePtr	从属于数据源的数据类型名称（例如"CHARACTER""CHARACTE VARYING""CHARACTER LONG VARYING"）

FieldIdentifier	返回的信息	说明
SQL_DESC_UNNAMED (ODBC 3.0)	NumericAttributePtr	SQL_NAMED或SQL_UNNAMEDIRD的SQL_DESC_NAME字段中包含 名或column名时返回SQL_NAMED否则返回SQL_UNNAME 此信息从IRD的SQL_DESC_UNNAMED记录字段返回
SQL_DESC_UNSIGNED (ODBC 1.0)	NumericAttributePtr	• SQL_TRUE：列并非unsigned或数字 • SQL_FALSE：列为signed
SQL_DESC_UPDATABLE (ODBC 1.0)	NumericAttributePtr	列可以为 SQL_ATTR_READONLYSQL_ATTR_WRITESQL_ATTR_READWRIT 值

SQLColAttributes

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 3.x中的ODBC 2.0 SQLColAttributes函数被替代为SQLColAttribute函数

详细内容参考**SQLColAttribute**函数

SQLColumnPrivileges

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLColumnPrivileges以结果集返回指定表的column及相关权限目录

语句

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

参数

StatementHandle

【输入】命令语句柄

CatalogName

【输入】目录名称CatalogName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时CatalogName为不区分大小写的标识符设置为SQL_FALSE时CatalogName区分大小写是按字面处理的普通参数

NameLength1

【输入】*CatalogName的长度

SchemaName

【输入】SCHEMA名称SchemaName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时SchemaName为不区分大小写的标识符设置为SQL_FALSE时SchemaName区分大小写是按字面处理的普通参数

NameLength2

【输入】*SchemaName的长度

TableName

【输入】表名此参数不能为空指针TableName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时TableName为不区分大小写的标识符为SQL_FALSE时TableName区分大小写是按字面处理的普通参数

NameLength3

【输入】*TableName的长度

ColumnName

【输入】column名字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时ColumnName为不区分大小写的标识符为SQL_FALSE时ColumnName区分大小写是按照字面处理的模式（pattern）值

NameLength4

【输入】*ColumnName的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
24000	Invalid cursor state	存在通过调用SQLFetchSQLFetchScroll打开的游标
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY009	Invalid use of null pointer	TableName参数为null指针 SQL_ATTR_METADATA_ID命令属性值为SQL_TRUESchemaName或ColumnName为空指针
HY010	Function sequence error	调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution变量之前调用了此函数
HY090	Invalid string or buffer length	名称长度参数中的一个值小于0但不是SQL_NTS
HYT00	Timeout expired	从数据源接收结果集之前语句限制时间超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置

说明

SQLColumnPrivileges返回按照

TABLE_CATTABLE_SCHEMTABLE_NAMECOLUMN_NAMEPRIVILEGE顺序排列的标准结果集

以下为结果集合的column

column名	column编号	数据类型	说明
TABLE_CAT (ODBC 1.0)	1	VARCHAR	目录标识符

column名	column 编号	数据类型	说明
TABLE_SCHEM (ODBC 1.0)	2	VARCHAR	SCHEMA标识符
TABLE_NAME (ODBC 1.0)	3	VARCHAR not NULL	表标识符
COLUMN_NAME (ODBC 1.0)	4	VARCHAR not NULL	Column名对没有名称的column返回空字符串
GRANTOR (ODBC 1.0)	5	VARCHAR	授予权限的用户名称
GRANTEE (ODBC 1.0)	6	VARCHAR not NULL	被授予权限的用户名称

column名	column 编号	数据类型	说明
PRIVILEGE (ODBC 1.0)	7	VARCHAR not NULL	column权限标识符可以是以下中的一个 • SELECT：被授予权限的用户可以查询column数据 • INSERT：被授予权限的用户可以向相关表的column添加数据 • UPDATE：被授予权限的用户可以变更column数据 • REFERENCES：被授予权限的用户可以在约束条件（例如uniquereferentialtable check约束条件）中参考column
IS_GRANTABLE (ODBC 1.0)	8	VARCHAR	被授予权限的用户是否可以向其他用户授予权限 "YES""NO"

SQLColumns

兼容性

导入版本：ODBC 1.0

遵从标准：Open Group

概要

SQLColumns以结果集返回指定表的column名目录

语句

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

参数

StatementHandle

【输入】命令语句柄

CatalogName

【输入】目录名称CatalogName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时CatalogName为不区分大小写的标识符为SQL_FALSE时CatalogName区分大小写是按字面处理的普通参数

NameLength1

【输入】*CatalogName的长度

SchemaName

【输入】SCHEMA名称的字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时SchemaName为不区分大小写的标识符为SQL_FALSE时SchemaName区分大小写是按字面处理的普通参数

NameLength2

【输入】*SchemaName的长度

TableName

【输入】表名的字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时TableName为不区分大小写的标识符为SQL_FALSE时TableName区分大小写是按字面处理的普通参数

NameLength3

【输入】*TableName的长度

ColumnName

【输入】column名字的符串检索模式

SQL_ATTR_METADATA_ID命令属性为SQL_TRUE时ColumnName为不区分大小写的标识符为SQL_FALSE时ColumnName区分大小写是按字面处理的模式值

NameLength4

【输入】*ColumnName的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
24000	Invalid cursor state	存在通过调用SQLFetchSQLFetchScroll打开的游标
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY009	Invalid use of null pointer	TableName参数为null指针 SQL_ATTR_METADATA_ID命令属性值为SQL_TRUESchemaName或ColumnName为空指针
HY010	Function sequence error	调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution变量之前调用了此函数
HY090	Invalid string or buffer length	名称长度参数中有一个值小于0但不是SQL_NTS
HYT00	Timeout expired	从数据源接收结果集之前语句限制时间超时限制时间可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置

说明

通常在数据源的目录中执行检索表或表column信息的命令之前使用此函数SQLColumns用于检索SQLTables返回的所有数据类型相反SQLColAttribute和SQLDescribeCol说明结果集的columnSQLNumResultCols返回结果集的column数

SQLColumns返回以TABLE_CATTABLE_SCHEMTABLE_NAMEORDINAL_POSITION顺序排列的标准结果集

以下为结果集的column

column名	column 编号	数据类型	说明
TABLE_CAT (ODBC 1.0)	1	VARCHAR	目录名
TABLE_SCHEM (ODBC 1.0)	2	VARCHAR	SCHEMA名
TABLE_NAME (ODBC 1.0)	3	VARCHAR not NULL	表名
COLUMN_NAME (ODBC 1.0)	4	VARCHAR not NULL	Column名没有名称的column时返回空字符串
DATA_TYPE (ODBC 1.0)	5	SMALLINT not NULL	SQL 数据类型datetime或interval数据类型时该column返回SQL_TYPE_DATESQL_INTERVAL_YEAR_TO_MONTH等concise数据类型
TYPE_NAME (ODBC 1.0)	6	VARCHAR not NULL	从属于数据源的数据类型名称（例如"CHARACTER""CHARACTER VARYING""CHARACTER LONG VARYING"）

column名	column编号	数据类型	说明
COLUMN_SIZE (ODBC 1.0)	7	INTEGER	DATA_TYPE为SQL_CHAR或SQL_VARCHAR时此column包含column的最大长度字符数为datetime数据column时是将值转换为字符时所需的字符数为数字数据类型时是NUM_PREC_RADIX column的总位数或column允许的位数（bits）为interval数据类型时是由interval leading precision表达时所需的字符数
BUFFER_LENGTH (ODBC 1.0)	8	INTEGER	指定SQL_C_DEFAULT时是传送给SQLGetDataSQLFetchSQLFetchScroll的数据的字节长度
DECIMAL_DIGITS (ODBC 1.0)	9	SMALLINT	正数时小数点右边的有效位数负数时小数点左边的有效位数SQL_TYPE_TIME或SQL_TYPE_TIMESTAMP时该column为fractional seconds的位数包含second的interval数据类型时column是小数点右边的位数(fractional seconds)不能适用DECIMAL_DIGITS的数据类型返回NULL

column名	column编号	数据类型	说明
NUM_PREC_RADIX (ODBC 1.0)	10	SMALLINT	数字型数据类型时是2或10 2时是COLUMN_SIZE和DECIMAL_DIGITS为column允许的位数10时COLUMN_SIZE和DECIMAL_DIGITS为dolumn允许的bits数 不能适用NUM_PREC_RADIX的数据类型返回NULL
NULLABLE (ODBC 1.0)	11	SMALLINT not NULL	• SQL_NULLABLE：column可以有空值 • SQL_NO_NULLS：column不能有空值 • SQL_NULLABLE_UNKNOWN：无法知道column可否使用空值
REMARKS (ODBC 1.0)	12	VARCHAR	Column的说明
COLUMN_DEF (ODBC 3.0)	13	VARCHAR	Column的默认值该值用双引号引住时此column应解析为字符串
SQL_DATA_TYPE (ODBC 3.0)	14	SMALLINT not NULL	IRD的SQL_DESC_TYPE记录字段的SQL数据类型此column除了datetime与interval数据类型外与DATA_TYPE相同对于datetime及interval数据类型此column返回SQL_DATESQL_INTERVAL等nonconcise数据类型可以通过SQL_DATETIME_SUB column决定特定数据类型

column名	column编号	数据类型	说明
SQL_DATETIME_SUB (ODBC 3.0)	15	SMALLINT	Datetime和interval数据类型的sub type code其他类型时返回NULL
CHAR_OCTET_LENGTH (ODBC 3.0)	16	INTEGER	字符或二进制数据类型column的最大字节长度其他类型时返回NULL
ORDINAL_POSITION (ODBC 3.0)	17	INTEGER not NULL	表中的column位置
IS_NULLABLE (ODBC 3.0)	18	VARCHAR	• "NO" : column不能包含NULL • "YES" : column可包含NULL • 无法知道是否为NULL时返回长度为0的字符串

SQLConnect

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLConnect设置驱动程序与数据源之间的连接连接句柄参考状态事务状态错误信息等连接相关的所有信息

语句

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

参数

ConnectionHandle

【输入】连接句柄

ServerName

【输入】数据源名称

NameLength1

【输入】*ServerName的长度

UserName

【输入】用户标识符

NameLength2

【输入】*UserName的长度

Authentication

【输入】认证字符串（一般为密码）

NameLength3

【输入】*Authentication的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE,

SQL_STILL_EXECUTING.

检测

SQLSTATE	报错	说明
08001	Client unable to establish connection	驱动程序无法设置与数据源的连接
08002	Connection name in use	指定的ConnectionHandle已与数据源连接
08004	Server rejected the connection	设定值达到极限时数据源拒绝设置连接
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
28000	Invalid authorization specification	UserName或Authentication参数值不正确
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY090	Invalid string or buffer length	NameLength1NameLength2或NameLength3的值小于0但不是SQL_NTS
HYT00	Timeout expired	连接数据源之前登录限制时间超时该限制时间可通过SQLSetConnectAttr的SQL_ATTR_LOGIN_TIMEOUT设置

说明

驱动程序以$ODBCINI环境变量设置的文件$HOME/.odbc.ini/home/.odbc.ini文件的顺序依次检索用户DSN信息用户DSN中没有输入的DSN时按照系统DSN的$ODBCSYSINI/odbc.ini/etc/odbc.ini文件的顺序检索DSN信息

SQLCopyDesc

不支持

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLCopyDesc将描述符信息从一个描述符句柄复制到另一个描述符句柄

语句

```
SQLRETURN SQLCopyDesc(
    SQLHDESC     SourceDescHandle,
    SQLHDESC     TargetDescHandle);
```

SQLDescribeCol

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLDescribeCol返回结果集column中的一个column的名称类型column大小小数点位数是否允许null等信息在IRD的字段中可使用此信息

语句

```
SQLRETURN SQLDescribeCol(
    SQLHSTMT       StatementHandle,
    SQLUSMALLINT   ColumnNumber,
    SQLCHAR *      ColumnName,
    SQLSMALLINT    BufferLength,
    SQLSMALLINT *  NameLengthPtr,
    SQLSMALLINT *  DataTypePtr,
    SQLULEN *      ColumnSizePtr,
    SQLSMALLINT *  DecimalDigitsPtr,
    SQLSMALLINT *  NullablePtr);
```

参数

StatementHandle

【输入】命令语句柄

ColumnNumber

【输入】从1开始递增的结果集的column编号

ColumnName

【输出】返回column名的以null终止的缓冲区的指针此值可以在IRD的SQL_DESC_NAME字段中读取没有column名或无法查看column名时驱动程序返回空字符串

ColumnName为null时NameLengthPtr返回可返回的总字节数（不包含空终止字符）

BufferLength

【输入】*ColumnName的长度

NameLengthPtr

【输出】返回*ColumnName可返回的总字节数（不包含空终止字符）的缓冲区的指针可返回的长度大于或等于BufferLength时*ColumnName被截断为在BufferLength减去null终止字符的长度

DataTypePtr

【输出】返回column的SQL类型的缓冲区的指针此值可以在IPD的SQL_DESC_CONCISE_TYPE字段中读取

ColumnSizePtr

【输出】返回数据源的column大小的缓冲区的指针

DecimalDigitsPtr

【输出】返回数据源的小数点位数的缓冲区的指针

NullablePtr

【输出】返回column是否允许空值的缓冲区的指针此值可以在IRD的SQL_DESC_NULLABLE字段中读取是以下值中的一个值

- SQL_NO_NULLS : column不允许空值
- SQL_NULLABLE : column允许空值
- SQL_NULLABLE_UNKNOWN : 由驱动程序来决定column是否允许空值

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
01004	String data, right truncated	*ColumnName的大小不足以返回所有column名因此column名被截断未截断的column名长度返回至*NameLengthPtr（函数返回 SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
07005	Prepared statement not a cursor-specification	命令不返回结果集因此没有可描述的column
07009	Invalid descriptor index	ColumnNumber参数值大于结果集的column数
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation failure	内存分配错误
HY010	Function sequence error	调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution变量前调用了此函数
HY090	Invalid string or buffer length	BufferLength参数值小于0

说明

应用程序通常是调用SQLPrepare后在调用相关的SQLExecute前后调用SQLDescibeCol另外应用程序在调用SQLExecDirect后可调用SQLDescribeCol

SQLDescribeParam

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLDescribeParam返回预备的SQL语句以及其相关的参数标记的描述此信息也可用于IPD的字段中

语句

```
SQLRETURN SQLDescribeParam(
    SQLHSTMT       StatementHandle,
    SQLUSMALLINT    ParameterNumber,
    SQLSMALLINT *   DataTypePtr,
    SQLULEN *      ParameterSizePtr,
    SQLSMALLINT *   DecimalDigitsPtr,
    SQLSMALLINT *   NullablePtr);
```

参数

StatementHandle

【输入】命令语句柄

ParameterNumber

【输入】从1开始递增的参数标记编号

DataTypePtr

【输出】返回参数的SQL类型的缓冲区的指针此值可以在IPD的SQL_DESC_CONCISE_TYPE记录字段中读取

ParameterSizePtr

【输出】返回对应参数标记的column或表达式大小的缓冲区指针

DecimalDigitsPtr

【输出】返回对应参数标记的column或表达式的小数点位数的缓冲区指针

NullablePtr

【输出】显示参数是否允许空值的缓冲区指针此值可以在IPD的SQL_DESC_NULLABLE字段中读取是以下值中的一个

- SQL_NO_NULLS：参数不允许空值
- SQL_NULLABLE：参数允许空值
- SQL_NULLABLE_UNKNOWN：驱动程序无法决定参数是否允许空值

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
07009	Invalid descriptor index	ParameterNumber参数值小于1 ParameterNumber参数值大于相关SQL语句的参数数
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	调用SQLPrepare或SQLExecDirect之前先调用了此函数 调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution变量之前调用了此函数

说明

驱动程序无法向预备的SQL语句提供准确的参数信息因此必需向*DataTypePtr返回SQL_VARCHAR向*ParameterSizePtr返回4000向*DecimalDigitsPtr返回0向*NullablePtr返回SQL_NULLABLE

SQLDisconnect

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLDisconnect关闭与特定连接句柄关联的连接

语句

```
SQLRETURN SQLDisconnect(
    SQLHDBC     ConnectionHandle);
```

参数

ConnectionHandle

【输入】连接句柄

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE,

SQL_STILL_EXECUTING

检测

SQLSTATE	报错	说明
08003	Connection not open	未打开ConnectionHandle参数的连接
25000	Invalid transaction state	ConnectionHandle参数的连接有正在进行中的事务事务保持激活状态
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

说明

如果应用程序使用存在未完成事务的连接句柄调用SQLDisconnect则驱动程序将返回SQLSTATE 25000 (Invalid transaction state)不变更事务连接依然为打开状态未完成的事务是未通过SQLEndTran commit或rollback的事务

如果在解除所有命令语之前应用程序调用SQLDisconnect则驱动程序将解除数据源中的连接后删除所有命令语和分配给连接句柄的描述符

SQLDriverConnect

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLDriverConnect可以替代SQLConnect相比SQLConnect的3个参数支持需要更多的连接信息的数据源

SQLDriverConnect通过由数据源名称一个以上的用户一个以上的密码以及数据源所需的其他信息组成的连接字符串设置连接

建立连接后SQLDriverConnect返回已完成的连接字符串应用程序可以使用此字符串请求后续的连接

语句

```
SQLRETURN SQLDriverConnect(
    SQLHDBC         ConnectionHandle,
    SQLHWND         WindowHandle,
    SQLCHAR *       InConnectionString,
    SQLSMALLINT     StringLength1,
    SQLCHAR *       OutConnectionString,
    SQLSMALLINT     BufferLength,
```

```
    SQLSMALLINT *   StringLength2Ptr,
    SQLUSMALLINT    DriverCompletion);
```

参数

ConnectionHandle

【输入】连接句柄

WindowHandle

【输入】窗口句柄应用程序传送父窗口的句柄或空指针空指针的情况SQLDriverConnect不显示对话框

InConnectionString

【输入】所有连接字符串部分连接字符串或空字符串

StringLength1

【输入】*InConnectionString的长度

OutConnectionString

【输出】已完成连接字符串的缓冲区指针当成功连接到目标数据源时该缓冲区包含已完成的连接字符串应用程序至少要分配1024个字符的缓冲区

OutConnectionString为空值时StringLength2Ptr返回可返回的总字符数（不包括空终止字符）

BufferLength

【输入】*OutConnectionString的长度

StringLength2Ptr

【输出】返回*OutConnectionString可返回的总字符数（不包含空终止字符）的缓冲区指针可返回的长度大于或等于BufferLength时*OutConnectionString截断为在BufferLength减去null终止字符大小

DriverCompletion

【输入】作为驱动程序是否需要更多连接信息的标志有SQL_DRIVER_PROMPTSQL_DRIVER_COMPLETESQL_DRIVER_COMPLETE_REQUIRED或SQL_DRIVER_NOPROMPT

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

检测

SQLSTATE	报错	说明
01004	String data, right truncated	*OutConnectionString缓冲区大小不足以返回整个连接字符串因此连接字符串被截断未被截断长度返回至*StringLength2Ptr（函数返回SQL_SUCCESS_WITH_INFO）
08001	Client unable to establish connection	驱动程序无法建立与数据源的连接
08002	Connection name in use	指定的ConnectionHandle已连接数据源

SQLSTATE	报错	说明
08004	Server rejected the connection	设定值达到极限时数据源拒绝建立连接
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
28000	Invalid authorization specification	连接字符串的用户标识符和认证字符串有误
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY090	Invalid string or buffer length	StringLength1 参数值小于0但不是SQL_NTS BufferLength参数值小于0
HY110	Invalid driver completion	DriverCompletion参数值不是 SQL_DRIVER_PROMPTSQL_DRIVER_COMPLETESQL_DRIVER_COMPLETE_REQUIREDSQL_DRI
HYC00	Optional feature not implemented	驱动程序不支持应用程序请求的ODBC操作
HYT00	Timeout expired	连接数据源之前登录限制时间超时限制时间通过SQLSetConnectAttr的SQL_ATTR_LOGIN_TIM

说明

连接字符串语句如下

```
connection-string ::= empty-string[;] | attribute[;] | attribute;
connection-string

empty-string ::=attribute ::= attribute-keyword=attribute-value |
DRIVER=[{]attribute-value[}]

attribute-keyword ::= DSN | PROTOCOL | CS_MODE | HOST | PORT | TCP_NODELAY
| UID | PWD | ALTERNATE_SERVERS | FAILOVER_TYPE | FAILOVER_GRANULARITY |
DATE_FORMAT | TIME_FORMAT | TIME_WITH_TIME_ZONE_FORMAT | TIMESTAMP_FORMAT
| TIMESTAMP_WITH_TIME_ZONE_FORMAT | CHAR_LENGTH_UNITS | CONN_NAME

attribute-value ::= character-string
```

character-string为0个以上的字符attribute-keyword不区分大小写attribute-value区分大小写

DSN关键字的值不只由空白构成

下表为attribute-keyword

关键字	说明
DSN	数据源名称
PROTOCOL	连接方式（DA, TCP）

关键字	说明
CS_MODE	设置以dedicated模式连接还是以shared模式连接 不设置时根据listener的 configuration(DEFAULT_CS_MODE)决定模式
HOST	主机名称或者IP地址
PORT	连接端口号
TCP_NODELAY	socket TCP_NODELAY选项
UID	用户ID
PWD	用户ID的密码没有用户ID密码时是空字符串(PWD=;)
ALTERNATE_SERVERS	发生failover时尝试连接的服务器目录各服务器以逗号（）区分 不使用failover功能时不设置ALTERNATE_SERVERS
FAILOVER_TYPE	• CONNECTION：连接失败时连接到ALTERNATE_SERVERS • SESSION：连接失败或处理statement过程中连接中断时连接ALTERNATE_SERVERS并复原statement如果断开连接时没有进行中的事务则failover后执行进行中的statement

关键字	说明
FAILOVER_GRANULARITY	• 0：failover进行过程中发生错误也继续进行failover • 1：failover进行过程中发生除SQLExeceute()SQLExecDirect()外的错误时failover将失败 • 2：failover进行过程中发生错误时failover将失败
DATE_FORMAT	DATE类型字符串
TIME_FORMAT	TIME类型字符串
TIME_WITH_TIME_ZONE_FORMAT	TIME WITH TIME ZONE类型字符串
TIMESTAMP_FORMAT	TIMESTAMP类型字符串
TIMESTAMP_WITH_TIME_ZONE_FORMAT	TIMESTAMP WITH TIME ZONE类型字符串
CHAR_LENGTH_UNITS	SQLBindParameter()中ParameterType为SQL_CHARSQL_VARCHAR时的ColumnSize的单位 • BYTE, OCTETS：以byte为单位 • CHAR, CHARACTERS：以字符为单位
CONN_NAME	XA中使用的Connection Name此处指定的名称仅在Embedded SQL程序有效其余均被忽略

SQLEndTran

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLEndTran对与连接相关的所有命令的激活操作请求提交或回滚

语句

```
SQLRETURN SQLEndTran(
    SQLSMALLINT   HandleType,
    SQLHANDLE     Handle,
    SQLSMALLINT   CompletionType);
```

参数

HandleType

【输入】句柄标识符环境句柄的情况应为SQL_HANDLE_ENV连接句柄的情况应为SQL_HANDLE_DBC

Handle

【输入】表示事务范围的HandleType的句柄

CompletionType

【输入】SQL_COMMIT 或SQL_ROLLBACK

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE,

SQL_STILL_EXECUTING

检测

SQLSTATE	报错	说明
08003	Connection not open	HandleType为SQL_HANDLE_DBCHandle是未连接状态
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution变量前调用了此函数
HY012	Invalid transaction operation code	CompletionType参数值并非SQL_COMMIT或SQL_ROLLBACK

SQLSTATE	报错	说明
HY092	Invalid attribute/option identifier	HandleType参数值并非SQL_HANDLE_ENV或SQL_HANDLE_DBC

说明

CompletionType为SQL_COMMIT时SQLEndTran对与连接相关的所有命令的激活操作请求commitCompletionType为SQL_ROLLBACK时SQLEndTran对与连接相关的所有命令的激活请求rollback没有激活的事务时SQLEndTran不影响数据源并返回SQL_SUCCESS

驱动程序为手动commit模式（调用SQLSetConnectAttr把SQL_ATTR_AUTOCOMMIT属性设置为SQL_AUTOCOMMIT_OFF）时SQL命令语句对当前数据源执行时默认开始新的事务

提交时SQLEndTran不影响与连接相关的已打开的游标游标留在调用SQLEndTran之前指向的行

回滚时SQLEndTran关闭所有命令语句中已打开的所有游标SQLEndTran设置命令语为准备状态应用程序可以在不调用SQLPrepare的情况下调用SQLExecute

没有激活的事务时SQLEndTran返回SQL_SUCCESS

驱动程序为自动提交模式时SQLEndTran不管CompletionType总是返回SQL_SUCCESS

SQLError

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

SQLError返回错误或状态信息

SQLExecDirect

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

如果命令语中有参数时SQLExecDirect使用参数标记的当前值执行命令语句SQLExecDirect是用于仅执行一次命令的最快方式

语句

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

参数

StatementHandle

【输入】命令语句柄

StatementText

【输入】执行的SQL命令语

TextLength

【输入】*StatementText的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING,

SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

检测

SQLSTATE	报错	说明
01004	String data, right truncated	返回至输入/输出或输出变量的字符串或二进制数据被截断字符串截断的情况是右侧被截断 SQL_SUCCESS_WITH_INFO）
01S02	Option value changed	指定的命令语属性值不适合执行操作临时被替代为相近值（可通过调用SQLGetStmtAttr查看 的值）此替代值有效至关闭游标关闭游标后重新变更为原来的值 可变更的命令语属性如下 SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ ATTR_QUERY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR （函数返回 SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
07006	Restricted data type attribute violation	SQLBindParameter的ValueType参数识别的数据值不能转换为SQLBindParameter的Paramet 数识别的数据类型 返回至SQL_PARAM_INPUT_OUTPUT或SQL_PARAM_OUTPUT参数的数据值不能转换为识别为 SQLBindParameter的ValueType参数的数据类型 （如果成功返回一个以上的行函数则返回SQL_SUCCESS_WITH_INFO）
07007	Restricted parameter value violation	参数类型为SQL_PARAM_INPUT_OUTPUTSQLBindParameter的*StrLen_or_IndPtr不是 SQL_NULL_DATASQL_DEFAULT_PARAMSQL_LEN_DATA_AT_EXEC(len)或SQL_DATA_AT_EXE
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
22001	String data, right truncation	字符串或二进制数据被截断
22002	Indicator variable required but not supplied	空数据被绑定到SQLBindParameter的StrLen_or_IndPtr空指针输出参数

SQLSTATE	报错	说明
24000	Invalid cursor state	通过SQLFetchSQLFetchScroll使游标位于StatementHandle 游标为打开状态但不位于StatementHandle *StatementText为positioned update或delete命令语游标位于结果集开始前或结束后
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY009	Invalid use of null pointer	*StatementText为空指针
HY010	Function sequence error	调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution 变量之 此函数
HY090	Invalid string or buffer length	TextLength参数值小于0但不是SQL_NTS SQLBindParameter设置的参数值为空指针参数长度为0，不是小于 SQL_NULL_DATASQL_DATA_AT_EXECSQL_DEFAULT_PARAMSQL_LEN_DATA_AT_EXEC_OFFS SQLBindParameter设置的参数值不是空指针C数据类型为SQL_C_BINARY 或SQL_C_CHAR参数 0但不是小于 SQL_NTSSQL_NULL_DATASQL_DATA_AT_EXECSQL_DEFAULT_PARAMSQL_LEN_DATA_AT_EX 的值

SQLSTATE	报错	说明
HYT00	Timeout expired	从数据源获取结果集前查询限制时间超时限制时间通过SQLSetStmtAttr的SQL_ATTR_QUERY 设置

说明

应用程序调用SQLExecDirect将SQL命令语句传送给数据源

应用程序可以在SQL命令中包含一个以上的参数标记字符为了包含参数标记应用程序需要在SQL命令的恰当位置包含一个问号（？）

SQL命令为SELECT语句应用程序用SQLSetCursorName连接游标时驱动程序使用指定的游标应用程序不连接命令和游标时驱动程序生成游标名

数据源为手动提交模式并事务尚未开始时驱动程序在传送SQL命令之前开始事务

SQLExecDirect发现data-at-execution参数时返回SQL_NEED_DATA应用程序使用SQLParamData和SQLPutData传送数据

SQLExecDirect已执行带有检索条件的更新插入删除命令但数据源中没有变更的记录时SQLExecDirect调用将返回SQL_NO_DATA

SQL_ATTR_PARAMSET_SIZE命令属性值大于1并且SQL命令中至少包含一个参数标记字符时SQLExecDirect将对SQLBindParameter的ParameterValuePtr参数指向的数组中的参数集合各执行一次SQL命令

SQLExecute

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

命令语中有参数时SQLExecute使用变量标记的当前值执行命令

语句

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

参数

StatementHandle

【输入】命令语句柄

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING,

SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

检测

SQLSTATE	报错	说明
01004	String data, right truncated	返回给输入/输出或输出变量的字符串二进制数据被截断字符串的右侧被截断（函数返回SQL_SUCCESS_WITH_INFO）
01S02	Option value changed	指定的命令属性值不适合执行操作临时被替代为相近值（通过调用SQLGetStmtAttr可查看临值）此替代值有效至关闭游标关闭游标后重新变更为原来的值 可变更的命令属性如下 SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR （函数返回 SQL_SUCCESS_WITH_INFO）
07006	Restricted data type attribute violation	SQLBindParameter的ValueType参数识别的数据不能转换为SQLBindParameter的ParameterT识别的数据类型 返回至SQL_PARAM_INPUT_OUTPUT或SQL_PARAM_OUTPUT参数的数据不能转换为SQLBind的ValueType参数识别的数据类型 （如果成功返回一个以上的行函数则返回SQL_SUCCESS_WITH_INFO）
07007	Restricted parameter value violation	参数类型为SQL_PARAM_INPUT_OUTPUTSQLBindParameter的*StrLen_or_IndPtr不是SQL_NULL_DATASQL_DEFAULT_PARAMSQL_LEN_DATA_AT_EXEC(len)或SQL_DATA_AT_EXEC

SQLSTATE	报错	说明
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
22001	String data, right truncation	字符串或二进制数据被截断
22002	Indicator variable required but not supplied	空数据被绑定到SQLBindParameter的StrLen_or_IndPtr空指针输出参数
24000	Invalid cursor state	通过SQLFetchSQLFetchScroll使游标位于StatementHandle 游标为打开状态但不位于StatementHandle *StatementText为positioned update或delete命令游标位于结果集开始前或结束后
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution变量之前 函数 未准备好的StatementHandle

SQLSTATE	报错	说明
HY090	Invalid string or buffer length	TextLength 参数值小于0但不是SQL_NTS SQLBindParameter设置的参数值为空指针参数长度为0不小于SQL_NULL_DATASQL_DATA_AT_EXECSQL_DEFAULT_PARAMSQL_LEN_DATA_AT_EXEC_OFFS SQLBindParameter设置的参数值不为空指针C数据类型为SQL_C_BINARY或SQL_C_CHAR参数但不小于SQL_NTSSQL_NULL_DATASQL_DATA_AT_EXECSQL_DEFAULT_PARAMSQL_LEN_DATA_AT_EX的值
HYT00	Timeout expired	从数据源获取结果集前查询限制时间超时限制时间可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置

说明

SQLExecute执行SQLPrepare准备的命令应用程序抛弃SQLExecute调用的结果后使用新的参数值重新调用SQLExecute

如果要执行一次以上的SELECT命令应用程序再次执行SELECT命令之前必须重新调用SQLCloseCursor

数据源为手动提交模式并事务还未开始时驱动程序在传送SQL命令之前开始事务

SQLExecute发现data-at-execution参数时返回SQL_NEED_DATA应用程序通过SQLParamData和SQLPutData传送数据

SQLExecute已执行带有检索条件的更新插入删除命令但数据源中没有变更的记录时SQLExecute

调用将返回SQL_NO_DATA

SQL_ATTR_PARAMSET_SIZE命令属性值大于1并且命令语至少包含一个参数标记字符时SQLExecute将对SQLBindParameter的ParameterValuePtr参数指向的数组中的参数集合各执行一次SQL命令

SQLExtendedFetch

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

SQLExtendedFetch从结果集中获取指定的数据集并返回给绑定的所有column

语句

```
SQLRETURN SQLExtendedFetch(
    SQLHSTMT         StatementHandle,
    SQLUSMALLINT     FetchOrientation,
    SQLLEN           FetchOffset,
    SQLULEN *        RowCountPtr,
    SQLUSMALLINT *   RowStatusArray);
```

SQLFetch

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLFetch从结果集里获取下一个行集并返回给绑定的所有column

语句

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

参数

StatementHandle

【输入】命令语句柄

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01004	String data, right truncated	返回给column的字符串或二进制数据被截断字符串的右侧被截断
01S07	Fractional truncation	返回的column的数据被截断数字数据类型时小数部分被截断包含期间构成要素的timetimestampinterval数据类型时时间的小数部分被截断（函数返回 SQL_SUCCESS_WITH_INFO）
07006	Restricted data type attribute violation	结果集的column数据值不能转换为SQLBindCol的TargetType指定的数据类型
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
22002	Indicator variable required but not supplied	SQLBindCol的StrLen_or_IndPtr(或SQLSetDescFieldSQLSetDescRec设置的SQL_DESC_INDICATOR_PTR)向空指针的列赋予空值
22003	Numeric value out of range	一个以上的column中返回数值的整数部分（非小数部分）被截断
22007	Invalid datetime format	结果集的字符串为无效的datetimetimestamp类型

SQLSTATE	报错	说明
22012	Division by zero	返回算术表达式除以0的结果
22015	Interval field overflow	将exact numeric或interval SQL数据类型指定为interval C类型时损失先行字段的有效数字 SQL类型的值不能表示为interval C类型
22018	Invalid character value for cast specification	结果集的字符column包含不能以C缓冲区的字符集表示的字符 C类型为exact或approximate numericdatetimeinterval 数据类型SQL类型为字符数据类型时绑定到C类型的column值不是有效的字符
24000	Invalid cursor state	StatementHandle为已执行状态但没有与StatementHandle关联的结果集
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	指定的StatementHandle不在执行状态中未调用SQLExecDirectSQLExecute目录函数的情况下调用了此函数 调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution 变量之前调用了此函数

SQLSTATE	报错	说明
HYT00	Timeout expired	从数据源接收结果集之前查询限制时间超时限制时间通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置

说明

SQLFetch返回结果集的下一个数据集合生成结果集并在关闭游标之前即只能在结果集存在期间调用SQLFetch有绑定的column时向column返回数据应用程序指定返回行状态数组的指针或获取的行数的缓冲区时SQLFetch也返回此信息可以混合调用SQLFetch与SQLFetchScroll

游标的位置

生成结果集后游标位于结果集的开始前SQLFetch获取下一行集合这与FetchOrientation调用设置为SQL_FETCH_NEXT的SQLFetchScroll相同

SQL_ATTR_ROW_ARRAY_SIZE命令属性指定行集合的行数如果SQLFetch获取的行集合与结果集的最后重叠则SQLFetch返回部分行集合假设S为获取的行集合的开始行R为行集合的大小L为结果集的最后一行时S + R - 1大于L仅有行集合的第一个L - S + 1个行为有效其他行均为空变为SQL_ROW_NOROW 状态

返回SQLFetch后当前行为行集合的第一行

以下表格中根据第二个表格的条件调用SQLFetch后说明游标位置

状态	新的行集合的第一行
Before start	1

状态	新的行集合的第一行
CurrRowsetStart <= LastResultRow – RowsetSize[1]	CurrRowsetStart + RowsetSize[2]
CurrRowsetStart > LastResultRow - RowsetSize[1]	After end
After end	After end

[1] fetch过程中变更行集合的大小时行集合的大小为该fetch之前使用的行集合大小

[2] fetch过程中变更行集合的大小时行集合的大小为新的fetch中使用的行集合大小

标记法	含义
Before start	block游标位于结果集合的开始之前新的行集合的第一行位于结果集合的开始之前时SQLFetch返回SQL_NO_DATA
After end	block游标位于结果集合的结束之后新的行集合的第一行位于结果集合的结束之后时SQLFetch返回SQL_NO_DATA
CurrRowsetStart	当前行集合的第一行编号
LastResultRow	结果集的最后一行编号
RowsetSize	行集合大小

以下为结果集中有100个行每行集合的大小为5的示例以下表格为对不同起始位置的SQLFetch返回的行集合和返回码

当前行集合	返回码	新的行集合	获取的行数
Before start	SQL_SUCCESS	1 to 5	5
1 to 5	SQL_SUCCESS	6 to 10	5
52 to 56	SQL_SUCCESS	57 to 61	5
91 to 95	SQL_SUCCESS	96 to 100	5
93 to 97	SQL_SUCCESS	98 to 100 行状态数组的行4和行5设置为 SQL_ROW_NOROW	3
96 to 100	SQL_NO_DATA	无	0
99 to 100	SQL_NO_DATA	无	0
After end	SQL_NO_DATA	无	0

在绑定的列返回数据

像SQLFetch返回各行一样向绑定到列的缓冲区中的各个绑定列输入数据如果没有绑定的列SQLFetch不返回任何数据也不向前移动游标可以通过SQLGetData继续获取数据如果游标为多重行游标（SQL_ATTR_ROW_ARRAY_SIZE大于1时）则可在SQLGetInfo的InfoType设置为SQL_FETDATA_EXTENSIONS和返回SQL_GD_BLOCK时调用SQLGetData

详细内容参考 **SQLGetData**

SQLFetch对行的各绑定column可以执行以下操作

1. 数据为空时将长度/指示符缓冲区设置为SQL_NULL_DATA后操作下一个列如果数据为NULL并未绑定长度/指示符缓冲区时SQLFetch对行和下一个要移动的行返回SQLSTATE 22002 (Indicator variable required but not supplied)决定长度/指示符地址的详细内容参考SQLBindCol的缓冲区地址column的数据不是空时SQLFetch执行第二个步骤
2. 将SQL_ATTR_MAX_LENGTH命令属性设置为非0的值column包含字符或二进制数据时数据被截断为SQL_ATTR_MAX_LENGTH字节长度

> Note:
>
> SQL_ATTR_MAX_LENGTH命令属性用于减少网络消耗通常体现在数据源在网络返回数据之前截断数据驱动程序与数据源不要求支持此功能因此为了保证数据按照特定长度被截断应用程序需要在SQLBindCol的cbValueMax参数中定义其大小并生成缓冲区

3. 按照SQLBindCol的TargetType指定的类型转换数据
4. 数据被转换为字符或二进制等可变长度数据类型时SQLFetch查看数据的长度是否超过数据缓冲区的长度（包含NULL终止字符的）字符数据的长度超过数据缓冲区的长度时SQLFetch把数据截断为小于NULL终止字符长度的数据缓冲区长度此时数据以NULL终止如果二进制数据的长度超过数据缓冲区长度SQLFetch将其截断为数据缓冲区长度数据缓冲区长度指定在SQLBindCol的BufferLengthSQLFetch绝对不截断被转换为固定长度类型的数据因为数据缓冲区的长度始终与数据类型的长度一致
5. 在数据缓冲区存储转换的（可能的话为被截断的）数据决定数据缓冲区地址的详细内容参考SQLBindCol的缓冲区地址
6. 在长度/指示符缓冲区中存储数据的长度如果指示符指针与长度指针均设置于同一个缓冲区（调用SQLBindCol）则缓冲区中记录有效数据的长度空值时在缓冲区记录SQL_NULL_DATA如果未绑定长度/指示符缓冲区则SQLFetch不返回长度

- 字符或二进制数据的情况由于缓冲区大小太小因此其为转换数据后截断之前的数据长度如

果是非常长的数据驱动程序在转换后无法确定数据长度时将其长度设置为SQL_NO_TOTAL 如果数据被SQL_ATTR_MAX_LENGTH命令属性截断则在长度/指示符缓冲区中记录此属性值而不是实际长度由于此属性被设置为转换前在服务器上直接截断数据因此驱动程序无法计算数据的实际长度

- 其他数据类型的情况其为转换后的数据长度

7. 转换数据的过程中如果没有有效数据的流失（例如实数1.234被截断为1）SQLFetch返回SQLSTATE 01S07 (Fractional truncation)与SQL_SUCCESS_WITH_INFO数据缓冲区的长度过小而被截断时（例如字符串"abcdef"记录于4字节的缓冲区时）SQLFetch返回SQLSTATE 01004 (Data truncated)与SQL_SUCCESS_WITH_INFO数据被SQL_ATTR_MAX_LENGTH命令属性截断时SQLFetch返回SQL_SUCCESS不返回SQLSTATE 01S07 (Fractional truncation)或SQLSTATE 01004(Data truncated)转换数据时如果有效数据流失（例如超过100,000的SQL_INTEGER值转换为SQL_C_TINYINT时）则SQLFetch返回SQLSTATE 22003 (Numeric value out of range)与 (行集合大小为1时)SQL_ERROR或(行集合大小大于1时)SQL_SUCCESS_WITH_INFO

如果不反悔SQLFetchSQLFetchScroll的SQL_SUCCESS或SQL_SUCCESS_WITH_INFO则不定义绑定数据的缓冲区及长度/指示符缓冲区的内容

行状态数组

行状态数组用于返回行集合的各个状态该数组地址指定为SQL_ATTR_ROW_STATUS_PTR命令属性应用程序需分配与SQL_ATTR_ROW_ARRAY_SIZE命令属性指定的数量相同的要素此值由SQLFetchSQLFetchScrollSQLBulkOperations或SQLSetPos设置SQL_ATTR_ROW_STATUS_PTR命令属性值为空指针时此函数不返回行的状态

SQLFetchSQLFetchScroll不返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO时不定义行状态缓冲

区的内容

以下值返回至行状态数组

行状态数组值	说明
SQL_ROW_SUCCESS	成功获取行在结果集最后一次获取后未再变更
SQL_ROW_SUCCESS_WITH_INFO	成功获取行在结果集最后一次获取后未再变更但返回了对行的告警
SQL_ROW_ERROR	获取行时报错
SQL_ROW_UPDATED	成功获取行在结果集最后一次获取后存在变更重新获取行或重新执行SQLSetPos时状态变更为新的行状态
SQL_ROW_DELETED	结果集中获取最后一次行后被删除
SQL_ROW_NOROW	行集合与结果集的最后重叠不返回任何行

行获取缓冲区

行获取缓冲区用于返回获取的行数也包含获取数据时报错而没有数据的行即行状态数组中非SQL_ROW_NOROW值的行数SQL_ATTR_ROWS_FETCHED_PTR命令属性定义此缓冲区的地址应用程序分配此缓冲区SQLFetchSQLFetchScroll设置此缓冲区SQL_ATTR_ROWS_FETCHED_PTR命令属性值为空指针时此函数不返回获取的行数如果要查看结果集的当前行数应用程序可以以SQL_ATTR_ROW_NUMBER属性调用SQLGetStmtAttr

SQLFetchSQLFetchScroll中不返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO时不定义行读取缓冲区的内容返回SQL_NO_DATA时行读取缓冲区的值设置为0

错误处理

报错与告警可适用于个别行或整个函数

对整个函数的报错与告警

如SQLSTATE HYT00 (Timeout expired)或SQLSTATE 24000 (Invalid cursor state)等适用于整个函数的报错时SLQFetch返回SQL_ERROR与对应SQLSTATE不定义行缓冲区的内容不变更游标的位置

适用于整个函数的告警SQLFetch返回SQL_SUCCESS_WITH_INFO与对应SQLSTATE在记录适用于个别行的状态之前返回适用于整个函数的告警

对个别行的报错与告警

SQLSTATE 22012(Division by zero)等报错或SQLSTATE 01004(Data truncated)等告警适用于个别行

SQLFetch执行以下操作

- 在行状态数组的对应要素中设置报错的SQL_ROW_ERROR或告警的SQL_ROW_SUCCESS_WITH_INFO
- 增加一条以上对报错或告警包含SQLSTATE的记录
- 在状态记录中设置行与列编号字段SQLFetch无法查看行或列编号时编号分别设置为SQL_ROW_NUMBER_UNKNOWN或SQL_COLUMN_NUMBER_UNKNOWN状态记录不适用于特定列时SQLFetch将column编号设置为SQL_NO_COLUMN_NUMBER

当行集合的所有行（不包含状态为SQL_ROW_NOROW的行）报错时SQLFetch返回SQL_ERROR部

分行报错时返回SQL_SUCCESS_WITH_INFO行集合的大小为1时发生行报错时SQLFetch返回SQL_ERROR

描述符与SQLFetch

SQLFetch使用以下描述符字段

描述符字段	描述符	字段的位置	设置
SQL_DESC_ARRAY_SIZE	ARD	头部	SQL_ATTR_ROW_ARRAY_SIZE命令属性
SQL_DESC_ARRAY_STATUS_PTR	IRD	头部	SQL_ATTR_ROW_STATUS_PTR命令属性
SQL_DESC_BIND_OFFSET_PTR	ARD	头部	SQL_ATTR_ROW_BIND_OFFSET_PTR命令属性
SQL_DESC_BIND_TYPE	ARD	头部	SQL_ATTR_ROW_BIND_TYPE命令属性
SQL_DESC_COUNT	ARD	头部	SQLBindCol的ColumnNumber 参数
SQL_DESC_DATA_PTR	ARD	记录	SQLBindCol的TargetValuePtr 参数
SQL_DESC_INDICATOR_PTR	ARD	记录	SQLBindCol的StrLen_or_IndPtr 参数
SQL_DESC_OCTET_LENGTH	ARD	记录	SQLBindCol的BufferLength 参数
SQL_DESC_OCTET_LENGTH_PTR	ARD	记录	SQLBindCol的StrLen_or_IndPtr 参数
SQL_DESC_ROWS_PROCESSED_PTR	IRD	记录	SQL_ATTR_ROWS_FETCHED_PTR命令属性

描述符字段	描述符	字段的位置	设置
SQL_DESC_TYPE	ARD	记录	SQLBindCol的TargetType 参数

所有描述符字段可通过SQLSetDescField设置

长度与指示符缓冲区的分离

应用程序存储长度和指示符值时可绑定单个缓冲区或两个缓冲区应用程序调用SQLBindCol时在传达至StrLen_or_IndPtr参数的相同地址中设置ARD的SQL_DESC_OCTET_LENGTH_PTR和SQL_DESC_INDICATOR_PTR字段应用程序可通过调用SQLSetDescField或SQLSetDescRec在两个字段设置不同的地址

SQLFetch判断应用程序是否指定另外的长度及指示符缓冲区数据不为空时SQLFetch把指示符缓冲区设置为0在长度缓冲区中返回长度数据为空时SQLFetch把指示符缓冲区设置为SQL_NULL_DATA不修改长度缓冲区

SQLFetchScroll

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLFetchScroll从结果集中获取指定的数据集合向绑定的所有column返回数据

语句

```
SQLRETURN SQLFetchScroll(
    SQLHSTMT      StatementHandle,
    SQLSMALLINT   FetchOrientation,
    SQLLEN        FetchOffset);
```

参数

StatementHandle

【输入】命令语句柄

FetchOrientation

【输入】fetch类型：

SQL_FETCH_NEXTSQL_FETCH_PRIORSQL_FETCH_FIRSTSQL_FETCH_LASTSQL_FETCH_AB

SOLUTESQL_FETCH_RELATIVESQL_FETCH_BOOKMARK

FetchOffset

【输入】要获取的行数此参数的解析取决于FetchOrientation的参数值

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01004	String data, right truncated	返回至column的字符串或二进制数据被截断字符串的右侧被截断
01S07	Fractional truncation	返回至column的数据被截断对于数字数据类型截断小数部分对于包含期间构成要素的timetimestampinterval 数据类型阶段时间的小数部分（函数返回 SQL_SUCCESS_WITH_INFO）
07006	Restricted data type attribute violation	结果集的column数据值不能转换为SQLBindCol的TargetType指定的数据类型
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败

SQLSTATE	报错	说明
22002	Indicator variable required but not supplied	SQLBindCol的StrLen_or_IndPtr(或以SQLSetDescFieldSQLSetDescRec设置的SQL_DESC_INDICATOR_PTR)向空指针的列赋予空值
22003	Numeric value out of range	一个以上的column中返回的数字的整数部分（非小数部分）被截断
22007	Invalid datetime format	结果集的字符串不是有效的datetimetimestamp类型
22012	Division by zero	返回算术表达式除以0的结果
22015	Interval field overflow	将exact numeric或interval SQL数据类型指定为interval C类型时丢失先行字段的有效数字 SQL类型的值不能表示为interval C类型
22018	Invalid character value for cast specification	结果集的字符列包含了不能以C缓冲区的字符集表示的字符 C类型为exact 或approximate numericdatetimeinterval 数据类型SQL类型为字符数据类型时绑定到C类型的column值不是有效的字符

SQLSTATE	报错	说明
24000	Invalid cursor state	StatementHandle为执行的状态但没有与StatementHandle关联的结果集
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	指定的StatementHandle未执行未调用SQLExecDirectSQLExecute目录函数的情况下调用了此函数 调用SQLExecuteSQLExecDirect 后返回了SQL_NEED_DATA传送所有data-at-execution 变量之前调用了此函数
HY106	Fetch type out of range	FetchOrientation参数中指定的值无效 SQL_ATTR_CURSOR_TYPE命令属性值为SQL_CURSOR_FORWARD_ONLYFetchOrientation参数值不是SQL_FETCH_NEXT SQL_ATTR_CURSOR_SCROLLABLE命令属性值为SQL_NONSCROLLABLEFetchOrientation参数值不是SQL_FETCH_NEXT
HYT00	Timeout expired	从数据源接收结果集之前查询限制时间超时限制时间可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置

说明

SQLFetchScroll返回结果集的特定行集合行集合可以指定为绝对或相对位置或书签位置生成结果集并关闭游标之前即只能在结果集存在时调用SQLFetchScroll已绑定column时向column返回数据应用程序指定返回行状态数组的指针或获取的行数的缓冲区时SQLFetchScroll也返回此信息可以混合调用SQLFetch与SQLFetchScroll

游标的位置

生成结果集后游标位于结果集的开始之前如下表SQLFetchScroll基于FetchOrientation及FetchOffset参数定位块游标以下为决定新的行集合开始的规则

FetchOrientation	含义
SQL_FETCH_NEXT	返回下一个行集合与调用SQLFetch相同 SQLFetchScroll忽略FetchOffset 值
SQL_FETCH_PRIOR	返回上一个行集合 SQLFetchScroll忽略FetchOffset值
SQL_FETCH_RELATIVE	返回从当前行集合开始的FetchOffset的行集合
SQL_FETCH_ABSOLUTE	返回从FetchOffset开始的行集合
SQL_FETCH_FIRST	返回结果集的第一个行集合 SQLFetchScroll忽略FetchOffset值
SQL_FETCH_LAST	返回结果集的最后所有行集合 SQLFetchScroll忽略FetchOffset值

FetchOrientation	含义
SQL_FETCH_BOOKMARK	在SQL_ATTR_FETCH_BOOKMARK_PTR命令属性指定的书签中返回FetchOffset的行集合

SQL_ATTR_ROW_ARRAY_SIZE命令属性指定行集合的行数如果SQLFetchScroll获取的行集合与结果集的最后重叠则SQLFetchScroll返回部分行集合假设S为获取的行集合的开始行R为行集合的大小L为结果集的最后一行时S + R - 1大于L行集合的第一个L - S + 1个行为有效行其他行均为空并变为SQL_ROW_NOROW状态

返回SQLFetchScroll后的当前行为行集合的第一行

游标位置规则

下一节说明各FetchOrientation的正确规则此规则使用以下标记法

标记法	含义
Before start	块游标位于结果集的开始之前新的行集合的第一行为结果集的开始之前时SQLFetchScroll返回SQL_NO_DATA
After end	块游标位于结果集的结束之后新的行集合的第一行为结果集的结束之后时SQLFetchScroll返回SQL_NO_DATA
CurrRowsetStart	当前行集合的第一个行编号
LastResultRow	结果集的最后一个行编号
RowsetSize	行集合的大小
FetchOffset	FetchOffset参数的值

标记法	含义
BookmarkRow	SQL_ATTR_FETCH_BOOKMARK_PTR命令属性指定的对应书签的行

SQL_FETCH_NEXT

遵守以下规则

状态	新的行集合的第一行
Before start	1
CurrRowsetStart + RowsetSize[1] <= LastResultRow	CurrRowsetStart + RowsetSize[1]
CurrRowsetStart + RowsetSize[1] > LastResultRow	After end
After end	After end

[1] 行集合的大小在fetch过程中变更时行集合的大小为该fetch之前使用的行集合大小

SQL_FETCH_PRIOR

遵守以下规则

状态	新的行集合的第一行
Before start	Before start
CurrRowsetStart = 1	Before start
1 < CurrRowsetStart <= RowsetSize[1]	1
CurrRowsetStart > RowsetSize[1]	CurrRowsetStart – RowsetSize[1]

状态	新的行集合的第一行
After end AND LastResultRow < RowsetSize[1]	1
After end AND LastResultRow >= RowsetSize[1]	LastResultRow – RowsetSize + 1[1]

[1] 行集合的大小在fetch过程中变更时行集合的大小为新的fetch中使用的行集合大小

SQL_FETCH_RELATIVE

遵守以下规则

状态	新的行集合的第一行
(Before start AND FetchOffset > 0) OR (After end AND FetchOffset < 0)	_ [1]
BeforeStart AND FetchOffset <= 0	Before start
CurrRowsetStart = 1 AND FetchOffset < 0	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + FetchOffset < 1 AND | FetchOffset | > RowsetSize[2]	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + FetchOffset < 1 AND | FetchOffset | <= RowsetSize[2]	1
1 <= CurrRowsetStart + FetchOffset <= LastResultRow	CurrRowsetStart + FetchOffset
CurrRowsetStart + FetchOffset > LastResultRow	After end
After end AND FetchOffset >= 0	After end

[1] SQLFetchScroll与将FetchOrientation设置为SQL_FETCH_ABSOLUTE后调用的情况返回相同的行集合

[2] 在fetch过程中行集合大小发生变更时行集合的大小为新的fetch中使用的行集合的大小

SQL_FETCH_ABSOLUTE

遵守以下规则

状态	新的行集合的第一行
FetchOffset < 0 AND | FetchOffset | <= LastResultRow	LastResultRow + FetchOffset + 1
FetchOffset < 0 AND | FetchOffset | > LastResultRow AND | FetchOffset | > RowsetSize[1]	Before start
FetchOffset < 0 AND | FetchOffset | > LastResultRow AND | FetchOffset | <= RowsetSize[1]	1
FetchOffset = 0	Before start
1 <= FetchOffset <= LastResultRow	FetchOffset
FetchOffset > LastResultRow	After end

[1] 在fetch过程中行集合大小发生变更时行集合的大小为新的fetch中使用的行集合的大小

SQL_FETCH_FIRST

遵守以下规则

状态	新的行集合的第一行
Any	1

SQL_FETCH_LAST

遵守以下规则

状态	新的行集合的第一行
RowsetSize[1]<= LastResultRow	LastResultRow – RowsetSize + 1[1]
RowsetSize[1]> LastResultRow	1

[1] 在fetch过程中行集合发生变更时行集合的大小为新的fetch中使用的行集合的大小

SQL_FETCH_BOOKMARK

遵守以下规则

状态	新的行集合的第一行
BookmarkRow + FetchOffset < 1	Before start
1 <= BookmarkRow + FetchOffset <= LastResultRow	BookmarkRow + FetchOffset
BookmarkRow + FetchOffset > LastResultRow	After end

缓冲区地址

SQLFetchScroll决定与SQLFetch相同的数据的地址与长度/指示符缓冲区的地址详细内容参考SQLBindCol的**缓冲区地址**

行状态数组

SQLFetchScroll设置与SQLFetch相同的行状态数组详细内容参考SQLFetch的**行状态数组**

行获取缓冲区

SQLFetchScroll返回与SQLFetch相同的获取的行数详细内容参考SQLFetch的**行获取缓冲区**

错误处理

SQLFetchScroll返回与SQLFetch相同的报错与告警详细内容参考SQLFetch的**错误处理**

SQLForeignKeys

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLForeignKeys返回如下内容

- 指定表（参照其他表的主键指定的表的column）的外键目录
- 参照指定表的主键的其他表的外键目录

驱动程序将各目录作为结果集返回

语句

```
SQLRETURN SQLForeignKeys(
    SQLHSTMT       StatementHandle,
    SQLCHAR *      PKCatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      PKSchemaName,
    SQLSMALLINT    NameLength2,
```

```
SQLCHAR *      PKTableName,
SQLSMALLINT    NameLength3,
SQLCHAR *      FKCatalogName,
SQLSMALLINT    NameLength4,
SQLCHAR *      FKSchemaName,
SQLSMALLINT    NameLength5,
SQLCHAR *      FKTableName,
SQLSMALLINT    NameLength6);
```

参数

StatementHandle

【输入】命令语句柄

PKCatalogName

【输入】主键表目录名PKCatalogName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时PKCatalogName为不区分大小写的标识符为SQL_FALSE时PKCatalogName为区分大小写并按照字面处理的普通参数

NameLength1

【输入】*PKCatalogName的长度

PKSchemaName

【输入】主键表Schema名PKSchemaName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时PKSchemaName为不区分大小写的标识符为SQL_FALSE时PKSchemaName为区分大小写并按照字面处理的普通参数

NameLength2

【输入】*PKSchemaName的长度

PKTableName

【输入】主键表名 PKTableName不 能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时PKTableName为不区分大小写的标识符为SQL_FALSE时PKTableName为区分大小写并按照字面处理的普通参数

NameLength3

【输入】*PKTableName的长度

FKCatalogName

【输入】外键表目录名FKCatalogName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性为SQL_TRUE时FKCatalogName为不区分大小写的标识符为SQL_FALSE时FKCatalogName是区分大小写并按照字面处理的普通参数

NameLength4

【输入】*FKCatalogName的长度

FKSchemaName

【输入】外键表Schema名FKSchemaName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时FKSchemaName为不区分大小写的标识符为SQL_FALSE时FKSchemaName是区分大小写并按照字面处理的普通参数

NameLength5

【输入】*FKSchemaName的长度

FKTableName

【输入】外键表名FKTableName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时FKTableName为不区分大小写的标识符为SQL_FALSE时FKTableName是区分大小写并按照字面处理的普通参数

NameLength6

【输入】*FKTableName的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
24000	Invalid cursor state	存在调用SQLFetchSQLFetchScroll而打开的游标
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY009	Invalid use of null pointer	PKTableName与FKTableName参数为null指针 SQL_ATTR_METADATA_ID命令属性值为SQL_TRUEPKSchemaNameFKSchemaNamePKTableame或FKTableame参数为null指针
HY010	Function sequence error	调用SQLExecuteSQLExecDirect 后返回了SQL_NEED_DATA传送所有data-at-execution 变量之前调用了此函数
HY090	Invalid string or buffer length	名称长度参数中有一个值小于0但不是SQL_NTS
HYT00	Timeout expired	从数据源接收结果集之前查询限制时间超时限制时间通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置

说明

*PKTableName包含表名时SQLForeignKeys返回包含指定表的主键以及参照此主键的所有外键的结果集其他表的外键目录不包含指向指定表的unique约束条件的外键

*FKTableName包含表名时SQLForeignKeys返回包含指向其他表的主键的指定表的外键的结果集以及其参照的其他表的主键

*PKTableName和*FKTableName均包含表名时SQLForeignKeys返回参照*PKTableName指定的表的主键的*FKTableName中指定的表的外键此key应为一个

SQLForeignKeys返回标准结果集请求主键和相关外键时结果集按照FKTABLE_CATFKTABLE_SCHEMFKTABLE_NAMEKEY_SEQ顺序排列请求外键与相关的主键时结

果集按照PKTABLE_CATPKTABLE_SCHEMPKTABLE_NAMEKEY_SEQ顺序排列

下表为结果集的列

列名	列编号	数据类型	说明
PKTABLE_CAT (ODBC 1.0)	1	VARCHAR	主键表的目录名
PKTABLE_SCHEM (ODBC 1.0)	2	VARCHAR	主键表的Schema名
PKTABLE_NAME (ODBC 1.0)	3	VARCHAR not NULL	主键表的名
PKCOLUMN_NAME (ODBC 1.0)	4	VARCHAR not NULL	主键column名对没有名称的column返回空字符串
FKTABLE_CAT (ODBC 1.0)	5	VARCHAR	外键表的目录名
FKTABLE_SCHEM (ODBC 1.0)	6	VARCHAR	外键表的Schema名
FKTABLE_NAME (ODBC 1.0)	7	VARCHAR not NULL	外键表的名
FKCOLUMN_NAME (ODBC 1.0)	8	VARCHAR not NULL	外键column名对没有名称的column返回空字符

列名	列编号	数据类型	说明
KEY_SEQ (ODBC 1.0)	9	SMALLINT not NULL	从1开始的key的column序列号

列名	列编号	数据类型	说明
UPDATE_RULE (ODBC 1.0)	10	SMALLINT	SQL运算为UPDATE时适用于外键的操作（被参照的表为有主键的表参照的表为有外键的表） • SQL_CASCADE：被参照的表的主键发生变更时参照表的外键也会变更 • SQL_NO_ACTION：更新被参照的表的主键时如果参照表的行不对应被参照的表则更新被拒绝参照表的外键更新不作为参照表的主键值时此更新被拒绝 • SQL_SET_NULL：变更参照表的一个以上行的主键中的一个以上组成要素并更新时在对应主键的变更要素的参照表中把与参照外键的要素的表一致的所有行设置为NULL • SQL_SET_DEFAULT：变更参照表的一个以上行的主键中的一个以上组成要素并更新时在对应主键的变更要素的参照表中把与参照外键的要素的表一致的所有行设置为默认值

列名	列编号	数据类型	说明
DELETE_RULE (ODBC 1.0)	11	SMALLINT	SQL运算为DELETE时 适用于外键的操作（被参照的表为有主键的表参照的表为有外键的表） • SQL_CASCADE：删除被参照表的主键时也删除参照表的外键 • SQL_NO_ACTION：删除被参照表的主键时如果参照表的行不对应被参照表则此更新被拒绝 • SQL_SET_NULL：如果删除被参照表的一个以上的行则把与参照表外键的组成要素的表一致的所有行设置为NULL • SQL_SET_DEFAULT：如果删除被参照表的一个以上的行则把与参照表外键的组成要素的表一致的所有行设置为默认值
FK_NAME (ODBC 2.0)	12	VARCHAR	外键名
PK_NAME (ODBC 2.0)	13	VARCHAR	主键名

列名	列编号	数据类型	说明
DEFERRABILITY (ODBC 3.0)	14	SMALLINT	SQL_INITIALLY_DEFERRED, SQL_INITIALLY_IMMEDIATE, SQL_NOT_DEFERRABLE.

SQLFreeConnect

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 3.x中SQLFreeConnect函数被替代为SQLFreeHandle函数

详细内容参考**SQLFreeHandle**

SQLFreeEnv

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 3.x中SQLFreeEnv函数被替代为SQLFreeHandle函数

详细内容参考**SQLFreeHandle** 函数

SQLFreeHandle

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLFreeHandle释放指定的环境连接命令 描述符句柄以及相关资源

语句

```
SQLRETURN SQLFreeHandle(
    SQLSMALLINT   HandleType,
    SQLHANDLE     Handle);
```

参数

HandleType

【输入】SQLFreeHandle释放的句柄类型应为SQL_HANDLE_DBCSQL_HANDLE_DESCSQL_HANDLE_ENVSQL_HANDLE_STMT中的 一个HandleType不是这些值中的一个时SQLFreeHandle返回SQL_INVALID_HANDLE

Handle

【输入】要释放的句柄

返回

SQL_SUCCESS, SQL_ERROR, SQL_INVALID_HANDLE

SQLFreeHandle返回SQL_ERROR时句柄依然有效

检测

SQLSTATE	报错	说明
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	HandleType 参数为SQL_HANDLE_ENV至少一个连接是分配或处于连接状态调用HandleType 参数为SQL_HANDLE_ENV的SQLFreeHandle之前需调用SQLDisconnect和HandleType参数为SQL_HANDLE_DBC的SQLFreeHandle HandleType参数为SQL_HANDLE_DBC调用SQLDisconnect之前调用了函数 HandleType参数为SQL_HANDLE_STMT调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution 变量之前调用了此函数

说明

SQLFreeHandle用于释放环境连接命令语描述符句柄

释放句柄后应用程序不能使用已释放的句柄

释放环境句柄

调用HandleType为SQL_HANDLE_ENV的SQLFreeHandle之前应用程序应对环境分配的所有连接调用HandleType为SQL_HANDLE_DBC的SQLFreeHandle否则SQLFreeHandle将返回SQL_ERROR环境与激活的连接依然有效

释放连接句柄

调用HandleType为SQL_HANDLE_DBC的SQLFreeHandle之前如果句柄中有连接应用程序需要调用SQLDisconnect否则SQLFreeHandle返回SQL_ERROR连接依然有效

释放命令语句柄

HandleType为SQL_HANDLE_STMT的SQLFreeHandle调用HandleType为SQL_HANDLE_STMT的SQLAllocHandle释放已分配的所有资源应用程序使用有剩余结果的命令调用SQLFreeHandle时剩余结果将被删除应用程序释放命令语句柄时驱动程序也释放命令语以及相关的4个自动分配的描述符

SQLDisconnect自动删除连接中打开的所有命令语及描述符

SQLFreeStmt

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLFreeStmt终止指定命令语及相关处理关闭打开的游标删除剩余的结果或选择性的释放命令语句柄的相关所有资源

语句

```
SQLRETURN SQLFreeStmt(
    SQLHSTMT       StatementHandle,
    SQLUSMALLINT   Option);
```

参数

StatementHandle

【输入】命令语句柄

Option

【输入】以下选项中的一个

- SQL_ CLOSE : 关闭与StatementHandle相关的游标删除剩余的结果应用程序可以使用相

同或其他参数值重新执行SELECT语句并再次打开游标游标未打开时此选项不影响应用程序SQLCloseCursor也将关闭游标

• SQL_DROP： 不再使用此选项

• SQL_UNBIND： 释放以SQLBindCol绑定指定StatementHandle的所有column缓冲区并将ARD的SQL_DESC_COUNT字段设置为0

• SQL_RESET_PARAMS： 释放以SQLBindParameter设置指定StatementHandle的所有参数缓冲区并将APD的SQL_DESC_COUNT字段设置为0

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	调用SQLExecuteSQLExecDirect 后返回了SQL_NEED_DATA传送所有data-at-execution 变量之前调用了此函数
HY092	Option type out of range	Option 参数值不为SQL_CLOSESQL_DROPSQL_UNBINDSQL_RESET_PARAMS

说明

以SQL_CLOSE选项调用SQLFreeStmt与调用SQLCloseCursor相同但没有已打开的游标时以SQL_CLOSE选项调用SQLFreeStmt时不影响应用程序但SQLCloseCursor返回SQLSTATE 24000 (Invalid cursor state)

SQLGetConnectAttr

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLGetConnectAttr返回连接属性的当前设置

语句

```
SQLRETURN SQLGetConnectAttr(
    SQLHDBC        ConnectionHandle,
    SQLINTEGER     Attribute,
    SQLPOINTER     ValuePtr,
    SQLINTEGER     BufferLength,
    SQLINTEGER *   StringLengthPtr);
```

参数

ConnectionHandle

【输入】连接句柄

Attribute

【输入】要检索的属性

ValuePtr

【输出】返回指定为Attribute的属性的当前设定值的内存的指针

ValuePtr为null时StringLenthPtr返回可返回的所有字节数（除null终止字符）

BufferLength

【输入】Attribute为ODBC中定义的字段ValuePtr指向字符串或二进制缓冲区时该参数应为*ValuePtr的长度Attribute为ODBC中定义的字段*ValuePtr为整数时忽略此参数

StringLengthPtr

【输出】返回*ValuePtr中可返回的总字节数（不包括字符数据的空终止字符）的指针

对于字符数据可返回的字节数大于或等于BufferLength时*ValuePtr被截断为BufferLength减去1的长度驱动程序null终止

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01004	String data, right truncated	*在BufferLength中截取null终止字符大小后的数据返回至*ValuePtr未被截断的字符串的长度返回至*StringLengthPtr（函数返回 SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
08003	Connection not open	指定了连接状态可要求的Attribute值
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY090	Invalid string or buffer length	*ValuePtr为字符串BufferLength小于0但不是SQL_NTS
HY092	Invalid attribute/option identifier	Attribute参数值无效
HYC00	Optional feature not implemented	Attribute参数值有效但驱动程序不支持

说明

Attribute指定返回字符串的属性时ValuePtr应有字符串缓冲区的指针包含null终止字符返回的字符串的最大长度为BufferLength字节

Attribute	ValuePtr内容
SQL_ATTR_ACCESS_MODE (ODBC 1.0)	SQLUINTEGER值SQL_MODE_READ_ONLY用作表示不请求更新操作的连接的指示符此模式可用于对驱动程序或数据源的锁定计划事务管理优化 默认值为SQL_MODE_READ_WRITE
SQL_ATTR_AUTOCOMMIT (ODBC 1.0)	指定自动提交或手动提交模式的SQLUINTEGER值 • SQL_AUTOCOMMIT_ON：该值为默认值驱动程序使用自动提交模式各命令执行命令后立即提交从手动模式变更为自动模式时将SQL_ATTR_AUTOCOMMIT设置为SQL_AUTOCOMMIT_ON时提交连接中打开的事务 • SQL_AUTOCOMMIT_OFF：驱动程序使用手动提交模式应用程序要指定以SQLEndTrans提交或回滚
SQL_ATTR_CHARACTER_SET	驱动程序的字符集字符串
SQL_ATTR_DATABASE_CHARACTER_SET	数据源的字符集字符串

Attribute	ValuePtr内容
SQL_ATTR_DATE_FORMAT	驱动程序的DATE类型字符串
SQL_ATTR_LOGIN_TIMEOUT (ODBC 1.0)	以SQLUINTEGER值等待登录请求的时间（秒）ValuePtr为0时没有超时尝试连接无限等待

Attribute	ValuePtr内容
SQL_ATTR_METADATA_ID (ODBC 3.0)	决定目录函数的字符串参数的SQLUINTEGER值 SQL_TRUE时目录函数的字符串参数被识别为标识符不区分大小写对于不以分隔符区分的字符串驱动程序将删除前后空格字符串由大写组成对于以分隔符区分的字符串将删除前后空格分隔字符之间使用原有的字符这样的参数中有一个被设置为空指针时函数返回SQL_ERROR及SQLSTATE HY009 (Invalid use of null pointer) SQL_FALSE时目录函数的字符串参数不识别为标识符根据字符串参数可包含或不包含字符串检索模式 默认值为SQL_FALSE SQL_ATTR_METADATA_ID也可在命令语level中设置
SQL_ATTR_TIMESTAMP_FORMAT	驱动程序的TIMESTAMP类型字符串
SQL_ATTR_TIMESTAMP_WITH_TIMEZONE_FORMAT	驱动程序的TIMESTAMP WITHI TIME ZONE类型字符串

Attribute	ValuePtr内容
SQL_ATTR_TIMEZONE	驱动程序的时间段类型字符串
SQL_ATTR_TIME_FORMAT	驱动程序的TIME类型字符串
SQL_ATTR_TIME_WITH_TIMEZONE_FORMAT	驱动程序的TIME WITH TIME ZONE类型字符串
SQL_ATTR_TXN_ISOLATION (ODBC 1.0)	设置对当前连接的isolation level的32bit mask

SQLGetConnectOption

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 3.x中SQLGetConnectOption 函数被替代为SQLGetConnectAttr 函数

详细内容参考**SQLGetConnectAttr** 函数

SQLGetCursorName

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLGetCursorName返回指定的命令语以及相关游标名

语句

```
SQLRETURN SQLGetCursorName(
    SQLHSTMT        StatementHandle,
    SQLCHAR *       CursorName,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   NameLengthPtr);
```

参数

StatementHandle

【输入】命令语句柄

CursorName

【输出】返回游标名称的缓冲区指针

CursorName为空时NameLengthPtr返回可返回的字节长度（除NULL终止字符）

BufferLength

【输入】*CursorName的长度

NameLengthPtr

【输出】可返回至*CursorName的总字节长度（除空终止字符）的内存指针可返回的字节数大于或等于时*sCursorName被截断为BufferLength减去1的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01004	String data, right truncated	*CursorName 缓冲区大小不足以返回整个游标名因此游标名被截断未被截断的游标名的长度返回至*NameLengthPtr（函数返回SQL_SUCCESS_WITH_INFO）
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY010	Function sequence error	调用SQLExecuteSQLExecDirect 后返回了SQL_NEED_DATA传送所有data-at-execution 变量之前调用了此函数
HY090	Invalid string or buffer length	BufferLength 参数值小于0

说明

游标名仅用于positioned update和positioned delete语句（例：UPDATE table-name ...WHERE CURRENT OF cursor-name）应用程序中不以SQLSetCursorName设置游标名时驱动程序生成以SQL_CUR开头的游标名

命令语处于分配或准备状态时SQLGetCursorName可以重置游标名

指定或默认设置的游标名有效至相关命令调用HandleType为SQL_HANDLE_STMT的SQLFreeHandle删除为止

SQLGetData

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLGetData在结果集中检索一个column数据为了检索可变长度数据可多次调用

语句

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

参数

StatementHandle

【输入】命令语句柄

Col_or_Param_Num

【输入】用于在返回的数据中检索column数据的column编号结果集的column从1开始顺序递增

TargetType

【输入】*TargetValuePtr 缓冲区的C数据类型标识符

TargetType为SQL_ARD_TYPE时驱动程序使用ARD的SQL_DESC_CONCISE_TYPE字段指定的类型标识符

TargetValuePtr

【输出】返回数据的缓冲区的指针

TargetValuePtr不能为空

BufferLength

【输入】*TargetValuePtr 缓冲区的字节长度

驱动程序返回字符或二进制数据等可变长度数据时为了避免溢出*TargetValuePtr缓冲区而使用BufferLength当向*TargetValuePtr返回字符数据时应注意驱动程序也计算null终止字符因此*TargetValuePtr应包含null终止字符的空间否则驱动程序将截断数据

驱动程序返回整数或日期等固定长度数据时认为缓冲区足以存储数据而忽略BufferLength因此应用程序要给固定长度数据分配足够大的缓冲区否则驱动程序将截断数据

BufferLength小于0时SQLGetData返回SQLSTATE HY090 (Invalid string or buffer length)

StrLen_or_IndPtr

【输出】返回长度或指示符值的缓冲区指针此参数为空指针时不返回长度及指示符值获取空值时报错

SQLGetData可向长度/指示符缓冲区返回可返回的数据长度和

SQL_NO_TOTALSQL_NULL_DATA

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01004	String data, right truncated	用Col_or_Param_Num指定的column的所有数据无法调用单一函数进行检索调用SQL_NO_TOTAL或SQLGetData之前向*StrLen_or_IndPtr返回指定column中剩余的数据长度（函数返回 SQL_SUCCESS_WITH_INFO）
01S07	Fractional truncation	对一个以上的column返回的数据被截断对于数字数据类型截断小数部分对于包含期间构成要素的timetimestampinterval数据类型截断时间的小数部分（函数返回SQL_SUCCESS_WITH_INFO）
07006	Restricted data type attribute violation	结果集的column数据值不能转换为TargetType参数指定的C数据类型
07009	Invalid descriptor index	Col_or_Param_Num参数值大于结果集的column数

SQLSTATE	报错	说明
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
22002	Indicator variable required but not supplied	StrLen_or_IndPtr为空指针返回的数据为空
22003	Numeric value out of range	一个以上的column返回的数字值的整数部分（不是小数部分）被截断
22007	Invalid datetime format	结果集的字符串为无效的datetimetimestamp类型
22012	Division by zero	返回算术表达式除以0的结果
22015	Interval field overflow	exact numeric或interval SQL 数据类型中指定为interval C类型时丢失先行字段的有效数字 SQL类型的值不能表示在interval C类型

SQLSTATE	报错	说明
22018	Invalid character value for cast specification	结果集的字符column包含了不能以C缓冲区的字符集表示的字符 C类型为exact或approximate numericdatetimeinterval数据类型SQL类型为字符数据类型时绑定到C类型的字段值不是有效的字符
24000	Invalid cursor state	没有调用SQLFetch或SQLFetchScroll调用了函数 StatementHandle为执行的状态但没有与StatementHandle相关的结果集 调用SQLFetch或SQLFetchScroll打开了游标但游标指向结果集开始前或结束后
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY003	Program type out of range	TargetType参数值无效
HY009	Invalid use of null pointer	TargetValuePtr参数为null指针

SQLSTATE	报错	说明
HY010	Function sequence error	指定的StatementHandle为非执行状态未调用SQLExecDirectSQLExecute目录函数的情况下调用了此函数 调用SQLExecuteSQLExecDirect 后返回SQL_NEED_DATA传送所有data-at-execution 变量之前调用了此函数
HY090	Invalid string or buffer length	BufferLength 参数值小于0

说明

SQLGetData返回指定column的数据只能在SQLFetch或SQLFetchScroll的结果集中获取一个以上的行后调用如果由于应用程序限制可变长度数据太大而无法仅通过调用一次SQLGetData返回时SQLGetData可以执行部分搜索

SQLGetData的使用

TargetType参数为interval数据类型时默认值interval leading precision (2)interval seconds precision(6)各设置于ARD的SQL_DESC_DATETIME_INTERVAL_PRECISION和SQL_DESC_PRECISION字段中TargetType参数为SQL_C_NUMERIC时默认值precision(38)scale(0)各设置于ARD的SQL_DESC_PRECISION和SQL_DESC_SCALE字段中如果默认precision及scale不合适时应用程序应调用SQLSetDescField或SQLSetDescRec明确设置对应的描述符字段

可变长度数据的部分检索

SQLGetData可以部分检索SQL数据类型为

SQL_CHARSQL_VARCHARSQL_LONGVARCHARSQL_WCHARSQL_WVARCHARSQL_WLONGVARCHARSQL_BINARYSQL_VARBINARYSQL_LONGVARBINARY的可变长度数据部分

部分检索column的数据时应用程序对同一个column连续调用多次SQLGetData每次调用SQLGetData将返回数据的下一个部分应用程序应注意去掉字符数据中间的null终止字符并重新组合扔有可返回的数据时SQLGetData返回SQL_SUCCESS_WITH_INFO和SQLSTATE 01004 (Data truncated)返回数据的最后部分时返回SQL_SUCCESS

SQLGetData不能用于返回部分固定长度数据对包含固定长度数据的column调用一次以上的SQLGetData时首次调用以后返回SQL_NO_DATA

用SQLGetData检索数据

为了返回指定column的数据SQLGetData执行以下操作

1. 如果column已返回所有数据则返回SQL_NO_DATA
2. 数据为空时*StrLen_or_IndPtr设置为SQL_NULL_DATA数据为空*StrLen_or_IndPtr为空指针时SQLGetData返回SQLSTATE 22002 (Indicator variable required but not supplied) column的数据并非null时SQLGetData执行第3步骤
3. 如果SQL_ATTR_MAX_LENGTH命令属性设置为非零值或者列包含字符或二进制数据且之前从未对列调用过SQLGetData则数据被截断为SQL_ATTR_MAX_LENGTH字节长度

> Note:
>
> SQL_ATTR_MAX_LENGTH命令属性用于减少网络消耗通常由数据源实现在从网络返回数据之前截断数据驱动程序和数据源不要求支持此功能因此为了保证数据按照特定长度被截断应用程序应分配对应大小的缓冲区并指定BufferLength参数的大小

4. 把数据转换为TargetType指定的类型向数据赋予符合数据类型的默认精度与范围值

   TargetType为SQL_ARD_TYPE时使用ARD的SQL_DESC_CONCISE_TYPE字段的数据类型数据根据SQL_DESC_CONCISE_TYPE的数据类型提供ARD的SQL_DESC_DATETIME_INTERVAL_PRECISIONSQL_DESC_PRECISION以及SQL_DESC_SCALE字段的精度和范围所有的默认精度或范围都不合适时应用程序应调用SQLSetDescField或SQLSetDescRec指定设置恰当的描述符字段

5. 数据转换为字符或二进制等可变长度数据类型时SQLGetData检查数据的长度是否超过了BufferLength（包含null终止字符）字符数据的长度超过BufferLength时SQLGetData将截断除null终止字符外的BufferLength的数据二进制数据长度超过数据缓冲区长度时SQLGetData将截断为 BufferLength字节长度

   所提供的数据缓冲区中无法存储null终止字符时SQLGetData返回SQL_SUCCESS_WITH_INFO和SQLSTATE 01004

   SQLGetData不截断转换为固定长度数据类型的数据此时*TargetValuePtr的长度始终视为数据类型的大小

6. 在*TargetValuePtr存储转换的数据注意SQLGetData无法返回out of line的数据

7. 在*StrLen_or_IndPtr存储数据的长度StrLen_or_IndPtr为空指针时SQLGetData不返回长度

   - 对于字符或二进制数据此长度为转换以后并截断为BufferLength之前的长度对于很长的数据驱动程序无法查看转换之后的数据长度时返回SQL_SUCCESS_WITH_INFO并将长度设置为SQL_NO_TOTAL（SQLGetData的最后一次调用必需返回数据长度为非0或SQL_NO_TOTAL的长度值）如果数据被SQL_ATTR_MAX_LENGTH命令属性截断则属性值存储于*StrLen_or_IndPtr此属性在转换前从服务器上直接传输数据因此驱动程序无法查看数据的实际长度其是对相同column连续多次调用SQLGetData时当前调用的开始可使用的数据长度即长度随着后续每次调用将减少

   -

对于其他所有数据类型此长度为转换后的数据长度即数据转换的类型的大小

8. 转换数据的过程中在不丢失默认值的情况下数据被截断（例如转换实数1.234时被截断为整数1）或由于BufferLength过小而被截断时（例如"abcdef"存储于4字节长度的缓冲区）SQLGetData返回SQLSTATE 01004 (Data truncated)和SQL_SUCCESS_WITH_INFO由于SQL_ATTR_MAX_LENGTH命令属性截断数据时未丢失默认值则SQLGetData返回SQL_SUCCESS不返回SQLSTATE 01004 (Data truncated)

SQLGetData不返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO时（对绑定的column调用SQLGetData时）不定义绑定的数据缓冲区内容与长度/指示符缓冲区

连续调用SQLGetData时从被请求的最后一列开始检索数据之前的offset失效

例如按照以下顺序执行时

```
SQLGetData(icol=n), SQLGetData(icol=m), SQLGetData(icol=n)
```

第二次调用SQLGetData(icol=n)从n列开始检索数据由于之前的SQLGetData调用数据的所有offset不再有效

SQLGetData和描述符

SQLGetData不直接与任何描述符字段进行相互作用

TargetType为SQL_ARD_TYPE时使用ARD的SQL_DESC_CONCISE_TYPE字段的数据类型

TargetType为SQL_ARD_TYPE或SQL_C_DEFAULT时根据SQL_DESC_CONCISE_TYPE字段的数据类型指定ARD的SQL_DESC_DATETIME_INTERVAL_PRECISIONSQL_DESC_PRECISION以及SQL_DESC_SCALE字段的precision和scale

SQLGetDescField

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLGetDescField返回描述符记录的单个字段的当前设置或值

语句

```
SQLRETURN SQLGetDescField(
    SQLHDESC        DescriptorHandle,
    SQLSMALLINT     RecNumber,
    SQLSMALLINT     FieldIdentifier,
    SQLPOINTER      ValuePtr,
    SQLINTEGER      BufferLength,
    SQLINTEGER *    StringLengthPtr);
```

参数

DescriptorHandle

【输入】描述符句柄

RecNumber

【输入】应用程序要查找的信息的描述符记录FieldIdentifier参数为头部字段时忽略RecNumberRecNumber小于或等于SQL_DESC_COUNT行中不包含column或参数的数据时SQLGetDescField返回字段的默认值

FieldIdentifier

【输入】返回值的描述符字段

ValuePtr

【输出】返回描述符信息的缓冲区指针数据类型依赖于FieldIdentifier值

ValuePtr为整数类型时应用程序使用初始化为0的SQLULEN缓冲区

ValuePtr为空时StringLengthPtr返回可返回的总字节数（不包含空终止字符）

BufferLength

【输入】FieldIdentifier为ODBC定义的字段ValuePtr指向字符串或二进制缓冲区时此参数应为*ValuePtr的长度FieldIdentifier为ODBC定义的字段*ValuePtr为整数时忽略此参数

StringLengthPtr

【输出】返回*ValuePtr中可返回的总字节数（不包括字符数据的空终止字符）的指针

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE

RecNumber大于当前描述符记录数时返回SQL_NO_DATA

DescriptorHandle为IRD句柄命令语处于准备或执行状态但没有相关的游标时返回SQL_NO_DATA

检测

SQLSTATE	报错	说明
01000	General warning	各个驱动程序的信息（函数返回SQL_SUCCESS_WITH_INFO）
01004	String data, right truncated	*ValuePtr的缓冲区长度小于描述符字段值的长度而被截断向*StringLengthPtr返回剩余的描述符字段的长度（函数返回SQL_SUCCESS_WITH_INFO）
07009	Invalid descriptor index	RecNumber参数设置为0SQL_ATTR_USE_BOOKMARKS状态属性设置为SQL_UB_OFFDescriptorHandle参数为IRD句柄 FieldIdentifier参数为记录字段RecNumber参数为0DescriptorHandle参数为IPD句柄 RecNumber参数小于0
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY007	Associated statement is not prepared	DescriptorHandle与IRD句柄关联相关的状态句柄不在准备或执行阶段

SQLSTATE	报错	说明
HY010	Function sequence errord	在被调用的状态下仍然是与执行中的异步执行函数的StatementHandle相关的DescriptorHandle 调用了SQLExecuteSQLExecDirectSQLBulkOperationsSQLSetPos与返回SQL_NEED_DATA的StatementHandle相关的DescriptorHandle 对与DescriptorHandle相关的连接句柄调用了异步执行的函数调用SQLGetDescField时此异步执行函数仍在执行
HY013	Memory management error	作为参数的缓冲区大小值小于0或无法访问内存
HY021	Inconsistent descriptor information	SQL_DESC_TYPE和SQL_DESC_DATETIME_INTERVAL_CODE字段不是ODBC SQL类型特定驱动程序的（IPD的）SQL类型或（APD或ARD的）ODBC C类型的有效类型
HY090	Invalid string or buffer length	*Valueptr为字符串BufferLength小于0
HY091	Invalid descriptor field Identifier	FieldIdentifier不是ODBC中定义的字段也不是实现定义的值 FieldIdentifier未对DescriptorHandle进行定义

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考 **SQLEndTran**函数
HYT01	Connection timeout expired	数据源响应请求之前连接终止其限制时间可通过SQLSetConnectAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	与DescriptorHandle相关的驱动程序不支持函数

说明

应用程序可以为了返回描述符记录的单个字段值调用SQLGetDescFieldSQLGetDescField调用可返回包含头部字段与记录字段书签字段的所有描述符类型的所有字段设置应用程序可反复调用SQLGetDescField从相同或不同的描述符中获得任意顺序的多个字段设置为了返回驱动程序定义描述符字段可调用SQLGetDescField

由于性能方面的原因应用程序不能在执行命令前为了IRD调用SQLGetDescField

通过调用一次SQLGetDescRec可检索名称数据类型column或参数数据大小的多个字段设置可以调用SQLGetStmtAttr返回命令属性的描述符头部的单个字段设置SQLColAttribute和SQLDescribeColSQLDescribeParam返回记录或书签字段

应用程序为了检索未对特定描述符类型定义的字段值调用SQLGetDescField时函数返回

SQL_SUCCESS但不定义返回的字段的值例如对APD/ARD的SQL_DESC_NAME或SQL_DESC_NULLABLE字段调用SQLGetDescField将返回SQL_SUCCESS但不定义字段值

当应用程序为了检索已定义特定描述符类型但未设置默认值或任何其他设置的字段值而调用SQLGetDescField时函数返回SQL_SUCCESS但不定义返回的字段值

头部字段

由以下字段构成各描述符

SQL_DESC_ALLOC_TYPE[All]

只读SQLSMALLINT头部字段定义描述符被驱动程序自动分配还是被应用程序指定分配应用程序可以获得此字段但无法修改当应用程序自动分配时字段设置为SQL_DESC_ALLOC_AUTO

SQL_DESC_ARRAY_SIZE[Application descriptors]

存在于ARD并SQLULEN的此头部字段定义行集合的行数是通过调用SQLFetchSQLFetchScrolSQLBulkOperations或SQLSetPos时产生的运算返回的行数

存在于APD并SQLULEN的头部字段定义各参数值的数量

此字段的默认值为1当SQL_DESC_ARRAY_SIZE大于1时APD或ARD的SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTR和SQL_DESC_OCTET_LENGTH_PTR指数组各数组的常数与此字段的值相同

ARD的此字段可以通过与SQL_ATTR_ROW_ARRAY_SIZE属性同时调用SQLSetStmtAttr进行设置APD的字段也可以通过与SQL_ATTR_PARAMSET_SIZE属性同时调用SQLSetStmtAttr设置

SQL_DESC_ARRAY_STATUS_PTR[All]

各描述符类型的SQLUSMALLINT*的头部字段指SQLUSMALLINT值的数组这些数组包括行状态数组(IRD)参数状态数组(IPD)行运算数组(ARD)参数运算数组(APD)

IRD的该头部字段指包含调用SQLBulkOperationsSQLFetchSQLFetchScroll或SQLSetPos后的状态值的行状态数组应用程序应分配SQLUSMALLINT数组并让此字段指向数组此字段默认为空指针除驱动程序将SQL_DESC_ARRAY_STATUS_PTR字段设置为空指针外生成数组

Caution:

应用程序设置IRD的SQL_DESC_ARRAY_STATUS_PTR字段指向的行状态数组的要素时不定义驱动程序的操作

调用SQLBulkOperationsSQLFetchSQLFetchScroll或SQLSetPos时生成初始化的数组该调用不返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO时不定义此字段指向的数组的内容数组的要素可以包含以下值

- SQL_ROW_SUCCESS：成功获取行最后获取行后没有任何变化
- SQL_ROW_SUCCESS_WITH_INFO：成功获取行最后获取行后没有任何变化但对行返回告警
- SQL_ROW_ERROR：获取行的过程中报错
- SQL_ROW_UPDATED：成功获取行最后获取行后被更新如果再次获取行则状态为SQL_ROW_SUCCESS
- SQL_ROW_DELETED：最后获取后行被删除
- SQL_ROW_ADDED：SQLBulkOperations插入了行如果再次获取行则状态为SQL_ROW_SUCCESS
- SQL_ROW_NOROW：行集合与结果集的最后重叠无对应行状态数组要素返回的行

可与SQL_ATTR_ROW_STATUS_PTR属性同时调用SQLSetStmtAttr设置IRD的此字段

IRD的SQL_DESC_ARRAY_STATUS_PTR字段仅在返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO后有效如果返回码不是这两个中的一个则不定义SQL_DESC_ROWS_PROCESSED_PTR的指向

IPD的此头部字段指调用SQLExecute或SQLExecDirect后包含各参数状态信息的参数状态数组如果调用SQLExecute或SQLExecDirect后未返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO则不定义此字段指向的数组内容应用程序应分配SQLUSMALLINT数组并使此字段指向数组除了将SQL_DESC_ARRAY_STATUS_PTR字段设置为空指针外驱动程序将生成数组

数组的要素包含以下值

- SQL_PARAM_SUCCESS：对此参数集合成功执行SQL命令
- SQL_PARAM_SUCCESS_WITH_INFO：对此参数集合成功执行SQL命令但对检测数据结构体有可用的告警信息
- SQL_PARAM_ERROR：处理此参数集合时报错追加的错误信息在检测数据结构体中
- SQL_PARAM_UNUSED：处理之前的几个参数的过程中报错或由于APD的SQL_DESC_ARRAY_STATUS_PTR字段定义的数组的参数集合中设置了SQL_PARAM_IGNORE而未使用
- SQL_PARAM_DIAG_UNAVAILABLE：不能使用检测信息例如驱动程序把参数的数组识别为一个整体单元因此不生成错误信息的level

可与SQL_ATTR_PARAM_STATUS_PTR属性同时调用SQLSetStmtAttr设置IPD的该字段

在ARD中此字段指向应用程序设置的值的行操作数组以设置是否忽略SQLSetPos操作中的行数组的要素包含以下值

- SQL_ROW_PROCEED：行包含于使用SQLSetPos的批量操作（此设置不保证在该行中发生操

作如果行处于IRD行状态数组的SQL_ROW_ERROR状态则驱动程序不能对行进行操作）

- SQL_ROW_IGNORE：从使用SQLSetPos的批量操作中排除行

未设置数组的要素时批量操作包含所有行ARD的SQL_DESC_ARRAY_STATUS_PTR字段中的值为空指针时批量操作包含所有行指向指针有效的数组解释为数组的所有要素为SQL_ROW_PROCEED如果数组的所有要素设置为SQL_ROW_IGNORE则不变更被忽略的行的行状态数组的值

可与SQL_ATTR_ROW_OPERATION_PTR属性同时调用SQLSetStmtAttr 设置ARD的该字段

为了设置调用SQLExecute或SQLExecDirect时是否忽略此参数集合ARD的该标头字段指向应用程序可设置的值的参数操作数组数组的要素包含以下值

- SQL_PARAM_PROCEED：SQLExecute 或SQLExecDirect 调用包含参数集合
- SQL_PARAM_IGNORE：SQLExecute 或SQLExecDirect 调用不包含参数集合

没有设置数组要素时数组的所有参数集合用于调用SQLExecute或SQLExecDirectAPD的SQL_DESC_ARRAY_STATUS_PTR字段值为空指针时使用所有参数集合等同于指针指向有效的数组或所有要素为SQL_PARAM_PROCEED的数组

可与SQL_ATTR_PARAM_OPERATION_PTR属性同时调用SQLSetStmtAttr设置APD的该字段

SQL_DESC_BIND_OFFSET_PTR[Application descriptors]

该SQLLEN*头部字段指向绑定的offset其默认设置为空指针如果该字段不是空指针则驱动程序逆向参考指针及获取时间点描述符记录(SQL_DESC_DATA_PTR_SQL_DESC_INDICATOR_PTR和SQL_DESC_OCTET_LENGTH_PTR)的非空值的延迟字段的值绑定时使用新的指针值

绑定的offset始终直接追加于SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTR和SQL_DESC_OCTET_LENGTH_PTR字段如果变更offset值新的值直接继续追加于各描述符字段值中新的offset并非追加于之前的offset值

此字段为延迟的字段此字段不在设置时间点使用而是在有必要查看数据缓冲区地址时由驱动程序使用

可与SQL_ATTR_ROW_BIND_OFFSET_PTR属性同时调用SQLSetStmtAttr设置ARD的该字段也可与SQL_ATTR_PARAM_BIND_OFFSET_PTR属性同时调用SQLSetStmtAttr设置ARD的该字段

详细内容参考**SQLFetchScroll**或**SQLBindParameter**

SQL_DESC_BIND_TYPE[Application descriptors]

该SQLUINTEGER头部字段用于设置绑定的方向

ARD的该字段定义SQLFetchScroll或SQLFetch被相关的命令语句柄调用时的绑定方向

选择column的列式绑定时将此字段设置为SQL_BIND_BY_COLUMN（默认值）

可与SQL_ATTR_ROW_BIND_TYPE属性同时调用SQLSetStmtAttr设置ARD的此字段

此字段定义用于动态参数的绑定方向

选择参数的列式绑定时将此字段设置为SQL_BIND_BY_COLUMN（默认值）

可调用使用SQL_ATTR_PARAM_BIND_TYPE属性的SQLSetStmtAttr设置此字段

SQL_DESC_COUNT[All]

该SQLSMALLINT头部字段定义包含数据的最高级别记录的1-based索引驱动程序在描述符中设置数据结构体时为了显示重要记录的数量必须设置SQL_DESC_COUNT应用程序分配数据结构体的实例时不用指定为记录的预留空间如应用程序指定记录的内容驱动程序为了保证描述符句柄显示足够大小的数据结构体而执行必要的请求操作

SQL_DESC_COUNT不是绑定的所有数据列或所有参数数而是最高级别记录的数解除最高级别column或参数的绑定时SQL_DESC_COUNT变更为下一个最高级别的列或参数的数如

果解除小于最高级别的列或参数的数的列或参数的绑定时（TargetValuePtr 参数设置为空指针调用SQLBindCol或ParameterValuePtr参数设置为空指针调用SQLBindParameter时）不变更SQL_DESC_COUNT如果追加的列或参数绑定大于包含数据的最高级别记录的数时驱动程序自动增加SQL_DESC_COUNT字段的值通过SQL_UNBIND选项调用SQLFreeStmt解除对所有列的绑定时ARD和IRD中的SQL_DESC_COUNT字段设置为0通过SQL_RESET_PARAMS选项调用SQLFreeStmt时APD和IPD的SQL_DESC_COUNT字段设置为0应用程序可以调用SQLSetDescField明确设置SQL_DESC_COUNT值如果SQL_DESC_COUNT值明显减少则有效删除大于SQL_DESC_COUNT的新值的所有记录ARD的SQL_DESC_COUNT字段值明确设置为0时除了绑定的书签列释放所有的缓冲去ARD的此字段的记录数不包含绑定的书签列解除书签列绑定的唯一方法是把SQL_DESC_DATA_PTR字段设置为空指针

SQL_DESC_ROWS_PROCESSED_PTR[Implementation descriptors]

IRD的此SQLULEN*头部字段指包含调用SQLFetch或SQLFetchScroll后获取的行数调用QLBulkOperations或SQLSetPos执行的批量操作影响到的行数以及出错行数的缓冲区IPD的SQLUINTEGER*头部字段指包含处理的参数集合及错误数量的缓冲区如果为空指针不返回数量

SQL_DESC_ROWS_PROCESSED_PTR仅在调用SQLFetch或SQLFetchScroll(IRD)后或调用SQLExecuteSQLExecDirect或SQLParamData(IPD)并返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO后有效如果之前的函数没有返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO则不定义缓冲区的内容直到返回SQL_NO_DATA缓冲区的值设置为0

可与SQL_ATTR_ROWS_FETCHED_PTR属性同时调用SQLSetStmtAttr设置ARD的此字段ARD可与SQL_ATTR_PARAMS_PROCESSED_PTR属性同时调用并设置

应用程序分配此字段指向的缓冲区是驱动程序设置的延迟的输出缓冲区默认设置为空指

针

记录字段

各描述符包含一个以上的由基于描述符的类型定义column数据或动态参数中一个的字段构成的记录各记录为单个column或参数的完整的明细

SQL_DESC_AUTO_UNIQUE_VALUE[IRDs]

自动增加的column时只读SQLINTEGER的此记录字段为SQL_TRUE反之为SQL_FALSE此字段为只读但如果为自动增加column时就没必要是只读

SQL_DESC_BASE_COLUMN_NAME[IRDs]

只读SQLCHAR*的此记录字段包含结果集合column的默认column名如果没有默认column名则此字段包含空字符串

SQL_DESC_TABLE_NAME[IRDs]

只读SQLCHAR*的此记录字段包含结果集合column的默认表名如果不能定义或无法使用默认表名则此字段包含空字符串

SQL_DESC_CASE_SENSITIVE[Implementation descriptors]

排序比较列或参数时如果只读SQLINTEGER的此记录字段区分大小写则为SQL_TRUE不区分大小写或不是字符的column时则为SQL_FALSE

SQL_DESC_CATALOG_NAME[IRDs]

只读SQLCHAR*的此记录字段包含包含column的默认表的目录column为表达式或视图的一部分时返回值依赖于驱动程序数据源不支持目录或无法查看目录时此字段包含空字符串

SQL_DESC_CONCISE_TYPE[All]

此SQLSMALLINT的标头字段对包含datetime和interva数据类型的所有数据类型指定简洁的数据类型

SQL_DESC_CONCISE_TYPESQL_DESC_TYPE和SQL_DESC_DATETIME_INTERVAL_CODE字段值是互相依赖的如果字段中的一个被设置为time则其他的也要被设置

SQL_DESC_CONCISE_TYPE可通过调用SQLBindColSQLBindParameter或SQLSetDescField设置SQL_DESC_TYPE可通过调用SQLSetDescField或SQLSetDescRec设置

如果把SQL_DESC_CONCISE_TYPE设置为除interval或datetime数据类型外的其他简单的数据类型SQL_DESC_TYPE字段将被设置为相同的值SQL_DESC_DATETIME_INTERVAL_CODE字段被设置为0

如果把SQL_DESC_CONCISE_TYPE设置为简单的datetime或interval数据类型时

SQL_DESC_TYPE字段被设置为详细的类型(SQL_DATETIME或

SQL_INTERVAL)SQL_DESC_DATETIME_INTERVAL_CODE字段设置为适当的子码

SQL_DESC_DATETIME_INTERVAL_CODE[All]

SQLSMALLINT的记录字段在SQL_DESC_TYPE字段为SQL_DATETIME或SQL_INTERVAL时包含定义datetime或interval数据类型的子码其在SQL及C均相同代码包含代替datetime类型的"TYPE"或"C_TYPE", interval类型的"INTERVAL" 或"C_INTERVAL"的包含"CODE"的数据类型名称

把应用程序描述符的SQL_DESC_TYPE和SQL_DESC_CONCISE_TYPE设置为SQL_C_DEFAULT描述符与命令语句柄无关时不定义SQL_DESC_DATETIME_INTERVAL_CODE的内容

此字段可设置为以下表格列出的datetime数据类型

Datetime types	DATETIME_INTERVAL_CODE
SQL_TYPE_DATE/SQL_C_TYPE_DATE	SQL_CODE_DATE
SQL_TYPE_TIME/SQL_C_TYPE_TIME	SQL_CODE_TIME
SQL_TYPE_TIMESTAMP/SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP

此字段可设置为 以下表格列出的interval数据类型

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_DAY SQL_C_INTERVAL_DAY	SQL_CODE_DAY
SQL_INTERVAL_DAY_TO_HOUR SQL_C_INTERVAL_DAY_TO_HOUR	SQL_CODE_DAY_TO_HOUR
SQL_INTERVAL_DAY_TO_MINUTE SQL_C_INTERVAL_DAY_TO_MINUTE	SQL_CODE_DAY_TO_MINUTE
SQL_INTERVAL_DAY_TO_SECOND SQL_C_INTERVAL_DAY_TO_SECOND	SQL_CODE_DAY_TO_SECOND
SQL_INTERVAL_HOUR SQL_C_INTERVAL_HOUR	SQL_CODE_HOUR
SQL_INTERVAL_HOUR_TO_MINUTE SQL_C_INTERVAL_HOUR_TO_MINUTE	SQL_CODE_HOUR_TO_MINUTE
SQL_INTERVAL_HOUR_TO_SECOND SQL_C_INTERVAL_HOUR_TO_SECOND	SQL_CODE_HOUR_TO_SECOND

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_MINUTE SQL_C_INTERVAL_MINUTE	SQL_CODE_MINUTE
SQL_INTERVAL_MINUTE_TO_SECOND SQL_C_INTERVAL_MINUTE_TO_SECOND	SQL_CODE_MONUTE_TO_SECOND
SQL_INTERVAL_MONTH SQL_C_INTERVAL_MONTH	SQL_CODE_MONTH
SQL_INTERVAL_SECOND SQL_C_INTERVAL_SECOND	SQL_CODE_SECOND
SQL_INTERVAL_YEAR SQL_C_INTERVAL_YEAR	SQL_CODE_YEAR
SQL_INTERVAL_YEAR_TO_MONTH SQL_C_INTERVAL_YEAR_TO_MONTH	SQL_CODE_YEAR_TO_MONTH

SQL_DESC_DATETIME_INTERVAL_PRECISION[All]

SQLINTEGER的此记录字段在SQL_DESC_TYPE字段为SQL_INTERVAL时包含interval leading precisionSQL_DESC_DATETIME_INTERVAL_CODE字段设置为interval数据类型时此字段设置默认的interval leading precision

SQL_DESC_DISPLAY_SIZE[IRDs]

此只读SQLINTEGER的记录字段包含展示列数据而所需的最大字符数量

SQL_DESC_FIXED_PREC_SCALE[Implementation descriptors]

如果column为exact numeric column并有非0的scale与固定precision时其只读

SQLSMALLINT记录字段设置SQL_TRUE反之设置SQL_FALSE

SQL_DESC_INDICATOR_PTR[Application descriptors]

存在于ARD此SQLLEN*记录字段表示指示符变量column值为NULL时此变量包含SQL_NULL_DATEAPD的情况下为了把动态参数指定为NULL将指示符变量设置为SQL_NULL_DATA否则变量为0

ARD的SQL_DESC_INDICATOR_PTR字段为空指针时驱动程序拦截判断column是否为空的信息column为空SQL_DESC_INDICATOR_PTR为空指针时驱动程序调用SQLFetch或SQLFetchScroll后生成缓冲区时返回SQLSTATE 22002 (Indicator variable required but not supplied)如果SQLFetch或SQLFetchScroll 的调用不返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO则不定义缓冲区的内容

SQL_DESC_INDICATOR_PTR字段决定是否设置SQL_DESC_OCTET_LENGTH_PTR表示的字段column的数据值为NULL时驱动程序把指示符变量设置为SQL_NULL_DATA此时不设置SQL_DESC_OCTET_LENGTH_PTR表示的字段获取数据时如果不接收空值则将SQL_DESC_INDICATOR_PTR指向的缓冲区设置为0SQL_DESC_OCTET_LENGTH_PTR指向的缓冲区设置为数据的长度

APD的SQL_DESC_INDICATOR_PTR字段为空指针时应用程序为了把参数定义为空不能使用此描述符记录

此字段为延迟的字段设置时不使用此字段但驱动程序显示NULL可能性(ARD)或决定NULL可能性(APD)时使用

SQL_DESC_LABEL[IRDs]

只读SQLCHAR*的此记录字段包含列的标签或标记列不包含标签时此变量包含列的名称未命名列或无法使用标签时此变量包含空字符串

SQL_DESC_LENGTH[All]

SQLULEN的此记录字段为字符串或字节单位的二进制数据的最大长度或实际长度其是固定长度数据类型的最大长度或可变长数据类型的实际长度此值总是排除字符串最后的空终止字符类型为SQL_TYPE_DATESQL_TYPE_TIMESQL_TYPE_TIMESTAMP或SQL interval数据类型值时此字段为datetime或interval值重新转换为字符串类型时的字符长度

此字段的值不同于ODBC 2.x定义的"length"值

SQL_DESC_LITERAL_PREFIX[IRDs]

只读SQLCHAR*的此记录字段包含字符或驱动程序将其数据类型的字符识别为后缀的字符此变量为了无法转换为字符后缀的数据类型而包含空字符

SQL_DESC_LOCAL_TYPE_NAME[Implementation descriptors]

只读SQLCHAR*的此记录字段在数据类型中包含区域化名称这可能与数据类型的正规名称不同没有区域化名称时返回空字符串此字段只是展现为目的的字段

SQL_DESC_NAME[Implementation descriptor]

SQLCHAR*的此记录字段在行描述符中包含字段的别名如果不适用column的别名则返回column名根据情况驱动程序设置SQL_DESC_NAME字段时把SQL_DESC_UNNAMED字段设置为SQL_NAMED如果没有column名或column别名时驱动程序返回SQL_DESC_NAME字段的空字符把SQL_DESC_UNNAMED字段设置为SQL_UNNAMED

为了指定有参数名称或有名称的已存储的procedure参数应用程序可以设置IPD的SQL_DESC_NAME字段IRD的SQL_DESC_NAME字段为只读字段当应用程序设置此字段时将返回SQLSTATE HY091 (Invalid descriptor field identifier)

IPD的情况驱动程序不支持命名的参数时此字段不会被定义驱动程序支持命名的参数并可说明参数时此字段将返回参数名称

SQL_DESC_NULLABLE[Implementation descriptors]

在IRD的情况只读SQLSMALLINT的此记录字段的column可以为空值时是SQL_NULLABLE；不能有空值时是SQL_NO_NULLS；无法知道是否有空值时是SQL_NULLABLE_UNKNOWN此字段是为结果集的column而存在

在IPD的情况动态参数可以总是为空 应用程序不能设置此字段因此才字段始终为SQL_NULLABLE

SQL_DESC_NUM_PREC_RADIX[All]

SQLINTEGER的此字段在QL_DESC_TYPE字段为approximate numeric数据类型时为2因为SQL_DESC_PRECISION字段包含位数SQL_DESC_TYPE字段为exact numeric数据类型时SQL_DESC_PRECISION字段包含小数点位数因此此字段为10此字段在不是数字型数据类型时设置为0

SQL_DESC_OCTET_LENGTH[All]

SQLLEN的此记录字段包含字符串或二进制数据类型的字节单位长度固定长度的字符或二进制类型为字节单位的实际长度可变长度的字符或二进制类型为字节单位的最大长度此值对执行描述符不包含空终止字符的空白对应用程序描述符包含空终止字符的空白此字段包含应用程序数据的缓冲区大小APD的情况此字段只定义输出或输入/输出参数

SQL_DESC_OCTET_LENGTH_PTR[Application descriptors]

SQLLEN*的此记录字段指包含动态参数（参数描述符）或绑定的column值（行描述符）的字节单位的总长度的变量

APD的情况此值忽略字符串和二进制以外的所有参数如果此字段为SQL_NTS则动态参数要以NULL终止执行时为了显示数据参数为要绑定的参数应用程序在执行时把包含SQL_DATA_AT_EXEC或SQL_LEN_DATA_AT_EXEC Macro结果的变量设置于有APD记录的此字段这种的字段为一个以上时SQL_DESC_DATA_PTR设置为有助于决定应用程序要求的参数的识别参数的值

ARD的OCTET_LENGTH_PTR字段为空指针时驱动程序不返回column的长度信息APD的SQL_DESC_OCTET_LENGTH_PTR为空指针时驱动程序认为字符串和二进制值以NULL终止（二进制值不应该以NULL终止但为了避免数据被截断应赋予长度）

填充此字段指向的缓冲区的SQLFetch或SQLFetchScroll不返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO时不定义缓冲区的内容此字段为延迟的字段此字段不会马上被使用而后续驱动程序表示或决定数据的octet长度时使用

SQL_DESC_PARAMETER_TYPE[IRDs]

SQLSMALLINT的此记录字段在输入参数设置SQL_PARAM_INPUT在输入/输出参数设置SQL_PARAM_INPUT_OUTPUT在输出参数设置SQL_PARAM_OUTPUT在流输入/输出参数设置SQL_PARAM_INPUT_OUTPUT_STREAM或在流输出参数设置SQL_PARAM_OUTPUT_STREAM其默认设置为SQL_PARAM_INPUT

SQL_DESC_PRECISION[All]

SQLSMALLINT的此记录字段对exact numeric类型包含有效的整数数量对approximate numeic包含mantissa（二进制精度）位数或对SQL_TYPE_TIMESQL_TYPE_TIMESTAMP或SQL_INTERVAL_SECOND数据类型的fractional包含秒部分的有效整数的数量此字段不定义其他所有数据类型

此字段值不等于ODBC 2.x定义的"precision"值

SQL_DESC_ROWVER[Implementation descriptors]

SQLSMALLINT的此记录字段更新行时（例如SQL Serever的"timestamp"）表示数据库是否自动修改columncolumn为行版本column时此记录字段值为SQL_TRUE反之为SQL_FALSE此字段属性类似于为了决定列的自动更新与否在SQLSpecialColumn的IdentifierType参数设置SQL_ROWVER并调用

SQL_DESC_SCALE[All]

SQLSMALLINT的此记录字段包含decimal与numeric数据类型的定义的小数点位数此字段不在其他所有数据类型中定义

此字段的值不等于ODBC 2.x定义的"scale"值

SQL_DESC_SCHEMA_NAME[IRDs]

只读SQLCHAR*的此记录字段包含包含column的默认表的SCHEMA名称列为表达式或视图的一部分时返回值依赖于驱动程序数据源无法查看SCHEMA或SCHEMA名称时此变量包含空字符

SQL_DESC_SEARCHABLE[IRDs]

只读SQLSMAINT的此记录字段设置为以下值中的一个值

• Column不能用于WHERE子句时为SQL_PRED_NONE（其等同于ODBC 2.X的SQL_UNSEARCHABLE）

• Column只能与WHERE子句的LIKE条件一起使用时为SQL_PRED_CHAR

• Column在WHERE子句中可以与除LIKE条件外的其他比较运算符一起使用时为SQL_PRED_BASIC（其等同于ODBC 2.x的SQL_EXCEPT_LIKE值）

• Column可以与WHERE子句的任何比较运算符一起使用时为SQL_PRED_SEARCHABLE

SQL_DESC_TABLE_NAME[IRDs]

只读SQLCHAR*的此记录字段包含此column所属的表名column为表达式或视图的一部分时返回值依赖于驱动程序

SQL_DESC_TYPE[All]

SQLSMALLINT的此记录字段包含除了datetime与interval数据类型外的所有数据类型的缩略的SQL或C数据类型对datetime和interval数据类型此字段指定SQL_DATETIME或

SQL_INTERVAL

此字段包含SQL_DATETIME 或SQL_INTERVAL时SQL_DESC_DATETIME_INTERVAL_CODE字段必须包含缩略类型的适当的Sub code对datetime数据类型SQL_DESC_TYPE包含SQL_DATETIMESQL_DESC_DATETIME_INTERVAL_CODE字段包含定义datetime数据类型的Sub code对interval数据类型SQL_DESC_TYPE包含SQL_INTERVALSQL_DESC_DATETIME_INTERVAL_CODE字段包含定义interval数据类型的Sub code

SQL_DESC_TYPE和SQL_DESC_CONCISE_TYPE字段值相互依赖设置字段中的一个时其他字段也要设置SQL_DESC_TYPE是通过调用SQLSetDescField或SQLSetDescRec 设置SQL_DESC_CONCISE_TYPE是通过调用SQLBindColSQLBindParameter或SQLSetDescField设置

SQL_DESC_TYPE设置非interval或datetime数据类型的缩略的数据类型时SQL_DESC_CONCISE_TYPE字段设置为相同值SQL_DESC_DATETIME_INTERVAL_CODE字段设置为0

SQL_DESC_TYPE设置为datetime或interval数据类型的长数据类型（SQL_DATETIME 或SQL_INTERVAL）SQL_DESC_DATETIME_INTERVAL_CODE字段设置为适当的sub code时SQL_DESC_CONCISE_TYPE字段设置为对应于缩略的数据类型的值SQL_DESC_TYPE设置为缩略的datetime或interval类型中的一个时返回SQLSTATE HY021 (Inconsistent descriptor information)

SQL_DESC_TYPE字段通过调用SQLBindCol, SQLBindParameter或SQLSetDescField设置时如下表以下字段设置为对应的默认值不定义相同记录的剩余的字段

SQL_DESC_TYPE 值	其他字段的默认设置
SQL_CHAR, SQL_VARCHAR, SQL_C_CHAR, SQL_C_VARCHAR	SQL_DESC_LENGTH设置为1 SQL_DESC_PRECISION设置为0
SQL_DATETIME	SQL_DESC_DATETIME_INTERVAL_CODE设置为SQL_CODE_DATE或SQL_CODE_TIME时SQL_DESC_PRECISION设置为0设置为SQL_DESC_TIMESTAMP时SQL_DESC_PRECISION设置为6
SQL_DECIMAL, SQL_NUMERIC, SQL_C_NUMERIC	SQL_DESC_SCALE设置为0SQL_DESC_PRECISION设置为各数据类型定义的精密度
SQL_FLOAT, SQL_C_FLOAT	SQL_DESC_PRECISION设置为SQL_FLOAT定义的默认精密度
SQL_INTERVAL	SQL_DESC_DATETIME_INTERVAL_CODE以interval数据类型设置时SQL_DESC_DATETIME_INTERVAL_PRECISION设置为2（默认interval leading precision）interval有秒部分时SQL_DESC_PRECISION设置为6（默认interval seconds precision）

应用程序不调用SQLSetDescRec而调用SQLSetDescField设置描述符的字段时应用程序必须先定义数据类型此时之前表的其他字段也默认被设置如果不允许默认设置应用程序可通过调用SQLSetDescField或SQLSetDescRec设置

SQL_DESC_TYPE_NAME[Implementation descriptors]

只读SQLCHAR*的此记录字段包含依赖于数据源的类型名称（"CHAR", "VARCHAR"等）如果数据类型名称为unknown时变量包含空字符

SQL_DESC_UNNAMED[Implementation descriptors]

设置SQL_DESC_NAME字段时存在于行描述符SQLSMALLINT的此字段被驱动程序设置为SQL_NAME或SQL_UNNAMED中的一个SQL_DESC_NAME字段包含column的别名或不适用column的别名时SQL_DESC_UNNAMED字段被驱动程序设置为SQL_NAMED当应用程序把IPD的SQL_DESC_NAME字段设置为参数名称或别名时驱动程序把IPD的SQL_DESC_UNNAMED字段设置为SQL_NAMED不存在column名称或别名时驱动程序把SQL_DESC_UNNAMED字段设置为SQL_UNNAMED

应用程序可以把IPD的SQL_DESC_UNNAMED字段设置为SQL_UNNAMED如果应用程序试图把IPD的SQL_DESC_UNANMED字段设置为SQL_NAMED则返回SQLSTATE HY091 (Invalid descriptor field identifier)如果只读应用程序试图设置IRD的SQL_DESC_UNNAMED字段设置时则返回SQLSTATE HY091(Invalid descriptor field idenfitier)

SQL_DESC_UNSIGNED[Implemetation descriptors]

column类型为unsigned或non-numeric时只读SQLSMALLINT的此记录字段设置为SQL_TRUE；字段类型为signed时设置为SQL_FALSE

SQL_DESC_UPDATABLE[IRDs]

只读SQLSMALLINT的此记录字段设置为以下值中的一个

- 结果集合column为只读时是SQL_ATTR_READ_ONLY
- 结果集合column为可读可写时是SQL_ATTR_WRITE
- 无法知道是否可更新结果集合column时是SQL_ATTR_READWRITE_UNKNOWN

说明SQL_DESC_UPDATABLE结果集合中的（非默认表的column）column的更新可能性

以此结果集合column为基础的默认表的column的更新可能性可能与此字段中的值不同是否可更新column依赖于数据类型用户权限以及结果集合本身的定义无法确定是否可更新column时需返回SQL_ATTR_READWITE_UNKNOWN

SQL_DESC_CHAR_LENGTH_UNITS[Alls]

SQLSMALLINT的此记录字段表示SQL类型为SQL_CHARSQL_VARCHAR或SQL_LONGVARCHAR的字段长度单位

• SQL_CLU_CHARACTERS：长度单位为CHARACTER例如“字符串”数据在编码方式为UHC(Unified Hangul Code)时其长度为3

• SQL_CLU_OCTETS：长度单位为OCTETS例如“字符串”数据在编码方式为UHC时其长度为6

• SQL_CLU_NONE：未定义长度单位除以上SQL类型外的SQL类型时返回的值

SQLGetDescRec

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLGetDescRec返回描述符记录的多个字段的当前设置或值返回的字段说明名称数据类型column的大小或参数数据

语句

```
SQLRETURN SQLGetDescRec(
    SQLHDESC        DescriptorHandle,
    SQLSMALLINT     RecNumber,
    SQLCHAR *       Name,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   StringLengthPtr,
    SQLSMALLINT *   TypePtr,
    SQLSMALLINT *   SubTypePtr,
    SQLLEN *        LengthPtr,
    SQLSMALLINT *   PrecisionPtr,
    SQLSMALLINT *   ScalePtr,
    SQLSMALLINT *   NullablePtr);
```

参数

DescriptorHandle

【输入】描述符句柄

RecNumber

【输入】应用程序要查找的信息的描述符记录记录编号0为书签记录描述符记录从1开始编号RecNumber小于等于SQL_DESC_COUNT但行中不包含column或参数数据时SQLGetDescField返回字段的默认值

Name

【输出】返回描述符记录的SQL_DESC_NAME的缓冲区指针Name为空时StringLengthPtr返回Name的缓冲区长度（不包含null终止字符）

BufferLength

【输入】Name缓冲区长度

StringLengthPtr

【输出】返回*Name中除null终止字符外可返回的字符数量的指针字符数量大于或等于BufferLength时*Name在BufferLength中减少null终止字符长度长度并后接null终止字符

TypePtr

【输出】返回描述符记录的SQL_DESC_TYPE字段值的缓冲区指针

SubTypePtr

【输出】返回SQL_DATETIME或SQL_INTERAVL类型记录的

SQL_DESC_DATETIME_INTERVAL_CODE字段值的缓冲区指针

LengthPtr

【输出】返回描述符记录的SQL_DESC_OCTET_LENGTH字段值的指针

PrecisionPtr

【输出】返回描述符记录的SQL_DESC_PRECISION字段值的指针

ScalePtr

【输出】返回描述符记录的SQL_DESC_SCALE字段值的指针

NullablePtr

【输出】返回描述符记录的SQL_DESC_NULLABLE字段值的指针

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE.

RecNumber大于当前描述符记录数时返回SQL_NO_DATA

DescriptorHandle为IRD句柄命令语处于准备或执行状态但没有相关的游标时返回SQL_NO_DATA

检测

SQLSTATE	报错	说明
01000	General warning	各个驱动程序信息（函数返回 SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
01004	String data, right truncated	*ValuePtr的缓冲区的长度小于描述符字段值的长度因此被截断向*StringLengthPtr返回剩余的描述符字段的长度（函数返回SQL_SUCCESS_WITH_INFO）
07009	Invalid descriptor index	fieldIdentifier参数为记录字段RecNumber参数值为0DescriptorHandle参数为IPD句柄 RecNumber参数为0SQL_ATTR_USE_BOOKMARKS状态属性为SQL_UB_OFFDescriptorHandle参数为IRD句柄 RecNumber参数小于0
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY007	Associated statement is not prepared	DescriptorHandle与IRD句柄相关相关的命令语句柄不处于准备或执行阶段
HY010	Function sequence error	DescriptorHandle相关的StatementHandle异步执行函数调用时仍然在执行此函数 调用SQLExecuteSQLExecDirect后返回了SQL_NEED_DATA传送所有data-at-execution变量之前调用了此函数 调用了DescriptorHandle相关的连接句柄的函数但调用SQLGetDescRec时仍然在执行该函数

SQLSTATE	报错	说明
HY013	Memory management error	用作参数的缓冲区大小值小于0或无法访问内存
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran**函数
HYT01	Connection timeout expired	数据源响应请求之前连接超时其限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	与DescriptorHandle相关的驱动程序不支持函数

说明

对单个column或参数检索如下描述符字段的值时可调用SQLGetDescRec

- SQL_DESC_NAME
- SQL_DESC_TYPE
- SQL_DESC_DATETIME_INTERVAL_CODE
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE

- SQL_DESC_NULLABLE

SQLGetDescRec不检索头部字段的值

应用程序通过设置对应空指针字段的参数避免返回对字段的设置

应用程序调用SQLGetDescRec检索没有对特定描述符类型定义的字段值时函数返回SQL_SUCCESS但不定义字段的返回值例如为了APD或ARD的SQL_DESC_NAME或SQL_DESC_NULLABLE字段调用SQLGetDescRec时虽然返回SQL_SUCCESS但不定义字段的值

应用程序为了检索定义为特定描述符类型或没有设置为默认值的字段值调用SQLGetDescRec时函数返回SQL_SUCCESS但不定义字段的值详细的内容参考SQLSetDescField的"**描述符字段的初始化**"

通过调用SQLGetDescField可以分别检索各个字段的值对描述符头部或记录中的字段内容参考**SQLSetDescField**

SQLGetDiagField

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLGetDiagField在包含报错告警状态信息的检测数据结构中返回记录字段的当前值

语句

```
SQLRETURN SQLGetDiagField(
    SQLSMALLINT     HandleType,
    SQLHANDLE       Handle,
    SQLSMALLINT     RecNumber,
    SQLSMALLINT     DiagIdentifier,
    SQLPOINTER      DiagInfoPtr,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   StringLengthPtr);
```

参数

HandleType

【输入】需要检测的句柄类型的句柄类型标识符标识符应为以下中的一个

- SQL_HANDLE_DBC
- SQL_HANDLE_DESC
- SQL_HANDLE_ENV
- SQL_HANDLE_STMT

Handle

【输入】HandleType指向的用于类型的检测数据结构的句柄HandleType为SQL_HANDLE_ENV时handle可以是共享或非共享环境句柄

RecNumber

【输入】显示应用程序要查找的信息的状态记录状态记录从1开始编号DiagIdentifier参数指向检测头部的任何字段时忽略RecNumber反之RecNumber应大于0

DiagIdentifier

【输入】返回值的检测字段更详细的内容参考说明部分的**DiagIdentifier 参数**

DiagInfoPtr

【输出】返回检测信息的缓冲区指针数据类型取决于DiagIdentifier的值部分驱动程序只记录缓冲区的32位或16位的低位保留高位因此DiagInfoPtr为整数类型时应用程序应使用SQLULEN的缓冲区调用函数前将值初始化为0DiagInfoPtr为空时StringLengthPtr返回DiagInfoPtr指向的缓冲区中除空终止字符外的字节的总数

BufferLength

【输入】DiagIdentifier为ODBC定义的检测DiagInfoPtr指向字符串或二进制缓冲区时此参数应为*DiagInfoPtr的长度DiagIdentifier为ODBC定义的检测字段*DiagInfoPtr为数字时忽略BufferLength（调用SQLGetDiagFieldW时）*DiagInfoPtr值为unicode字符串时

BufferLength参数应为偶数DiagIdentifier为驱动程序定义的字段时应用程序设置BufferLength参数指向驱动程序管理器的字段特性BufferLength可以有以下值

• DiagInfoPtr为字符串指针时BufferLength为字符串长度或SQL_NTS

• DiagInfoPtr为二进制缓冲区的指针时应用程序将SQL_LENG_BINARY_ATTR(length)macro的结果存储于BufferLength负数存储在BufferLength

• DiagInfoPtr并非字符串或二进制缓冲区的指针时BufferLength为SQL_IS_POINTER值

• *DiagInfoPtr为固定长度数据类型时BufferLength为SQL_IS_INTEGERSQL_IS_UINTEGERSQL_IS_SMALLINT或SQL_IS_USMALLINT中的一个

StringLengthPtr

【输出】向字符数据返回*DiagInfoPtr中可返回的除null终止字符外的总字节长度的缓冲区指针可返回的字节长度大于或等于BufferLength时*DiagInfoPtr中的text截断BufferLength中除null终止字符外的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_NO_DATA

检测

SQLGetDiagField不显示检测记录使用以下返回值的结果

- SQL_SUCCESS: 函数成功返回检测信息
- SQL_SUCCESS_WITH_INFO: 不足以存储请求*DiagInfoPtr的检测字段的返回值而被截断为了查看是否被截断应用程序对比BufferLength与*StringLengthPtr记录的实际字节长度
- SQL_INVALID_HANDLE: Handle对HandleType指向的类型无效

- SQL_ERROR: 发生以下情况中的一个情况
  - DiagIdentifier参数为无效值
  - DiagIdentifier参数为SQL_DIAG_CURSOR_ROW_COUNTSQL_DIAG_DYNAMIC_FUNCTIONSQL_DIAG_DYNAMIC_FUNCTION_CODESQL_DIAG_ROW_COUNT中的一个handle不是命令语句柄
  - DiagIdentifier指向检测记录的字段时RecNumber参数为负数或0对头部字段忽略RecNumber
  - 请求的值为字符串BufferLength小于0
  - 使用异步提醒时句柄的异步执行未结束
- SQL_NO_DATA: RecNumber大于对handle指定的检测记录数没有handle检测记录时对所有正数RecNumber函数均返回SQL_NO_DATA

说明

应用程序通常为了以下三种目的中的一个调用SQLGetDiagField

- 函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时要获取具体报错或告警信息时(SQLBrowseConnect返回SQL_NEED_DATA)
- 通过SQLExecuteSQLExecDirectSQLBulkOperationsSQLSetPos执行INSERTDELETE或UPDATE命令并要从数据源获取行的数量时或驱动程序可提供并要获取当前打开的游标中的行数时
- 要查看哪个函数调用执行SQLExecDirect或SQLExecute时

每次调用此函数时所有的ODBC函数都会显示0条以上的检测记录因此应用程序可在调用所有函数后再调用SQLGetDiagField不限制单次可存储的检测记录数SQLGetDiagField检索最新的与handle参数中指定的检测数据结构相关的检测信息当应用程序调用其他ODBC函数代替

SQLGetDiagField或SQLGetDiagRec时可能丢失同一个句柄的之前通过调用获取的检测信息

SQLGetDiagField返回SQL_SUCCESS时应用程序可增加RecNumber的同时读取检测记录状态记录数标记于SQL_DIAG_NUMBER头部字段SQLGetDiagField的调用不影响头部字段与记录字段过程中不调用检测函数外的其他函数时应用程序可使用相同句柄再次调用SQLGetDiagField检索记录的字段

应用程序为了始终能返回所有的检测字段可以调用SQLGetDiagFieldhandle不是命令语句柄时SQL_DIAG_CURSOR_ROW_COUNT或SQL_DIAG_ROW_COUNT返回SQL_ERROR因此被排除如果未定义任何检测字段调用SQLGetDiagField将返回SQL_SUCCESS和没有设置的值

除异步执行的函数外API调用返回HY010(Fuction sequence error)但结束异步执行前无法检索报错记录

HandleType 参数

各个句柄类型有于此相关的检测信息HandleType参数指向Handle的类型

部分标头与记录字段不会对环境连接命令语描述符句柄返回以下头部字段与记录字段表说明了字段中不恰当的句柄

HandleType为SQL_HANDLE_ENV时handle可以为共享或不共享环境句柄

驱动程序的特定标头检测记录与环境句柄无关

对描述符句柄定义的检测header只有SQL_DIAG_NUMBER与SQL_DIAG_RETURNCODE

DiagIdentifier 参数

此参数为检测数据结构所需要的字段的标识符RecNumber大于等于1时字段的数据为函数返回

的检测信息RecNumber为0时字段存在于检测数据结构的头部包含与返回非具体信息的检测信息的函数调用相关的数据

驱动程序在检测数据结构中可定义各驱动程序的头部和记录字段

使用ODBC 2.x驱动程序的ODBC 3.x应用程序在DiagIdentifier参数为SQL_DIAG_CLASS_ORIGINSQL_DIAG_CLASS_SUBCLASS_ORIGINSQL_DIAG_CONNECTION_NAMESQL_DIAG_MESSAGE_TEXTSQL_DIAG_NATIVESQL_DIAG_NUMBERSQL_DIAG_RETURNCODESQL_DIAG_SERVER_NAMESQL_DIAG_SQLSTATE时可调用SQLGetDiagField对其他检测字段返回SQL_ERROR

头部字段

DiagIdentifier	返回类型	返回
SQL_DIAG_CURSOR_ROW_COUNT	SQLLEN	此字段包含游标中的行数其指取决于SQLGetInfo信息类型信息类型SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2SQL_KEYSET_CURSO表示各游标类型(存在于SQL_CA2_CRC_EXACT和SQL_CA2_CRC_APPL 此字段的内容仅在调用命令语句柄SQLExecuteSQLExecDirect或SQL SQL_DIAG_CURSOR_ROW_COUNT以外的命令语句柄时SQLGetDiagF
SQL_DIAG_DYNAMIC_FUNCTION	SQLCHAR *	说明执行基本函数的SQL命令语的字符串此字段内容在调用SQLExe DiagIdentifier为SQL_DIAG_DYNAMIC_FUNCTION以外命令语句柄时
SQL_DIAG_DYNAMIC_FUNCTION_CODE	SQLINTEGER	说明执行基本函数的SQL命令语的数字代码此字段内容在调用SQLE DiagIdentifier为SQL_DIAG_DYNAMIC_FUNCTION_CODE以外命令语
SQL_DIAG_NUMBER	SQLINTEGER	指定句柄的可使用的状态记录的数字

DiagIdentifier	返回类型	返回
SQL_DIAG_RETURNCODE	SQLRETURN	函数返回的代码驱动程序不需要实现SQL_DIAG_RETURNCODE由驱 没有函数调用handle时为了SQL_DIAG_RETURNCODE返回SQL_SUCC
SQL_DIAG_ROW_COUNT	SQLLEN	由SQLExecuteSQLExecDirectSQLBulkOperations执行的INSERTDEL 驱动程序定义此字段的内容只为命令语句柄定义 DiagIdentifier为SQL_DIAG_ROW_COUNT以外的命令语句柄时SQLGe SQLRowCount的RowCountPtr返回此字段的数据中向SQLRowCoun 备或分配状态调用非检测函数后重新设置

记录字段

DiagIdentifier	返回类型	返回
SQL_DIAG_CLASS_ORIGIN	SQLCHAR *	表示定义此记录中的SQLSTATE值的CLASS部分 的文档的字符串此值对由开放group与ISO调用级 别接口定义的所有SQLSTATE是ISO 9075对指定 的（SQLSTATE calss为IM的所有)ODBC SQLSTATEs此值为ODBC 3.0

DiagIdentifier	返回类型	返回
SQL_DIAG_COLUMN_NUMBER	SQLINTEGER	行集合或参数集合中SQL_DIAG_ROW_NUMBER为有效的行编号时此字段为表示结果集合的column编号或参数集合的参数编号的值结果集合的column编号总是从1开始如果此状态记录与书签column有关此字段可以为0参数编号是从1开始 状态记录与column编号和参数编号无关时其值为SQL_NO_COLUMN_NUMBER值如果驱动程序无法决定与记录相关的column编号或参数编号时此字段的值为SQL_COLUMN_NUMBER_UNKNOWN 此字段值只对命令语句柄定义
SQL_DIAG_CONNECTION_NAME	SQLCHAR *	表示与检测记录相关的连接名称的字符串此字段是驱动程序定义为了与检测数据结构相关的环境句柄和服务器之间无关的检测此字段是长度为0的字符串
SQL_DIAG_MESSAGE_TEXT	SQLCHAR *	错误或告警信息
SQL_DIAG_NATIVE	SQLINTEGER	驱动程序/数据源的具体的原始报错代码如果没有原始报错代码驱动程序返回0

DiagIdentifier	返回类型	返回
SQL_DIAG_ROW_NUMBER	SQLLEN	此字段包含状态记录与相关的行集合的行编号或参数集合的参数编号行编号与参数编号从1开始状态记录与行编号或参数编号无关时此字段值为SQL_NO_ROW_NUMBER驱动程序无法决定此记录和相关的行编号或参数编号时此字段值为SQL_ROW_NUMBER_UNKNOWN 此字段内容只能对命令语句柄定义
SQL_DIAG_SERVER_NAME	SQLCHAR *	表示与检测记录相关的服务器名称的字符串用SQL_DATA_SOURCE_NAME选项调用SQLGetInfo后返回的值相同为了与数据结构相关的环境句柄和服务器之间无关的检测此字段是长度为0的字符串
SQL_DIAG_SQLSTATE	SQLCHAR *	五个字符的SQLSTATE检测代码

DiagIdentifier	返回类型	返回
SQL_DIAG_SUBCLASS_ORIGIN	SQLCHAR *	与定义SQLSTATE代码子类部分的识别SQL_DIAG_CLASS_ORIGIN相同格式的有效值的字符串 ODBC 3.0中ODBC指定SQLSTATE代码如下 01S00, 01S01, 01S02, 01S06, 01S07, 07S01, 08S01, 21S01, 21S02, 25S01, 25S02, 25S03, 42S01, 42S02, 42S11, 42S12, 42S21, 42S22, HY095, HY097, HY098, HY099, HY100, HY101, HY105, HY107, HY109, HY110, HY111, HYT00, HYT01, IM001, IM002, IM003, IM004, IM005, IM006, IM007, IM008, IM010, IM011, IM012.

动态函数字段的值

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
alter-domain-statement	"ALTER DOMAIN"	SQL_DIAG_ALTER_DOMAIN
alter-table-statement	"ALTER TABLE"	SQL_DIAG_ALTER_TABLE
assertion-definition	"CREATE ASSERTION"	SQL_DIAG_CREATE_ASSERTION

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
character-set-definition	"CREATE CHARACTER SET"	SQL_DIAG_CREATE_CHARACTER_SET
collation-definition	"CREATE COLLATION"	SQL_DIAG_CREATE_COLLATION
create-index-statement	"CREATE INDEX"	SQL_DIAG_CREATE_INDEX
create-table-statement	"CREATE TABLE"	SQL_DIAG_CREATE_TABLE
create-view-statement	"CREATE VIEW"	SQL_DIAG_CREATE_VIEW
cursor-specification	"SELECT CURSOR"	SQL_DIAG_SELECT_CURSOR
delete-statement-positioned	"DYNAMIC DELETE CURSOR"	SQL_DIAG_DYNAMIC_DELETE_CURSOR
delete-statement-searched	"DELETE WHERE"	SQL_DIAG_DELETE_WHERE
domain-definition	"CREATE DOMAIN"	SQL_DIAG_CREATE_DOMAIN
drop-assertion-statement	"DROP ASSERTION"	SQL_DIAG_DROP_ASSERTION

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
drop-character-set-stmt	"DROP CHARACTER SET"	SQL_DIAG_DROP_CHARACTER_SET
drop-collation-statement	"DROP COLLATION"	SQL_DIAG_DROP_COLLATION
drop-domain-statement	"DROP DOMAIN"	SQL_DIAG_DROP_DOMAIN
drop-index-statement	"DROP INDEX"	SQL_DIAG_DROP_INDEX
drop-schema-statement	"DROP SCHEMA"	SQL_DIAG_DROP_SCHEMA
drop-table-statement	"DROP TABLE"	SQL_DIAG_DROP_TABLE
drop-translation-statement	"DROP TRANSLATION"	SQL_DIAG_DROP_TRANSLATION
drop-view-statement	"DROP VIEW"	SQL_DIAG_DROP_VIEW
grant-statement	"GRANT"	SQL_DIAG_GRANT
insert-statement	"INSERT"	SQL_DIAG_INSERT

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
ODBC-procedure-extension	"CALL"	SQL_DIAG_ CALL
revoke-statement	"REVOKE"	SQL_DIAG_REVOKE
schema-definition	"CREATE SCHEMA"	SQL_DIAG_CREATE_SCHEMA
translation-definition	"CREATE TRANSLATION"	SQL_DIAG_CREATE_TRANSLATION
update-statement-positioned	"DYNAMIC UPDATE CURSOR"	SQL_DIAG_DYNAMIC_UPDATE_CURSOR
update-statement-searched	"UPDATE WHERE"	SQL_DIAG_UPDATE_WHERE
Unknown	empty string	SQL_DIAG_UNKNOWN_STATEMENT

状态记录的顺序

状态记录根据行编号与检测类型顺序定位驱动程序管理器决定返回其生成的状态记录的最终顺序

如果驱动程序管理器与驱动程序显示检测记录则驱动程序管理器决定检测记录的顺序

当有两个以上的状态记录时记录的顺序首先取决于行编号Row决定检测记录的顺序时适用如下规则

- SQL_NO_ROW_NUMBER定义为-1因此不对应任何行的记录位于对应特定行的记录之前
- SQL_ROW_NUMBER_UNKNOWN定义为-2因此无法知道行编号的记录位于其他所有记录之前
- 与特定行相关的所有记录分类为SQL_DIAG_ROW_NUMBER字段的值列出第一行的所有报错与告警后按照顺序列出下一行的所有报错和告警

> Note:
>
> 在ODBC 2.x驱动程序返回SQLSTATE 01S01或在ODBC 3.x驱动程序调用SQLExtendedFetch或向位于SQLExtendedFetch的游标调用SQLSetPos时如果返回SQLSTATE 01S01则ODBC 3.x驱动程序管理器不会向检测队列请求状态记录

对在各行中或者不对应行或不知道行编号的所有记录或者拥有SQL_NO_ROW_NUMBER等行编号的所有记录通过排序规则的集合决定第一个列出的记录第一条记录后不定义影响行的其他记录的顺序第一条记录后应用程序无法假设报错在告警前出现应用程序为了尽可能获取函数调用失败的信息需要检查全部的检测数据结构

决定行的第一个记录时适用如下规则最高等级的记录为第一条记录记录的源（驱动程序管理器驱动程序gateway等）不影响决定记录顺序

- 记录报错的errors状态记录为最高等级通过以下规则排序报错
  - 事务失败或指向可能发生事务失败的记录比其他记录的等级高
  - 描述相同报错状态的两个以上的记录时开源group CLI配置（CLASS 03~HZ）中定义的SQLSTATE的等级高于ODBC与驱动程序定义的SQLSTATE的等级
- 没有数据值的（CLASS 02）驱动程序定义的Implementation-defined No Data Values状态记录为第二个等级

- 记录告警（CLASS01）的warnings状态记录为最低等级描述相同报错状态的两个以上的记录中开源group CLI配置定义的SQLSTATE的等级高于ODBC与驱动程序定义的SQLSTATE

SQLGetDiagRec

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLGetDiagRec返回包含报错告警状态信息的（与指定句柄相关的）检测数据源的记录字段的当前值

语句

```
SQLRETURN SQLGetDiagRec(
    SQLSMALLINT     HandleType,
    SQLHANDLE       Handle,
    SQLSMALLINT     RecNumber,
    SQLCHAR *       SQLState,
    SQLINTEGER *    NativeErrorPtr,
    SQLCHAR *       MessageText,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   TextLengthPtr);
```

参数

HandleType

【输入】需要检测的句柄的类型标识符标识符应为以下中的一个

- SQL_HANDLE_DBC
- SQL_HANDLE_DESC
- SQL_HANDLE_ENV
- SQL_HANDLE_STMT

Handle

【输入】HandleType指向的类型的检测数据结构的句柄HandleType为SQL_HANDLE_ENV时Handle可以是共享或非共享环境句柄

RecNumber

【输入】显示应用程序要查找的信息的状态记录状态记录从1开始编号

SQLState

【输出】对检测记录RecNumber返回五个字符的SQLSTATE代码的缓冲区指针前两个字母是CLASS后三个字母是子类SQL_DIAG_SQLSTATE 检测字段中有此信息

NativeErrorPtr

【输出】向数据源返回具体的原始报错代码的缓冲区指针SQL_DIAG_NATIVE 检测字段中有此信息

MessageText

【输出】返回检测信息文本字符串的缓冲区指针SQL_DIAG_MESSAGE_TEXT检测字段中

有此信息

MessageText为空时TextLegnthPtr返回MessageText指向的缓冲区中可返回的字符（除空终止字符外）的总数

BufferLength

【输入】*MessageText缓冲区的字符长度没有检测消息文本的最大长度

TextLegnthPtr

【输出】返回可向*MessageText返回的字符（除空终止字符外）总数的缓冲区指针返回的字符数大于BufferLength时*MessageText的检测信息文本被截断在BufferLength减去null终止字符的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE.

检测

SQLGetDiagRec不显示检测记录使用以下返回值的结果

- SQL_SUCCESS: 函数成功返回检测信息
- SQL_SUCCESS_WITH_INFO: *MessageText缓冲区不足以存储请求的检测字段而未生成检测记录为了查看是否被截断应用程序需要对比 BufferLength和*StringLengthPtr中记录的实际字节长度
- SQL_INVALID_HANDLE: Handle对HandleType指向的类型无效
- SQL_ERROR: 发生以下情况中的一个
    - RecNumber参数为负数或0

- BufferLength小于0
- 使用异步提醒时句柄的异步执行未结束

- SQL_NO_DATA: RecNumber大于为handle指定的检测记录数没由handle的检测记录时对所有RecNumber函数返回SQL_NO_DATA

说明

通常在ODBC函数返回SQL_ERROR或SQL_SUCCESS_WITH_INFO时应用程序调用SQLGetDiagRec但任何ODBC函数均可以有0个以上的检测记录因此应用程序调用任何函数后可以调用SQLGetDiagRec应用程序为了返回检测数据结构的部分或所有记录可多次调用SQLGetDiagRecODBC不限制单词可存储的检测记录的数量

SQLGetDiagRec不能用于返回检测数据结构的头部字段（RecNumber参数应大于0）对此应用程序代替SQLGetDiagRec调用SQLGetDiagField

SQLGetDiagRec只检索handle参数指定的句柄与最新的相关检测信息应用程序调用其他ODBC函数时在同一个句柄中（除SQLGetDiagRecSQLGetDiagFieldSQLError外）丢失之前调用的检测信息

在SQLGetDiagRec返回SQL_SUCCESS的期间应用程序可增加RecNumber的同时反复检索检测记录SQLGetDiagRec的调用不影响头部字段和记录字段如果应用程序没有除SQLGetDiagRecSQLGetDiagFieldSQLError函数外的其他函数的介入可再次调用SQLGetDiagRec检索记录字段为了检索SQL_DIAG_NUMBER字段的值应用程序可以条用SQLGetDiagField可多次调用SQLGetDiagRec检索可使用的检测记录的总数

HandleType 参数

各个句柄类型拥有相关的检测信息 HandleType参数指Handle类型

部分头部字段与记录字段不会对环境连接命令语描述符句柄返回

SQLGetDiagField的头部字段和记录字段表中说明了不恰当的句柄

HandleType为表示共享句柄的SQL_HANDLE_SENV时调用SQLGetDiagRec返回

SQL_INVALID_HANLDEHandleType为SQL_HANDLE_ENV时 handle可以是共享或不共享环境句柄

SQLGetEnvAttr

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLGetEnvAttr返回环境属性的当前设置

语句

```
SQLRETURN SQLGetEnvAttr(
    SQLHENV        EnvironmentHandle,
    SQLINTEGER     Attribute,
    SQLPOINTER     ValuePtr,
    SQLINTEGER     BufferLength,
    SQLINTEGER *   StringLengthPtr);
```

参数

EnvironmentHandle

【输入】环境句柄

Attribute

【输入】要检索的属性

ValuePtr

【输出】返回Attribute指定的属性的当前值的缓冲区指针ValuePtr为空时StringLengthPtr返回ValuePtr指向的缓冲区可返回的（除空终止字符外）总字节数

BufferLength

【输入】ValuePtr指字符串时此参数应为*ValuePtr的长度ValuePtr为整数时忽略BufferLength*ValuePtr（调用SQLGetEnvAttrW时）为Unicode字符串时BufferLength参数应为偶数属性值不是字符串时不使用BufferLength

StringLengthPtr

【输出】返回*ValuePtr中可返回的（不包括空终止字符）总字节数的缓冲区指针ValuePtr为空指针时不返回长度属性值为字符串可返回的字节数大于或等于BufferLength时*ValuePtr被截断为在BufferLength减去null终止符的长度并由驱动程序以null终止

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
01000	General warning	具体的驱动程序信息（函数返回SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
01004	String data, right truncated	*ValuePtr返回的值被截断为在*BufferLength减去空终止字符的长度向*StringLengthPtr返回剩余的字符串值的长度（函数返回SQL_SUCCESS_WITH_INFO）
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	SQL_ATTR_ODBC_VERSION尚未通过SQLSetEnvAttr设置如果使用SQLAllocHandleStd则不需要设置SQL_ATTR_ODBC_VERSION
HY013	Memory management error	用作参数的缓冲区大小小于0或无法访问内存
HY092	Invalid attribute/option identifier	Attribute参数指定的值在驱动程序支持的ODBC版本中无效
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考 **SQLEndTran** 函数
HYC00	Optional feature not implemented	Attribute参数指定的值在驱动程序支持的ODBC版本的ODBC环境属性中有效但驱动程序不支持

SQLSTATE	报错	说明
IM001	Driver does not support this function	与DescriptorHandle相关的驱动程序不支持此函数

说明

没有驱动程序固有的环境属性Attribute指定返回字符串的属性时ValuePtr应为返回字符串的缓冲区指针包含空终止字符的字符串的最大长度为BufferLength字节

分配和解除环境句柄的过程中可随时调用SQLGetEnvAttr直到与SQL_HANDLE_ENV的Handletype同时在EnvrionmentHandle中调用SQLFreeHandle之前维持应用程序成功分配的所有环境属性建议仅使用一个环境句柄

> Note:
> 遵守标准的应用程序支持SQL_ATTR_OUTPUT_NTS环境属性调用SQLGetEnvAttr时ODBC 3.x驱动程序管理器对此属性总是返回SQL_TRUESQL_ATTR_OUTPUT_NTS只能通过调用SQL_ATTR_OUTPUT_NTS设置为SQL_TRUE

以下表格为通过SQLGetEnvAttr可查询的属性列表

Attribute	ValuePtr内容
SQL_ATTR_CONNECTION_POOLING (ODBC 3.8)	驱动程序不支持

Attribute	ValuePtr内容
SQL_ATTR_CP_MATCH (ODBC 3.0)	驱动程序不支持

SQL_ATTR_ODBC_VERSION (ODBC 3.0)	表示特定功能以ODBC 2.x或ODBC 3.x执行的32位整数 以下值用于设置属性 SQL_OV_ODBC3_80 = 驱动程序管理器及驱动程序如下以ODBC 3.8执行 SQL_OV_ODBC3 = 驱动程序管理器及驱动程序如下以ODBC 3.0执行 SQL_OV_ODBC2 = 驱动程序管理器及驱动程序如下以ODBC 2.0执行其非常利于在ODBC 3.x 驱动程序下执行ODBC 2.x应用程序 调用拥有SQLHENV参数或返回SQLSTATE HY010(Function sequence error)的函数之前应用程序需要设置此环境属性值驱动程序需指定是否存在对这些环境标志的额外行为 • 驱动程序对DATETIMETIMESTAMP请求并返回ODBC 3.x代码值 • 驱动程序调用SQLErrorSQLGetDiagField或SQLGetDiagRec时返回ODBC 3.x SQLSTATE代码 • SQLTables的CatalogName参数允许模式检索 • 驱动程序对DATETIMETIMESTAMP请求并返回 ODBC 3.x代码值 • 驱动程序调用SQLErrorSQLGetDiagField或

Attribute	ValuePtr内容
	SQLGetDiagRec时返回ODBC 3.x SQLSTATE代码 • SQLTables的CatalogName参数允许模式检索 • 驱动程序管理器不支持C数据类型的扩展性 • 驱动程序对DATETIMETIMESTAMP请求并返回 ODBC 2.x代码值 • 驱动程序调用SQLErrorSQLGetDiagFied或 SQLGetDiagRec时返回ODBC 2.x SQLSTATE代码 • SQLTables的CatalogName参数允许模式检索 • 驱动程序管理器不支持C数据类型的扩展性
SQL_ATTR_OUTPUT_NTS (ODBC 3.0)	驱动程序不支持

SQLGetFunctions

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLGetFunctions返回驱动程序是否支持指定ODBC函数的信息在驱动程序管理器或驱动程序中实现此函数如果驱动程序实现SQLGetFunctions驱动程序管理器将调用驱动程序中的函数

语句

```
SQLRETURN SQLGetFunctions(
    SQLHDBC          ConnectionHandle,
    SQLUSMALLINT     FunctionId,
    SQLUSMALLINT *   SupportedPtr);
```

参数

ConnectionHandle

【输入】连接句柄

FunctionId

【输入】识别与SQL_API_ODBC3_ALL_FUNCTIONS或SQL_API_ALL_FUNCTIONS相关的

ODBC函数的#define值为了决定是否在ODBC 3.x与之前版本函数中支持ODBC 3.x应用程序里使用SQL_API_ODBC3_ALL_FUNCTIONS为了决定是否在ODBC 2.x与之前版本函数中支持ODBC 2.x应用程序里使用SQL_API_ALL_FUNCTIONS

识别ODBC 函数的#define值的列表参考说明部分的表

SupportedPtr

【输出】FunctionId识别单个ODBC函数时SupportedPtr指单个SQLUSMALLINT值驱动程序支持指定的函数时为SQL_TRUE反之为SQL_FALSE

FunctionId为SQL_API_ODBC3_ALL_FUNCTIONS时SupprtedPtr指与SQL_API_ODBC3_ALL_FUNCTIONS_SIZE拥有相同数量的要素的SQLSMALLINT数组该数组是为了查看是否在ODBC 3.x或之前函数中支持的可使用的4000-bit bit map由驱动程序管理器处理为了查看是否支持函数而调用SQL_FUNC_EXISTS macroODBC 3.x应用程序为了ODBC 3.x 或ODBC 2.x可使用SQL_API_ODBC3_ALL_FUNCTIONS调用SQLGetFunctions

FunctionId为SQL_API_ALL_FUNCTIONS时SupportedPtr指有100个要素的数组该数组以用于识别各ODBC函数的FunctionId的#define值进行索引部分要素暂不使用只预留如果驱动程序支持ODBC 2.x或之前的函数则要素为SQL_TRUE如果是驱动程序不支持的ODBC函数或不是ODBC函数则为SQL_FALSE

向*SupportedPtr的数组使用基于0的indexing

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE.

检测

SQLSTATE	Error	说明
01000	General warning	具体的驱动程序信息（函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	调用SQLConnectSQLBrowseConnectSQLDriverConnect之前调用了SQLGetFunctions 为ConnectionHandle调用了SQLBrowseConnect并返回了SQL_NEED_DATA SQLBrowseConnect返回SQL_SUCCESS_WIRTH_INFO或SQL_SUCCESS之前调用了 为ConnectionHanndle调用了SQLExecuteSQLExecDirectSQLMoreResults并返回了SQL_PARAM_DATA_AVAILABLE数据检索所有连续的参数之前调用了此函数

SQLSTATE	Error	说明
HY013	Memory management error	用作参数的缓冲区的大小小于0或无法访问内存
HY095	Function type out of range	Functionld值无效
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考 **SQLEndTran** 函数
HYT01	Connection timeout expired	数据源响应请求之前连接超时连接超时周期可通过SQLSetConnectAttr设置为SQL_ATTR_CONNECTION_TIMEOUT
IM001	Driver does not support this function	与DescriptorHandle相关的驱动程序不支持此函数

说明

SQLGetFunctions返回所支持的SQLGetFunctionsSQLDataSourcesSQLDrivers因为驱动程序管理器实现了这些函数存在unicode函数时驱动程序管理器映射对应unicode函数的ANSI函数；存在ANSI函数时则映射对应ANSI函数的unicode函数

以下为遵循ISO 92标准等级的函数的FunctionId有效值的目录

- SQL_API_SQLALLOCHANDLE
- SQL_API_SQLBINDCOL
- SQL_API_SQLCANCEL
- SQL_API_SQLCLOSECURSOR
- SQL_API_SQLCOLATTRIBUTE
- SQL_API_SQLCONNECT
- SQL_API_SQLCOPYDESC
- SQL_API_SQLDATASOURCES
- SQL_API_SQLDESCRIBECOL
- SQL_API_SQLDISCONNECT
- SQL_API_SQLDRIVERS
- SQL_API_SQLENDTRAN
- SQL_API_SQLEXECDIRECT
- SQL_API_SQLEXECUTE
- SQL_API_SQLFETCH
- SQL_API_SQLFETCHSCROLL
- SQL_API_SQLFREEHANDLE
- SQL_API_SQLFREESTMT
- SQL_API_SQLGETCONNECTATTR
- SQL_API_SQLGETCURSORNAME
- SQL_API_SQLGETDATA
- SQL_API_SQLGETDESCFIELD
- SQL_API_SQLGETDESCREC
- SQL_API_SQLGETDIAGFIELD
- SQL_API_SQLGETDIAGREC

- SQL_API_SQLGETENVATTR
- SQL_API_SQLGETFUNCTIONS
- SQL_API_SQLGETINFO
- SQL_API_SQLGETSTMTATTR
- SQL_API_SQLGETTYPEINFO
- SQL_API_SQLNUMRESULTCOLS
- SQL_API_SQLPARAMDATA
- SQL_API_SQLPREPARE
- SQL_API_SQLPUTDATA
- SQL_API_SQLROWCOUNT
- SQL_API_SQLSETCONNECTATTR
- SQL_API_SQLSETCURSORNAME
- SQL_API_SQLSETDESCFIELD
- SQL_API_SQLSETDESCREC
- SQL_API_SQLSETENVATTR
- SQL_API_SQLSETSTMTATTR

以下为遵循标准开源函数的FunctionId有效值的目录

- SQL_API_SQLCOLUMNS
- SQL_API_SQLSPECIALCOLUMNS
- SQL_API_SQLSTATISTICS
- SQL_API_SQLTABLES

以下为遵循标准ODBC函数的FunctionId有效值的目录

- SQL_API_SQLBINDPARAMETER
- SQL_API_SQLBROWSECONNECT
- SQL_API_SQLBULKOPERATIONS [1]
- SQL_API_SQLCOLUMNPRIVILEGES
- SQL_API_SQLDESCRIBEPARAM
- SQL_API_SQLDRIVERCONNECT
- SQL_API_SQLFOREIGNKEYS
- SQL_API_SQLMORERESULTS
- SQL_API_SQLNATIVESQL
- SQL_API_SQLNUMPARAMS
- SQL_API_SQLPRIMARYKEYS
- SQL_API_SQLPROCEDURECOLUMNS
- SQL_API_SQLPROCEDURES
- SQL_API_SQLSETPOS
- SQL_API_SQLTABLEPRIVILEGES

[1]使用ODBC 2.x驱动程序时SQLBulkOperations仅在ODBC 2.x驱动程序支持SQLSetPos并且信息类型SQL_POS_OPERATIONS属性将已设置的SQL_POS_ADD bit进行返回时才会支持并返回

对ODBC 3.8以后导入的函数的FunctionId有效值为SQL_API_SQLCANCELHANDLE [2]

[2] 驱动程序均支持SQLCancel和SQLCancelHandle时返回SQLCancelHandle支持SQLCancel但不支持SQLCancelHandle时应用程序映射SQLCancel因此仍然可对命令句柄调用SQLCancelHandle

SQL_FUNC_EXISTS macro

为了使用FunctionId参数在SQL_API_ODBC3_ALL_FUNCTIONS调用SQLGetFunctions后查看是否

支持ODBC 3.x或之前的函数使用SQL_FUNC_EXISTS(SupportedPtr, FunctionID)macro应用程序将SQLGetFuncions传输的SupportedPtr设置为SupportedPtr参数将FunctionID 参数设置为函数的#define值并使用SQL_FUNC_EXIST如果支持函数SQL_FUNC_EXIST返回SQL_TRUE反之返回SQL_FALSE

Note:

使用ODBC 2.x驱动程序时ODBC 3.x驱动程序管理器对SQLAllocHandle和SQLFreeHandle返回SQL_TRUE因为SQLAllocHandle映射于SQLAllocEnvSQLAllocConnect或SQLAllocStmtSQLFreeHandle映射于SQLFreeEnv SQLFreeConnect或SQLFreeStmt但即使对SQLFreeHandle返回SQL_TRUE也不支持以HandleType参数使用SQL_HANDLE_DESC的SQLFreeHandle由于没有映射于这种情况的ODBC 2.x 函数的函数

SQLGetGroupCount

兼容性

遵循标准：无

概要

SQLGetGroupCount返回集群组的数量

语句

```
SQLRETURN SQLGetGroupCount(
    SQLHDBC      ConnectionHandle,
    SQLINTEGER * GroupCountPtr );
```

参数

ConnectionHandle

【输入】连接句柄

GroupCountPtr

【输入】集群组的数量

返回

SQL_SUCCESS, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	Error	说明
08003	Connection not open	未连接ConnectionHandle
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
HY010	Function sequence error	此函数应在设置SQL_ATTR_LOCALITY_AWATE_TRANSACTION连接属性的情况下调用

说明

SQLGetGroupCount只能在SQL_ATTR_LOCALITY_AWARE_TRANSACTION连接属性的情况下调用

SQLGetGroupIDs

兼容性

遵循标准：无

概要

SQLGetGroupIDs返回集群组的ID

语句

```
SQLRETURN SQLGetGroupIDs(
    SQLHDBC      ConnectionHandle,
    SQLINTEGER * GroupIDArray );
```

参数

ConnectionHandle

【输入】连接句柄

GroupIDArray

【输入】集群组ID的数组

返回

SQL_SUCCESS, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	Error	说明
08003	Connection not open	未连接ConnectionHandle
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败
HY010	Function sequence error	此函数应在设置SQL_ATTR_LOCALITY_AWATE_TRANSACTION连接属性的情况下调用

说明

SQLGetGroupIDs只能在SQL_ATTR_LOCALITY_AWARE_TRANSACTION连接属性的情况下调用

GroupIDArray的element数量应与SQLGetGroupCount返回的集群组的数量相同

SQLGetGroupName

兼容性

遵循标准：无

概要

SQLGetGroupName返回对应GroupID的集群租的名称

语句

```
SQLRETURN SQLGetGroupName(
    SQLHDBC      ConnectionHandle,
    SQLINTEGER    GroupID,
    SQLCHAR     * GroupName,
    SQLSMALLINT   BufferLength,
    SQLSMALLINT * NameLengthPtr );
```

参数

ConnectionHandle

【输入】连接句柄

GroupID

【输入】集群组的ID

GroupName

【输出】返回以null终止的集群租的名称的缓冲区指针

BufferLength

【输入】*GroupName的长度

NameLengthPtr

【输出】返回可向*GroupName返回的总字节数（除null终止符外）的缓冲区指针可返回的长度大于等于BufferLength时*GroupName被截断为在BufferLength减去null终止符的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	Error	说明
01004	String dataright truncated	*GroupName的大小不足以返回集群组名称因此集群租名被截断未被截断的集群名的长度返回至*NameLengthPtr（函数返回SQL_SUCCESS_WITH_INFO）
08003	Connection not open	未连接ConnectionHandle
08S01	Communication link failure	在函数完成处理之前驱动程序与数据源之间的连接失败

SQLSTATE	Error	说明
HY010	Function sequence error	此函数应在设置SQL_ATTR_LOCALITY_AWATE_TRANSACTION连接属性的情况下调用

说明

SQLGetGroupIDs只能在SQL_ATTR_LOCALITY_AWARE_TRANSACTION连接属性的情况下调用

SQLGetInfo

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLGetInfo返回与驱动程序和数据源相关的连接的基本信息

语句

```
SQLRETURN SQLGetInfo(
    SQLHDBC         ConnectionHandle,
    SQLUSMALLINT    InfoType,
    SQLPOINTER      InfoValuePtr,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   StringLengthPtr);
```

参数

ConnectionHandle

【输入】连接句柄

InfoType

【输入】信息类型

InfoValuePtr

【输出】返回信息的缓冲区指针根据请求的InfoType返回null终止字符串SQLUSMALLINT值 SQLUINTEGER bit maskSQLUINTEGER flagSQLUINTEGER 二进制值或者SQLULEN值中的一个

InfoType参数为SQL_DRIVER_HDESC或SQL_DRIVER_HSTMT时InfoValuePtr参数为输入与输出

InfoValuePtr为空时StringLengthPtr返回可向InfoValuePtr指向的缓冲区返回的（除空终止字符外）总字节数

BufferLength

【输入】*InfoValuePtr缓冲区的长度*InfoValuePtr不是字符串或InfoValuePtr为空指针时忽略BufferLength参数驱动程序根据InfoType将*InfoValuePtr的大小视为SQLUSMALLINT或SQLUINTEGER（调用SQLGetInfoW时）InfoValuePtr为unicode字符串时BufferLength应为偶数反之返回SQLSTATE HY090

StringLengthPtr

【输出】返回可向*InfoValuePtr返回的（不包括空终止字符）总字节数的缓冲区指针

对于字符数据可返回的字节数大于或等于BufferLength时*InfoValuePtr的信息（除null终止字符与根据驱动程序的null终止长度外）截断为BufferLength字节

对于其他数据类型忽略 BufferLength驱动程序根据InfoType将*InfoValuePtr的大小视为SQLUSMALLINT或SQLUINTEGER

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, 或SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
01000	General warning	具体的驱动程序信息（函数返回SQL_SUCCESS_WITH_INFO）
01004	String data, right truncated	*InfoValuePtr的缓冲区大小不足以返回信息因此被截断未被截断的信息的长度返回于*StringLengthPtr（函数返回SQL_SUCCESS_WITH_INFO）
08003	Connection not open	InfoType请求的信息的类型连接应为打开状态ODBC中预留的信息类型SQL_ODBC_VER即使没有打开状态的连接也可返回
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	为StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults并返回了SQL_PARAM_DATA_AVAILABLE数据检索所有连续的参数之前调用了此函数
HY013	Memory management error	作为参数的缓冲区的大小小于0或无法访问内存

SQLSTATE	报错	说明
HY024	Invalid attribute value	infoType参数为SQL_DRIVER_HSTMTInfoValuePtr指向的值为无效的命令语句柄 InfoType参数为SQL_DRIVER_HDESCInfoValuePtr指向的值为无效的描述符句柄
HY090	Invalid string or buffer length	BufferLength参数值小于0 BufferLength参数为奇数InfoValuePtr为unicode数据类型
HY096	Information type out of range	InfoType参数值在驱动程序支持的ODBC版本里无效
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran**函数
HYC00	Optional field not implemented	InfoType参数指定的值为驱动程序不支持的驱动程序特定值
HYT01	Connection timeout expired	数据源响应请求之前连接超时其限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置

SQLSTATE	报错	说明
IM001	Driver does not support this function	与DescriptorHandle相关的驱动程序不支持此函数

说明

以下说明当前定义的信息类型ODBC预留信息类型的范围驱动程序开发人员应该在开放组合中预留自身驱动程序的使用值SQLGetInfo不执行unicode转换或驱动程序定义的InfoTypes的thunking向*InfoValuePtr返回的信息类型取决于请求的InfoTypes

SQLGetInfo返回以下5个类型中的一个信息

- 空终止字符串
- SQLUSMALLINT值
- SQLUINTEGER bit mask
- SQLUINTEGER 值
- SQLUINTEGER 二进制值

应用程序根据向*InfoValuePtr返回的值进行casting

驱动程序应返回以下表中定义的信息类型的值如果数据类型不适用于驱动程序或数据源时驱动程序应返回以下表中记录的一个值

Character string ("Y" or "N")

"N"

Character string (not "Y" or "N")

Empty string

SQLUSMALLINT

0

SQLUINTEGER bit mask或SQLUINTEGER 二进制值

0L

例如数据源不支持过程时SQLGetInfo对过程相关的InfoType值返回以下表中记录的值

SQL_PROCEDURES

"N"

SQL_ACCESSIBLE_PROCEDURES

"N"

SQL_MAX_PROCEDURE_NAME_LEN

0

SQL_PROCEDURE_TERM

Empty string

SQLGetInfo对存在于ODBC预留的信息类型范围内但未定义驱动程序支持的ODBC版本的InfoType值返回SQLSTATE HY096为了查看ODBC版本驱动程序以应用程序的SQL_DRIVER_ODBC_VER信息类型调用SQLGetInfo的同时进行编译SQLGetInfo对存在于驱动程序为了常用而预留的信息类型范围内但驱动程序不支持的InfoType值返回SQLSTATE HYC00

除了返回驱动程序管理器版本的 InfoType为SQL_ODBC_VER的情况外所有SQLGetInfo应打开连

接

驱动程序信息

InfoType参数的以下值返回激活状态的命令语数据源名称标准接口级别等ODBC信息

- SQL_ACTIVE_ENVIRONMENTS
- SQL_ASYNC_DBC_FUNCTIONS
- SQL_ASYNC_MODE
- SQL_ASYNC_NOTIFICATION
- SQL_BATCH_ROW_COUNT
- SQL_BATCH_SUPPORT
- SQL_DATA_SOURCE_NAME
- SQL_DRIVER_AWARE_POOLING_SUPPORTED
- SQL_DRIVER_HDBC
- SQL_DRIVER_HDESC
- SQL_DRIVER_HENV
- SQL_DRIVER_HLIB
- SQL_DRIVER_HSTMT
- SQL_DRIVER_NAME
- SQL_DRIVER_ODBC_VER
- SQL_DRIVER_VER
- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2

- SQL_FILE_USAGE
- SQL_GETDATA_EXTENSIONS
- SQL_INFO_SCHEMA_VIEWS
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_MAX_ASYNC_CONCURRENT_STATEMENTS
- SQL_MAX_CONCURRENT_ACTIVITIES
- SQL_MAX_DRIVER_CONNECTIONS
- SQL_ODBC_INTERFACE_CONFORMANCE
- SQL_ODBC_STANDARD_CLI_CONFORMANCE
- SQL_ODBC_VER
- SQL_PARAM_ARRAY_ROW_COUNTS
- SQL_PARAM_ARRAY_SELECTS
- SQL_ROW_UPDATES
- SQL_SEARCH_PATTERN_ESCAPE
- SQL_SERVER_NAME
- SQL_STATIC_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES2

> Note:
>
> 执行SQLGetInfo时驱动程序可通过传递信息或最小化服务器请求的次数来提高性能

DBMS产品信息

如下InfoType参数值返回DBMS名称或信息等DBMS产品信息

- SQL_DATABASE_NAME
- SQL_DBMS_NAME
- SQL_DBMS_VER

数据源信息

以下InfoType参数值返回游标特性与事务功能等数据源的信息

- SQL_ACCESSIBLE_PROCEDURES
- SQL_ACCESSIBLE_TABLES
- SQL_BOOKMARK_PERSISTENCE
- SQL_CATALOG_TERM
- SQL_COLLATION_SEQ
- SQL_CONCAT_NULL_BEHAVIOR
- SQL_CURSOR_COMMIT_BEHAVIOR
- SQL_CURSOR_ROLLBACK_BEHAVIOR
- SQL_CURSOR_SENSITIVITY
- SQL_DATA_SOURCE_READ_ONLY
- SQL_DEFAULT_TXN_ISOLATION
- SQL_DESCRIBE_PARAMETER
- SQL_MULT_RESULT_SETS
- SQL_MULTIPLE_ACTIVE_TXN
- SQL_NEED_LONG_DATA_LEN
- SQL_NULL_COLLATION
- SQL_PROCEDURE_TERM
- SQL_SCHEMA_TERM

- SQL_SCROLL_OPTIONS
- SQL_TABLE_TERM
- SQL_TXN_CAPABLE
- SQL_TXN_ISOLATION_OPTION
- SQL_USER_NAME

支持的SQL

以下InfoType参数值返回数据源支持的SQL命令语信息

- SQL_AGGREGATE_FUNCTIONS
- SQL_ALTER_DOMAIN
- SQL_ALTER_SCHEMA
- SQL_ALTER_TABLE
- SQL_ANSI_SQL_DATETIME_LITERALS
- SQL_CATALOG_LOCATION
- SQL_CATALOG_NAME
- SQL_CATALOG_NAME_SEPARATOR
- SQL_CATALOG_USAGE
- SQL_COLUMN_ALIAS
- SQL_CORRELATION_NAME
- SQL_CREATE_ASSERTION
- SQL_CREATE_CHARACTER_SET
- SQL_CREATE_COLLATION
- SQL_CREATE_DOMAIN
- SQL_CREATE_SCHEMA

- SQL_CREATE_TABLE
- SQL_CREATE_TRANSLATION
- SQL_DDL_INDEX
- SQL_DROP_ASSERTION
- SQL_DROP_CHARACTER_SET
- SQL_DROP_COLLATION
- SQL_DROP_DOMAIN
- SQL_DROP_SCHEMA
- SQL_DROP_TABLE
- SQL_DROP_TRANSLATION
- SQL_DROP_VIEW
- SQL_EXPRESSIONS_IN_ORDERBY
- SQL_GROUP_BY
- SQL_IDENTIFIER_CASE
- SQL_IDENTIFIER_QUOTE_CHAR
- SQL_INDEX_KEYWORDS
- SQL_INSERT_STATEMENT
- SQL_INTEGRITY
- SQL_KEYWORDS
- SQL_LIKE_ESCAPE_CLAUSE
- SQL_NON_NULLABLE_COLUMNS
- SQL_SQL_CONFORMANCE
- SQL_OJ_CAPABILITIES
- SQL_ORDER_BY_COLUMNS_IN_SELECT
- SQL_OUTER_JOINS

- SQL_PROCEDURES
- SQL_QUOTED_IDENTIFIER_CASE
- SQL_SCHEMA_USAGE
- SQL_SPECIAL_CHARACTERS
- SQL_SUBQUERIES
- SQL_UNION

SQL限制

以下InfoType参数返回适用于相同标识符的最大长度和选择目录中的最大列数等SQL命令中识别符或子句的约束驱动程序或数据源可赋予约束

- SQL_MAX_BINARY_LITERAL_LEN
- SQL_MAX_CATALOG_NAME_LEN
- SQL_MAX_CHAR_LITERAL_LEN
- SQL_MAX_COLUMN_NAME_LEN
- SQL_MAX_COLUMNS_IN_GROUP_BY
- SQL_MAX_COLUMNS_IN_INDEX
- SQL_MAX_COLUMNS_IN_ORDER_BY
- SQL_MAX_COLUMNS_IN_SELECT
- SQL_MAX_COLUMNS_IN_TABLE
- SQL_MAX_CURSOR_NAME_LEN
- SQL_MAX_IDENTIFIER_LEN
- SQL_MAX_INDEX_SIZE
- SQL_MAX_PROCEDURE_NAME_LEN
- SQL_MAX_ROW_SIZE

- SQL_MAX_ROW_SIZE_INCLUDES_LONG
- SQL_MAX_SCHEMA_NAME_LEN
- SQL_MAX_STATEMENT_LEN
- SQL_MAX_TABLE_NAME_LEN
- SQL_MAX_TABLES_IN_SELECT
- SQL_MAX_USER_NAME_LEN

标量函数信息

以下InfoType参数返回数据源与驱动程序支持的标量函数信息

- SQL_CONVERT_FUNCTIONS
- SQL_NUMERIC_FUNCTIONS
- SQL_STRING_FUNCTIONS
- SQL_SYSTEM_FUNCTIONS
- SQL_TIMEDATE_ADD_INTERVALS
- SQL_TIMEDATE_DIFF_INTERVALS
- SQL_TIMEDATE_FUNCTIONS

转换信息

以下InfoType参数值返回可使用CONVERT标量函数将数据源转换为指定SQL数据类型的SQL数据类型目录

- SQL_CONVERT_BIGINT
- SQL_CONVERT_BINARY
- SQL_CONVERT_BIT

- SQL_CONVERT_CHAR
- SQL_CONVERT_DATE
- SQL_CONVERT_DECIMAL
- SQL_CONVERT_DOUBLE
- SQL_CONVERT_FLOAT
- SQL_CONVERT_INTEGER
- SQL_CONVERT_INTERVAL_YEAR_MONTH
- SQL_CONVERT_INTERVAL_DAY_TIME
- SQL_CONVERT_LONGVARBINARY
- SQL_CONVERT_LONGVARCHAR
- SQL_CONVERT_NUMERIC
- SQL_CONVERT_REAL
- SQL_CONVERT_SMALLINT
- SQL_CONVERT_TIME
- SQL_CONVERT_TIMESTAMP
- SQL_CONVERT_TINYINT
- SQL_CONVERT_VARBINARY
- SQL_CONVERT_VARCHAR

ODBC 3.x中增加的信息类型

ODBC 3.x中增加了如下InfoType参数值

- SQL_ACTIVE_ENVIRONMENTS
- SQL_AGGREGATE_FUNCTIONS
- SQL_ALTER_DOMAIN

- SQL_ALTER_SCHEMA
- SQL_ANSI_SQL_DATETIME_LITERALS
- SQL_ASYNC_DBC_FUNCTIONS
- SQL_ASYNC_MODE
- SQL_ASYNC_NOTIFICATION
- SQL_BATCH_ROW_COUNT
- SQL_BATCH_SUPPORT
- SQL_CATALOG_NAME
- SQL_COLLATION_SEQ
- SQL_CONVERT_INTERVAL_YEAR_MONTH
- SQL_CONVERT_INTERVAL_DAY_TIME
- SQL_CREATE_ASSERTION
- SQL_CREATE_CHARACTER_SET
- SQL_CREATE_COLLATION
- SQL_CREATE_DOMAIN
- SQL_CREATE_SCHEMA
- SQL_CREATE_TABLE
- SQL_CREATE_TRANSLATION
- SQL_CURSOR_SENSITIVITY
- SQL_DDL_INDEX
- SQL_DESCRIBE_PARAMETER
- SQL_DM_VER
- SQL_DRIVER_AWARE_POOLING_SUPPORTED
- SQL_DRIVER_HDESC
- SQL_DROP_ASSERTION

- SQL_DROP_CHARACTER_SET
- SQL_DROP_COLLATION
- SQL_DROP_DOMAIN
- SQL_DROP_SCHEMA
- SQL_DROP_TABLE
- SQL_DROP_TRANSLATION
- SQL_DROP_VIEW
- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
- SQL_INFO_SCHEMA_VIEWS
- SQL_INSERT_STATEMENT
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_MAX_ASYNC_CONCURRENT_STATEMENTS
- SQL_MAX_IDENTIFIER_LEN
- SQL_PARAM_ARRAY_ROW_COUNTS
- SQL_PARAM_ARRAY_SELECTS
- SQL_STATIC_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES2
- SQL_XOPEN_CLI_YEAR

ODBC 3.x中改名的信息类型

ODBC 3.x中变更了如下InfoType参数值的名称

- SQL_ACTIVE_CONNECTIONS: SQL_MAX_DRIVER_CONNECTIONS
- SQL_ACTIVE_STATEMENTS: SQL_MAX_CONCURRENT_ACTIVITIES
- SQL_MAX_OWNER_NAME_LEN: SQL_MAX_SCHEMA_NAME_LEN
- SQL_MAX_QUALIFIER_NAME_LEN: SQL_MAX_CATALOG_NAME_LEN
- SQL_ODBC_SQL_OPT_IEF: SQL_INTEGRITY
- SQL_OWNER_TERM: SQL_SCHEMA_TERM
- SQL_OWNER_USAGE: SQL_SCHEMA_USAGE
- SQL_QUALIFIER_LOCATION: SQL_CATALOG_LOCATION
- SQL_QUALIFIER_NAME_SEPARATOR: SQL_CATALOG_NAME_SEPARATOR
- SQL_QUALIFIER_TERM: SQL_CATALOG_TERM
- SQL_QUALIFIER_USAGE: SQL_CATALOG_USAGE

ODBC 3.x中不再使用的信息类型

以下InfoType参数值是在ODBC 3.x驱动程序中不再使用但为了兼容ODBC 2.x应用程序而继续支持的信息类型

- SQL_FETCH_DIRECTION
- SQL_LOCK_TYPES
- SQL_ODBC_API_CONFORMANCE
- SQL_ODBC_SQL_CONFORMANCE
- SQL_POS_OPERATIONS
- SQL_POSITIONED_STATEMENTS
- SQL_SCROLL_CONCURRENCY
- SQL_STATIC_SENSITIVITY

SQLGetStmtAttr

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLGetStmtAttr返回命令属性的当前设置

语句

```
SQLRETURN SQLGetStmtAttr(
    SQLHSTMT        StatementHandle,
    SQLINTEGER      Attribute,
    SQLPOINTER      ValuePtr,
    SQLINTEGER      BufferLength,
    SQLINTEGER *    StringLengthPtr);
```

参数

StatementHandle

【输入】命令语句柄

Attribute

【输入】要检索的属性

ValuePtr

【输出】返回Attribute中指定的属性值的缓冲区指针ValuePtr为空时StringLengthPtr返回可向ValuePtr指向的缓冲区返回的（除空终止字符）总字节数

BufferLength

【输入】Attribute为ODBC定义的属性ValuePtr指向字符串或二进制缓冲区时此参数应为*ValuePtr的长度Attribute为ODBC定义的属性*ValuePtr为整数时忽略BufferLength（调用SQLGetStmtAttrW时）*ValuePtr为Unicode字符串时BufferLength参数应为偶数Attribute为驱动程序定义属性时应用程序通过设置BufferLength参数在驱动程序管理器表明属性的特性BufferLength为以下值

• *ValuePtr为字符串指针时BufferLength是字符串的长度或SQL_NTS

• *ValuePtr为二进制缓冲区指针时应用程序把SQL_LEN_BINARY_ATTR(length) macro结果存储于BufferLength其在BufferLength存储负数

• *ValuePtr为非字符串或二进制字符串的其他值的指针时BufferLength是SQL_IS_POINTER的值

• *ValuePtr为固定长度数据类型时BufferLength拥有SQL_IS_INTEGER 或SQL_IS_UINTEGER

StringLengthPtr

【输出】返回可向*ValuePtr返回的总字节数的指针（不包括空终止字符 ）ValuePtr为空指针时不返回长度属性值为字符串时如果可返回的字节数大于或等于BufferLength*ValuePtr的数据（除空终止字符与驱动程序的null终止外）被截断为在BufferLength减去null终止符的长度并由驱动程序null终止

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, 或SQL_INVALID_HANDLE.

检测

SQLSTATE	Error	说明
01000	General warning	具体的驱动程序信息（函数返回 SQL_SUCCESS_WITH_INFO）
01004	String data, right truncated	*ValuePtr返回的数据截断为在BufferLength减去null终止字符的长度未被截断的字符串的值返回至*StringLengthPtr (函数返回 SQL_SUCCESS_WITH_INFO)
24000	Invalid cursor state	Attribute参数为SQL_ATTR_ROW_NUBER未打开游标或游标位于结果集的开始前或结束后
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	Error	说明
HY010	Function sequence error	StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLGetStmtAttr时异步执行的函数还在执行中 对StatementHandle调用了异步执行的函数调用此函数时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution参数或column的数据之前调用了该函数
HY013	Memory management error	用作参数的缓冲区的大小小于0或无法访问内存
HY090	Invalid string or buffer length	*ValuePtr为字符串BuferLength小于0但不等于SQL_NTS
HY092	Invalid attribute/option identifier	Attribute参数的指定的值在驱动程序支持的ODBC版本中无效
HY109	Invalid cursor position	Attribute参数为SQL_ATTR_ROW_NUMBER已删除或无法读取行

SQLSTATE	Error	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran**函数
HYC00	Optional field not implemented	Attribute参数中指定的值为驱动程序支持的ODBC版本的有效的ODBC命令属性但驱动程序不支持
HYT01	Connection timeout expired	数据源响应请求之前超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	与DescriptorHandle相关的驱动程序不支持函数

说明

SQLGetStmtAttr向*ValuePtr返回attribute中指定的命令属性值返回的值可以为SQLULEN值或空终止字符串此值为SQLULEN时部分驱动程序记录缓冲区的低位32bit或16bit维持高位bit因此应用程序使用SQLULEN的缓冲区调用函数之前应初始化为0而且不使用BufferLength和StringLengthPtr此值为空终止字符字符串时应用程序指定BufferLength参数的最大字符串长度驱动程序向*StringLengthPtr缓冲区返回字符串的长度

使用ODBC 2.x驱动程序的应用程序调用SQLGetStmtAttr时 应将SQLGetStmtAttr映射到驱动程序管理器的SQLGetStmtOption

以下命令属性为只读属性因此可以用SQLGetStmtAttr检索但不能设置为SQLSetStmtAttr

- SQL_ATTR_IMP_PARAM_DESC
- SQL_ATTR_IMP_ROW_DESC
- SQL_ATTR_ROW_NUMBER

命令属性

以下表格说明当前定义的属性与介绍其属性的版本

属性	ValuePtr内容
SQL_ATTR_APP_PARAM_DESC (ODBC 3.0)	后续向命令语句柄的SQLExecute和SQLExecDirect调用的APD句柄此属性的初始值为 令语时内部分配的描述符当此属性设置为SQL_NULL_DESC或句柄原本分配的描述符 前命令语相关的指定分配的APD句柄 命令语句柄以内部分配的APD句柄运行 此属性不能设置为其他命令内部分配的描述符或同一个命令中包含的其他描述符句 的描述符句柄不能关联于一个命令或描述符句柄

属性	ValuePtr内容
SQL_ATTR_APP_ROW_DESC (ODBC 3.0)	用于命令语句柄的下一个fetch的ARD句柄此属性的初始值为最初分配命令语时内部 符当此属性设置为SQL_NULL_DESC或句柄原本分配的描述符时分离与之前命令语相 配的ARD句柄 命令语句柄以内部分配的ARD句柄运行 此属性不能设置为其他命令内部分配的描述符或同一个命令中包含的其他描述符句 的描述符句柄不能关联于一个命令语或描述符句柄
SQL_ATTR_ASYNC_ENABLE (ODBC 1.0)	驱动程序不支持
SQL_ATTR_ASYNC_STMT_EVENT (ODBC 3.8)	驱动程序不支持
SQL_ATTR_ASYNC_STMT_PCALLBACK (ODBC3.8)	驱动程序不支持
SQL_ATTR_ASYNC_STMT_PCONTEXT (ODBC 3.8)	驱动程序不支持

SQL_ATTR_CONCURRENCY (ODBC 2.0)	SQL_ATTR_CONCURRENCY的默认值为SQL_CONCUR_READ_ONLY SQL_ATTR_CURSOR_TYPE attribute变更为SQL_ATTR_CONCURRENCY不支持的值时 SQL_ATTR_CONCURRENCY 的值在执行时会变更调用SQLExecDirect或SQLPrepare时 驱动程序支持SELECT FOR UPDATE命令语执行这些命令语的过程中如果 SQL_ATTR_CONCURRENCY变更为SQL_CONCUR_READ_ONLY则报错 如果SQL_ATTR_CONCURRENCY值变更为驱动程序支持的SQL_ATTR_CURSOR_TYPE值 值则SQL_ATTR_CURSOR_TYPE值在执行时变更并在执行SQLExecDirect或SQLPrepare SQLSTATE 01S02 (Option value changed) 如果数据源不支持明示的并发性时驱动程序将代替为其他的并发性并返回SQLSTATE (Option value changed)驱动程序将SQL_CONCUR_VALUES替代为SQL_CONCUR_ROWV 外SQL_CONCUR_LOCK以SQL_CONCUR_ROWVERSQL_CONCUR_VALUES顺序替代替代 直到执行时间点之前无法查看. • SQLULEN：明示游标并发性的值 • SQL_CONCUR_READ_ONLY：只读游标不允许更新 • SQL_CONCUR_LOCK：游标使用能够更新行的最小程度的level locking • SQL_CONCUR_POWVER：游标使用如SQLBase ROWID或Sybase TIMESTA

属性	ValuePtr内容
	对比行版本的并发性 • SQL_CONCUR_VALUES：游标使用控制并对比的并发行
SQL_ATTTR_CURSOR_SCROLLABLE (ODBC 3.0)	• SQL_NONSCROLLABLE：是默认值命令语句柄不要求滚动游标如果应用程 柄中调用SQLFetchScroll则FetchOrientation的唯一有效值为SQL_FETCH_ • SQLULEN：明示应用程序所要求的支持level的值如果设置此属性则影响 SQLExecDirect和SQLExecute • SQL_SCROLLABLE：命令语句柄要求滚动游标调用SQLFetchScroll时应用 序模式外的其他模式下定位游标并指定FetchOrientation的有效值

属性	ValuePtr内容
SQL_ATTR_CURSOR_SENSITIVITY (ODBC 3.0)	• SQLULEN：是否让命令语句柄的游标查看其他游标结果集变化的值设置之后的SQLExecDirect和SQLExecute的调用应用程序为了获取应用程序最始状态或状态可重新读取此属性值 • SQL_UNSPECIFIED：是默认值未指定游标的类型未指定是否让命令语句到其他游标结果集变化的值命令语句柄的游标可能看不到任何或只能看看到全部 • SQL_INSENSITIVE：命令语句柄的游标无法看到其他游标变更的内容只insensitive游标为只读游标属于只读的支持并发行的静态游标 • SQL_SENSITIVE：命令语句柄的所有游标能看到其他游标生成的结果集

属性	ValuePtr内容
SQL_ATTR_CURSOR_TYPE (ODBC 2.0)	默认值为SQL_CURSOR_FORWARD_ONLY此属性不能在准备的(prepared) SQL命令语 如果数据源不支持指定的游标类型则驱动程序代替为其他类型并返回SQLSTATE 01S value changed)驱动程序将混合游标或动态游标代替为keyset-driven或静态游标驱动 游标替代keyset-driven游标 • SQLULEN：指定游标类型的值 • SQL_CURSOR_FORWARD_ONLY：游标只能向前移动 • SQL_CURSOR_KEYSET_DRIVEN：驱动程序只存储并使用与SQL_ATTR_KE 执行的行数相同数量的的KEY • SQL_CURSOR_DYNAMIC：驱动程序存储并使用仅用于行集合的行的KEY
SQL_ATTR_ENABLE_AUTO_IPD (ODBC 3.0)	驱动程序不支持
SQL_ATTR_FETCH_BOOKMARK_PTR (ODBC 3.0)	驱动程序不支持
SQL_ATTR_IMP_PARAM_DESC (ODBC 3.0)	IPD的句柄此属性值为最初分配命令语时分配的描述符应用程序不能设置此属性值 此属性可以通过调用SQLGetStmtAttr查询但不能通过SQLSetStmtAttr设置

属性	ValuePtr内容
SQL_ATTR_IMP_ROW_DESC (ODBC 3.0)	IRD的句柄此属性值为最初分配命令语时分配的描述符应用程序不能设置此属性值 此属性可以通过调用SQLGetStmtAttr查询不能通过SQLSetStmtAttr设置
SQL_ATTR_KEYSET_SIZE (ODBC 2.0)	驱动程序不支持
SQL_ATTR_MAX_LENGTH (ODBC 1.0)	驱动程序不支持

属性	ValuePtr内容
SQL_ATTR_MAX_ROWS (ODBC 1.0)	此属性用于减少网络消耗理论上生成结果集时适用此属性限制第一个ValuePtr 行的 果集的行数大于ValuePtr时结果集合会减少 SQL_ATTR_MAX_ROWS适用于命令语的所有结果集包含目录函数返回的结果集 SQL_ATTR_MAX_ROWS设置游标行数的最大值 无法保证是否准确执行SQL_ATTR_MAX_ROWS时（数据源无法限制结果集大小时）驱 能对SQLFetch或SQLFetchScroll模仿SQL_ATTR_MAX_ROWS操作 由驱动程序定义是否对SELECT语句（类似目录函数）外的其他命令语适用 SQL_ATTR_MAX_ROWS 此属性值可设置于打开的游标但不会立即生效此时驱动程序返回SQLSTATE 01S02(O changed) 把属性设置为原来的值 • SQLULEN：对应SELECT命令语返回的最大行数的值*ValuePtr为0时驱动 有行

属性	ValuePtr内容
SQL_ATTR_METADATA_ID (ODBC 3.0)	SQL_TRUE时目录函数的字符串参数识别为标识符此时不区分大小写对未指定范围的 程序将去掉空白后转换为大写后返回字符串对指定范围的字符串驱动程序去掉前置空 空白后直接使用分隔符之间的字符此参数中的一个设置为空指针时函数返回SQL_ER SQLSTATE HY009 (Invalid use of null pointer) SQL_FALSE时 不将目录函数的字符串参数识别为标识符此时区分大小写根据参数参 包含检索模式字符串 默认值为SQL_FALSE SQLTables的TableType参数值不受此属性的影响 SQL_ATTR_METADATA_ID可在连接级别中设置（这个和SQL_ATTR_ASYNC_ENABLE 是命令属性也是连接属性） 详细内容参考**目录函数的参数** • SQLULEN：设置怎样使用目录函数的字符串参数的值
SQL_ATTR_NOSCAN (ODBC 1.0)	驱动程序不支持

属性	ValuePtr内容
SQL_ATTR_PARAM_BIND_OFFSET_PTR (ODBC 3.0)	绑定offset总是向 SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTRSQL_DESC_OCTET_LENGTH_PTR字 如果更改offset值则新的值也直接添加到描述符字段值新的offset值不会加到之前的o 通过设置此命令属性设置APD头部的SQL_DESC_BIND_TYPE字段 • SQLULEN*：表示添加修改动态参数绑定的指针的offset的值 此字段不是 程序反向参考指针并反向参考描述符记录 (SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTRSQL_DESC_OCTET_LEN 中的延迟字段的各个值绑定时使用新的指针值默认值为NULL
SQL_ATTR_PARAM_BIND_TYPE (ODBC 3.0)	此字段设置为选择列式绑定的SQL_PARAM_BIND_BY_COLUMN（默认值） 选择行式绑定时此字段应设置为结构体的长度或动态参数集合绑定的缓冲区实例此长 于绑定参数及结构体填充的所有空间或绑定参数地址以指定长度增加时结果应指向下 的开始使用ANSI C的sizeof运算符时保证其操作 通过设置此命令属性设置APD头部里的SQL_DESC_BIND_TYPE字段 • SQLULEN：表示动态参数中使用的绑定方向

属性	ValuePtr内容
SQL_ATTR_PARAM_OPERATION_PTR (ODBC 3.0)	可通过设置APD中的SQL_DESC_ARRAY_STATUS_PTR指向的数组状态值忽略处理过程 合此状态值为SQL_PARAM_PROCEED或没有设置数组要素时处理参数集合 此命令属性可以设置为空指针此时驱动程序不返回参数状态值可随时设置此属性但在 个SQLExecDirect或SQLExecute之前无法使用新的值 没有可绑定的参数时忽略此属性 通过设置此命令属性设置APD头部中的SQL_DESC_ARRAY_STATUS_PTR字段 • SQLUSMALLINT*：指用于执行SQL命令语过程中忽略参数的SQLUSMALL 组的值其各个值为（为了执行参数）SQL_PARAM_PROCEE或（为了忽略 SQL_PARAM_IGNORE

SQL_ATTR_PARAM_STATUS_PTR (ODBC 3.0)	此命令属性可以设置为空指针此时驱动程序不返回参数的状态值可以随时设置此属性 用下一个SQLExecDirect或SQLExecute之前无法使用新的值设置此属性可以影响驱动 数的操作 通过设置此命令属性设置IPD头部的SQL_DESC_ARRAY_STATUS_PTR字段 • SQLUSMALLINT*：指调用SQLExecute或SQLExecDirect后包含各个行状态 SQLUSMALLINT数组值的值此字段只在PARAMSET_SIZE大于1时使用状态 含其他值 • SQL_PARAM_SUCCESS：SQL命令语对此参数集合成功执行 • SQL_PARAM_SUCCESS_WITH_INFO：SQL命令语对此参数集合成功执行但 据结构体有告警 • SQL_PARAM_ERROR：处理参数集合时发生错误详细的错误信息存储于相 构体中 • SQL_PARAM_UNUSED：由于中断部分之前参数集合的处理或 SQL_ATTR_PARAM_OPERATION_PTR指定的数组参数集被设置为 SQL_PARAM_IGNORE而不使用参数集合 • SQL_PARAM_DIAG_UNAVAILABLE：为生成报错信息等级因此驱动程序将

属性	ValuePtr内容
	一个整体单元来处理
SQL_ATTR_PARAMS_PROCESSED PTR (ODBC 3.0)	如果是空指针则不返回 通过设置此命令属性设置IPD头部的SQL_DESC_ROWS_PROCESSED_PTR字段 如果调用SQLExecDirect或SQLExecute填充此属性设置的缓冲区时不返回SQL_SUCCES SQL_SUCCESS_WITH_INFO则不定义缓冲区的内容 • SQLULEN*：作为指返回已处理的参数集合数量的缓冲区的记录字段包括
SQL_ATTR_PARAMSET_SIZE (ODBC 3.0)	如果没有绑定的参数则忽略此属性 通过设置此命令属性设置APD头部的SQL_DESC_ARRAY_SIZE字段 • SQLULEN：指定每个参数值数量的值SQL_ATTR_PARAMSET_SIZE大于1时 SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTRSQL_DESC_OCTET_LEN 数组每个数组常数等于字段的值

属性	ValuePtr内容
SQL_ATTR_QUERY_TIMEOUT (ODBC 1.0)	如果指定的超时值超过数据源的最大值或小于最小值时SQLSetStmtAttr将替代值并返 SQLSTATE 01S02(Option value changed) 应用程序中即使SELECT命令语超时如果再次使用命令也不用重新调用SQLCloseCurs 设置此超时命令属性对同步方式与异步方式均有效 • SQLULEN：执行SQL命令语后向应用程序返回之前等待的秒单位的时间V （默认值） 时没有超时
SQL_ATTR_RETRIEVE_DATA (ODBC 2.0)	驱动程序不支持
SQL_ATTR_ROW_ARRAY_SIZE (ODBC 3.0)	指定的行集合大小超过数据源支持的行集合大小时驱动程序将替代值并返回SQLSTA (Option value changed) 通过设置此命令属性设置ARD头部的SQL_DESC_ARRAY_SIZE字段 • SQLULEN：调用SQLFetch或SQLFetchScroll后指定其返回的行数的值也是 SQLBulkOperations中用于批量书签运算的书签数组的行数默认值为1

属性	ValuePtr内容
SQL_ATTR_ROW_BIND_OFFSET_PTR (ODBC 3.0)	通过设置此命令属性设置ARD头部的SQL_DESC_BIND_OFFSET_PTR字段 • SQLULEN：为了表示列数据绑定的变更而增加的offset的值此字段不为n 序将反向参考指针并在描述符记录 (SQL_DESC_DATA_PTRSSQL_DESC_INDICATOR_PTRSQL_DESC_OCTET_LE 字段中追加反向参考值之后绑定时使用新的指针值默认设置为空值
SQL_ATTR_ROW_BIND_TYPE (ODBC 1.0)	如果指定长度必须包括所有要绑定的column与要绑定的column的地址长度按照特定 填充结构体或缓冲区的空间当在ANSI C 的结构体或共用体中使用sizeof运算时保证其 SQLFetch和SQLFetchScroll的默认绑定方向为列式绑定 通过设置此命令属性设置ARD头部的SQL_DESC_BIND_TYPE字段 • SQLULEN：相关命令语中调用SQLFetch或SQLFetchScroll时设置绑定方向 设置为SQL_BIND_BY_COLUMN值则采用列式绑定如果设置为一个结构体 column绑定的缓冲区实例长度则采用行式绑定

属性	ValuePtr内容
SQL_ATTR_ROW_NUMBER (ODBC 2.0)	此属性可通过调用SQLGetStmAttr查询不是通过调用SQLSetStmtAttr设置 • SQLULEN：整个结果集合中的当前行的顺序无法确定当前的行数或当前 驱动程序返回0
SQL_ATTR_ROW_OPERATION_PTR (ODBC 3.0)	驱动程序不支持
SQL_ATTR_ROW_STATUS_PTR (ODBC 3.0)	此属性可设置为空指针此时驱动程序不返回行状态值可以随时设置此属性但直到调 SQLBulkOperationsSQLFetchSQLFetchScroll或SQLSetPos之前不使用新的值 通过设置此命令属性设置IRD头部的SQL_DESC_ARRAY_STATUS_PTR字段 此属性在ODBC 2.x 驱动程序中映射到SQLExetendedFetch的rgbRowStatus数组 • SQLUSMALLINT*：指包含调用SQLFetch或SQLFetchScroll后的行状态值 SQLUSMALLINT数组值的值数组拥有与行集合包含的行数相同的要素

属性	ValuePtr内容
SQL_ATTR_ROWS_FETCHED_PTR (ODBC 3.0)	通过设置此命令属性设置IRD头部的SQL_DESC_ROWS_PROCESSED_PTR字段 此属性调用填充指缓冲区的SQLFetch或SQLFetchScroll时 如果不返回SQL_SUCCESS SQL_SUCCESS_WITH_INFO则不定义缓冲区的内容 • SQLULEN*：指向调用SQLFetch或SQLFetchScroll后返回获取的行数的缓 Operations参数设置SQL_REFRESH后调用SQLSetPos或调用SQLBulkOper 批量操作的行数此行数包含报错行
SQL_ATTR_SIMULATE_CURSOR (ODBC 2.0)	驱动程序不支持
SQL_ATTR_USE_BOOKMARKS (ODBC 2.0)	驱动程序不支持

SQLGetStmtOption

兼容性

导入版本：ODBC 1.0

遵从标砖：无

概要

ODBC 3.x中SQLGetStmtOption函数被替代为SQLGetStmtAttr函数

详细内容参考**SQLGetStmtAttr**

SQLGetSuitableGroupID

兼容性

遵从标砖：无

概要

SQLGetSuitableGroupID返回可在集群系统中最优化执行准备的命令的集群组的ID

语句

```
SQLRETURN SQLGetSuitableGroupID(
    SQLHSTMT     StatementHandle,
    SQLINTEGER * GroupIDPtr );
```

参数

ConnectionHandle

【输入】命令语句柄

GroupIDPtr

【输入】集群租的ID

返回

SQL_SUCCESS, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
HY010	Function sequence error	在SQLPrepare之前调用了此函数 应在设置SQL_ATTR_LOCALITY_AWARE_TRANSACTION连接属性的情况下调用此函数

说明

已设置SQL_ATTR_LOCALITY_AWARE_TRANSACTION连接属性并已准备命令的情况下才能调用SQLGetSuitableGroupID

命令中有参数时SQLGetSuitableGroupID返回使用参数标记的当前值可在集群系统中最优化执行命令的集群组的ID无法决定可最优化执行的集群租时返回SQL_INVALID_GROUP_ID(-1)

SQLGetTypeInfo

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLGetTypeInfo返回数据源支持的数据类型相关信息驱动程序以SQL结果集合类型返回信息数据类型用于数据定义语言（DDL）命令

语句

```
SQLRETURN SQLGetTypeInfo(
    SQLHSTMT      StatementHandle,
    SQLSMALLINT   DataType);
```

参数

StatementHandle

【输入】结果集的命令语句柄

DataType

【输入】SQL 数据类型数据类型或指定的驱动程序SQL数据类型SQL_ALL_TYPES指定为返回所有数据类型的信息

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, 或 SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
01000	General warning	具体的驱动程序信息（函数返回 SQL_SUCCESS_WITH_INFO）
01S02	Option value changed	由于实际操作条件指定的命令属性无效因此临时替代为相近值替代值直到关闭游标为止对 StatementHandle有效 可变更的命令属性如下 SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR. (函数返回 SQL_SUCCESS_WITH_INFO)
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败

SQLSTATE	报错	说明
24000	Invalid cursor state	StatementHandle中已打开游标并调用了SQLFetch或SQLFetchScroll SQLFetch或SQLFetchScroll返回SQL_NO_DATA时驱动程序返回此报错不返回SQL_NO_DATA时驱动程序管理器返回此报错 StatementHandle中已打开结果集但未调用SQLFetch或SQLFetchScroll
40001	Serailization failure	由于其他事务的资源死锁事务被回滚
40003	Statement completion unknown	相关连接在该函数的执行过程中失败无法查看事务状态
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY004	Invalid SQL data type	DataType参数指定的值既不是驱动程序支持的ODBC SQL数据类型标识符也不是驱动程序数据类型标识符

SQLSTATE	报错	说明
HY008	Operation canceled	可异步处理StatementHandle调用并完成此函数之前StatementHandle调用了SQLCancel或SQLCancelHandle函数之后此函数再次被StatementHandle调用 调用并完成此函数之前多线程应用程序从其他线程在StatementHandle调用了SQLCancel或SQLCancelHandle
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLGetStmtAttr时此异步执行的函数仍然在执行 对StatementHandle调用了异步执行函数调用此函数时此异步执行函数仍在执行中 对StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation 或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution变量或column的苏剧之前调用了此函数
HY013	Memory management error	作为参数的缓冲区大小小于0或无法访问内存

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran** 函数
HYC00	Optional field not implemented	驱动程序或数据源不支持SQL_ATTR_CONCURRENCY和SQL_ATTR_CURSOR_TYPE命令属性的当前设置的组合 SQL_ATTR_USE_BOOKMARKS命令属性没有设置于SQL_UB_VARIABLESQL_ATTR_CURSOR_TYPE命令属性设置为驱动程序不支持书签游标类型
HYT00	Timeout expired	数据源返回结果集之前语句执行超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置
HYT01	Connection timeout expired	数据源响应请求之前超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	DescriptorHandle相关的驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时无法使用轮询（Polling）

SQLSTATE	报错	说明
Im018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	之前的函数调用对句柄返回SQL_STILL_EXECUTING并提醒模式生效时为了句柄的后处理与结束操作必需调用SQLCompleteAsync

说明

SQLGetTypeInfo返回标准结果集合等结果其排列为紧密的映射于对应DATA_TYPEODBCSQL数据类型的数据类型数据源定义的数据类型优先于用户自定义的数据类型因此排序顺序不一定一致但通常以DATA_TYPE为第一个之后为TYPE_NAME顺序的升序排序

例如数据源定义了INTEGER和COUNTER数据类型并自动增加COUNTER定义了用户自定义数据类型WHOLENUM时以INTEGERWHOLENUMCOUNTER顺序返回因为此结果是WHOLENUM与ODBC SQL数据类型SQL_INTEGER紧密映射相反即使数据源支持但自动增加的数据类型也不会与ODBC SQL 数据类型紧密的映射

即使DataType参数在驱动程序支持的ODBC版本中有效但驱动程序不支持时返回空结果集合

在ODBC 3.x中以下column名发生变更应用程序以column编号绑定因此column名的变更不影响与之前版本的兼容性

ODBC 2.0 column	ODBC 3.x column
PRECISION	COLUMN_SIZE
MONEY	FIXED_PREC_SCALE

AUTO_INCREMENT	AUTO_UNIQUE_VALUE

以下column在ODBC 3.x中添加至SQLGetTypeInfo返回的结果集

- SQL_DATA_TYPE
- INTERVAL_PRECISION
- SQL_DATETIME_SUB
- NUM_PREC_RADIX

以下表格列出了结果集的column驱动程序可以定义column 19 (INTERVAL_PRECISION) 之后添加的行应用程序应该从结果集的末尾开始计数并访问特定驱动程序column而不是指定一个明确的序数位置

> Note:
>
> SQLGetTypeInfo可能不会对所有数据类型进行返回例如驱动程序可能不返回用户自定义的数据类型不管SQLGetTypeInfo的返回与否应用程序均可使用有效的数据类型
>
> SQLGetTypeInfo返回的数据类型是数据源支持的数据类型其用于数据定义语言（DDL）命令语驱动程序可能返回结果集数据使用的数据类型而不是SQLGetTypeInfo返回的类型生成目录函数的结果集时驱动程序可能使用数据源不支持的数据类型

Column name	Column number	Data type	Comments
TYPE_NAME (ODBC 2.0)	1	Varchar not NULL	数据源从属的数据类型名称有CHAR()VARCHAR()MONEYLONG VARBINARY"或CHAR ( ) FOR BIT DATA等应用程序在CREATE TABLE与ALTER TABLE命令语中必须使用此名称
DATA_TYPE (ODBC 2.0)	2	Smallint not NULL	SQL 数据类型其可以为ODBC SQL 数据类型或特定驱动程序SQL数据类型此column对DATETIME或INTERVAL数据类型返回简洁的数据类型 (SQL_TYPE_TIME或SQL_INTERVAL_YEAR_TO_MOUNT)
COLUMN_SIZE (ODBC 2.0)	3	Integer	服务器支持的数据类型的最大column大小数字型数据类型为最大Precision字符串数据类型为字符的长度DATETIME 数据类型为表示的字符长度INTERVAL数据类型为其INTERVAL本身的字符表示的字符长度对无法适用column大小的数据类型返回空值
LITERAL_PREFIX (ODBC 2.0)	4	Varchar	用作字符或字符串的前缀例如单引号(')用于字符数据类型或0x或二进制数据类型对无法用作前缀的数据类型返回空值

Column name	Column number	Data type	Comments
LITERAL_SUFFIX (ODBC 2.0)	5	Varchar	用作字符或字符串的后缀例如单引号(')用于字符数据类型对无法用作后缀的数据类型返回空值
CREATE_PARAMS (ODBC 2.0)	6	Varchar	使用返回于TYPE_NAME的名称时对应于应用程序在括号内指定的各参数（以逗号区分）的关键字目录 目录的关键字是长度Precision或Scale其按照语法使用关键字的顺序显示例如DECIMAL的CREATE_PARAMS是"precisionscaleVARCHAR的CRATE_PARAMS是相同的"长度" 如果没有定义数据类型的参数时返回空值（例如INTERGER）驱动程序以使用的国家/地区语言提供CREATE_PARAMS文本
NULLABLE (ODBC 2.0)	7	Smallint not NULL	数据类型是否可以为空值 SQL_NO_NULLS时数据类型无法接收空值 SQL_NULLABLE时数据类型可以接收空值 SQL_NULLABLE_UNKNOWN时不确定column是否接收空值

Column name	Column number	Data type	Comments
CASE_SENSITIVE (ODBC 2.0)	8	Smallint not NULL	在排序和比较运算中字符数据类型是否区分大小写 数据类型为字符数据类型并区分大小写时是SQL_TRUE 不是字符数据类型或不区分大小写时是SQL_FALSE
SEARCHABLE (ODBC 2.0)	9	Smallint not NULL	WHERE条件中数据类型的使用方法 WHERE条件中无法使用column时为SQL_PRED_NONE (与ODBC 2.x中的SQL_UNSEARCHABLE值相同) WHERE条件中可以使用column但只与LIKE条件同时使用时为SQL_PRED_CHAR (与ODBC 2.x的SQL_LIKE_ONLY值相同) WHERE条件中column除LIKE条件外可与所有其他比较运算同时使用时为SQL_PRED_BASIC (与ODBC 2.x的 SQL_ALL_EXCEPT_LIKE值相同) 可与WHERE条件中所有运算同时使用column时为SQL_SEARCHABLE

Column name	Column number	Data type	Comments
UNSIGNED_ATTRIBUTE (ODBC 2.0)	10	Smallint	数据类型中是否有符号 数据类型中无符号时为SQL_TRUE 数据类型中有符号时为SQL_FALSE 数据类型无法使用属性或数据类型不是数字时返回空值
FIXED_PREC_SCALE (ODBC 2.0)	11	Smallint not NULL	数据类型（各数据源）是否拥有提前定义的固定Precision与Scale 拥有提前定义的固定Precision和Scale时为SQL_TRUE 没有提前定义的固定Precision和Scale时为SQL_FALSE
AUTO_UNIQUE_VALUE (ODBC 2.0)	12	Smallint	数据类型是否自动增加 数据类型自动增加时为SQL_TRUE 数据类型不自动增加时为SQL_FALSE 属性不适用于数据类型或数据类型不是数字时返回空值 应用程序可以在拥有此属性的column插入值但不能更新column的值在自动增加的column中插入时插入column的固有值增加根据数据源而不额外定义应用程序不可预测自动增加的column从某个指定点开始或由某个指定值开始增加

Column name	Column number	Data type	Comments
LOCAL_TYPE_NAME (ODBC 2.0)	13	Varchar	从属于数据源的数据类型的本地化版本 如果数据源不支持本地化名称则返回空值此名称类似于对话窗口用于显示
MINIMUM_SCALE (ODBC 2.0)	14	Smallint	数据源的数据类型的最小Scale数据类型为固定长度Scale时MINIMUM_SCALE和 MAXIMUM_SCALE column拥有此值例如 SQL_TYPE_TIMESTAMP column对小数秒有固定 Scale不能使用Scale时返回空值
MAXIMUM_SCALE (ODBC 2.0)	15	Smallint	数据源的数据类型的最大Scale无法使用Scale时返回空值数据源不单独定义最大Scale但定义与最大Precision相同时此column拥有与 COLUMN_SIZE column相同的值
SQL_DATA_TYPE (ODBC 3.0)	16	Smallint NOT NULL	描述符的SQL_DESC_TYPE字段中出现的SQL数据类型的值此column与除INTERVAL和DATETIME数据类型外的DATA_TYPE column相同对于 INTERVAL和DATETIME数据类型结果集的 SQL_DATA_TYPE字段返回SQL_INTERVAL或 SQL_DATETIMESQL_DATETIME_SUB字段返回 INTERVAL或DATETIME数据类型的子码

Column name	Column number	Data type	Comments
SQL_DATETIME_SUB (ODBC 3.0)	17	Smallint	一个SQL_DATE_TYPE值为SQL_DATETIME或SQL_INTERVAL时此column拥有DATETIME/INTERVAL的子码对于其他数据类型此字段值是空值 对于INTERVAL或DATETIME数据类型结果集的SQL_DATE_TYPE返回SQL_INTERVAL或SQL_DATETIMESQL_DATETIME_SUB字段返回INTERVAL或DATETIME数据类型子码
NUM_PREC_RADIX (ODBC 3.0)	18	Integer	如果数据类型是近似数字类型为了表示COLUMN_SIZE指定位数此字段值为2对于准确的数字类型为了表示COLUMN_SIZE指定十进制数此字段为10否则此column为空值
INTERVAL_PRECISION (ODBC 3.0)	19	Smallint	对于INTERVAL数据类型此column拥有INTERVAL leading precision的值否则为空

属性信息可以应用于数据类型或结果集中的特定columnSQLGetTypeInfo返回与数据类型相关的属性信息SQLColAttribute返回结果集中与列相关的属性信息

SQLMoreResults

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLMoreResults查看在SELECTUPDATEINSERT或DELETE等命令语中是否可使用更多的结果可以时初始化对其结果的处理

语句

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

参数

StatementHandle

【输入】命令语句柄

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE, 或 SQL_PARAM_DATA_AVAILABLE

检测

SQLSTATE	报错	说明
01000	General warning	具体的驱动程序信息（函数返回SQL_SUCCESS_WITH_INFO）
01S02	Option value changed	批处理过程中属性值被变更（函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
40001	Serailization failure	因其他事务的资源死锁事务被回滚
40003	Statement completion unknown	执行此函数的过程中相关连接失败无法查看事务的状态
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY008	Operation canceled	可异步处理StatementHandle调用此函数后完成执行之前StatementHandle调用了SQLCancel或SQLCancelHandle函数之后此函数再次被StatementHandle调用 调用此函数之前多线程应用程序从其他多线程在StatementHandle调用了SQLCancel或SQLCancelHandle

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLMoreResultsr时此异步执行的函数还在执行中 对StatementHandle调用了异步执行的函数调用此函数时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos后返回了SQL_NEED_DATA传送所有data-at-execution变量之前调用了此函数
HY013	Memory management error	作为参数的缓冲区的大小值小于0或无法访问内存
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran** 函数
HYT01	Connection timeout expired	数据源响应请求之前超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置

SQLSTATE	报错	说明
IM001	Driver does not support this function	与DescriptorHandle相关的驱动程序不支持函数
IM017	Polling is disabled in asynchronous notification mode	使用提醒模式时无法使用轮询
Im018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	句柄的之前的函数调用返回SQL_STILL_EXECUTING并激活提醒模式后为了句柄的后处理和完成操作必需调用SQLCompleteAsyc

说明

SELECT语句返回结果集UPDATEINSERTDELETE语句返回受影响的行数

如果这些命令中有一个被批处理或传输了参数的数组或处于正在处理的状态则可返回多个结果集或行数

在执行批处理后应用程序位于第一个结果集应用程序对第一个或后续的结果集可以像有单一结果集一样调用SQLBindColSQLBulkOperationsSQLFetchSQLGetDataSQLFetchScrollSQLSetPos和所有元数据函数在第一个结果集执行SQLMoreResults时应用程序为了移动至下一个结果集调用SQLMoreResults如果可以使用另一个结果集或数量SQLMoreResults返回SQL_SUCCESS并初始化结果集或统计额外的操作如果行数生成语句出现在结果集生成语句之间行数生成语句可以转到调用SQLMoreResults应用程序对UPDATEINSERT或DELETE 命令语调用SQLMoreResult后可以

调用SQLRowCount

如果有未浏览的行的当前结果集时SQLMoreResults将丢弃其结果集并商城或统计下一个结果集如果完成处理所有的结果集时SQLMoreResults返回SQL_NO_DAT对于某些驱动程序处理完所有的结果集和行计数之前无法使用输出变量与返回值这种驱动程序只有在SQLMoreResults返回SQL_NO_DATA后才能使用输出变量与返回值

为以前的结果集创建的所有绑定仍然保持有效状态column结构与此结果集不同时调用SQLFetch或SQLFetchScroll会报错或被截断为了避免这种情况应用程序应调用SQLBindCol明确恰当的重新进行绑定另外应用程序为了解除所有column缓冲区的绑定可与SQL_UNBIND选项同时调用SQLFreeStmt

应用程序调用SQLMoreResults进行批处理的过程中可变更游标类型游标并发性key集合大小最大长度等命令属性值此时SQLMoreResults将返回SQL_SUCCESS_WITH_INFO和SQLSTATE 01S02(Option value has changed)

使用SQL_CLOSE option调用SQLCloseCursor或SQLFreeStmt时丢弃所有结果集与批处理结果等可用的行数命令语句柄返回分配的状态或准备（prepare）状态中的一个执行批处理并执行命令语句柄时为了取消异步执行功能调用SQLCancel如果执行成功将丢弃批处理生成的所有结果集的异步状态结果与行数此命令语返回准备或分配状态

命令语的批处理或procedure与SELECTUPDATEINSERTDELETE等其他SQL命令语混合使用时这种其他命令语不影响SQLMoreReuslt

命令语的批处理中如果更新插入或删除命令语不影响数据源的任何行则SQLMoreResults返回SQL_SUCCESS这与SQLExecDirectSQLExecute或SQLParamData在这种情况下返回SQL_NO_DATA不同在SQLMoreResults不影响任何行的情况下应用程序为了检索行数调用SQLRowCount时SQLRowCount返回SQL_NO_DATA

行数的可用性

批处理包含多个连续的行计数生成语句时这些行计数作为一个行计数进行rollup处理例如批处理有五个插入语句时特定数据源可返回五个单独的行数部分其他数据源只返回表示五个独立行数的总和的一个行数

当一个批处理包含生成结果集并生成行计数的命令语的组合时可能无法使用row数

根据行数的可用性的驱动程序操作通过调用SQLGetInfo在要使用的SQL_BATCH_ROW_COUNT信息类型中列出例如批处理包含两个INSERT与其他SELECT语句后接SELECT语句时可能会出现以下情况

- 无法使用对应两个INSERT语句的行数第一次调用SQLMoreResults将位于第二个SELECT语句的结果集
- 对应两个INSERT语句的行数可以独立使用（(SQLGetInfo对SQL_BATCH_ROW_COUNT信息类型不返回SQL_BRC_ROLLED_UP bit）第一次调用SQLMoreResults位于第一个INSERT语句的行数第二次调用位于第二个INSERT语句的行数第三次的调用位于第二个SELECT语句的结果集
- 对应两个INSERT语句的行数合rollup到可使用的单个行数（调用SQLGetInfo对SQL_BATCH_ROW_COUNT信息类型不返回SQL_BRC_ROLLED_UP bit）第一次调用SQLMoreResults位于叠加后的行数第二次调用SQLMoreResults位于第二个SELECT语句的结果集

特定驱动程序只对明确的批处理生成可用的行数

SQLNativeSql

不支持

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLNativeSql返回由驱动程序修改的SQL字符串SQLNativeSql不执行SQL命令语

语句

```
SQLRETURN SQLNativeSql(
    SQLHDBC        ConnectionHandle,
    SQLCHAR *      InStatementText,
    SQLINTEGER     TextLength1,
    SQLCHAR *      OutStatementText,
    SQLINTEGER     BufferLength,
    SQLINTEGER *   TextLength2Ptr);
```

SQLNumParams

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLNumParams 返回SQL命令语的参数数量

语句

```
SQLRETURN SQLNumParams(
    SQLHSTMT       StatementHandle,
    SQLSMALLINT *   ParameterCountPtr);
```

参数

StatementHandle

【输入】命令语句柄

ParameterCountPtr

【输出】返回命令语的参数数量的缓冲区指针

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR,

SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	驱动程序提供的信息（函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY008	Operation canceled	可异步处理StatementHandle调用并完成执行此函数之前StatementHandle调用了SQLCancel或SQLCancelHandle函数此函数再次被StatementHandle调用 多线程应用程序中调用并完成执行此函数之前从其他线程在StatementHandle调用了SQLCancel或SQLCancelHandle

SQLSTATE	报错	说明
HY010	Function sequence error	StatementHandle调用SQLPrepare或SQLExecDirect之前调用了此函数 StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLNumParams时异步执行的函数仍然在执行中 StatementHandle调用了异步执行的函数调用此函数时异步执行的函数仍然在执行中 StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution参数或column的数据之前调用了此函数
HY013	Memory management error	无法访问内部内存或可使用的内存不足
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran** 函数

SQLSTATE	报错	说明
HYT01	Connection timeout expired	数据源响应请求之前超时其限制时间通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	StatementHandle相关的驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄之前的函数调用返回SQL_STILL_EXECUTING并提醒模式生效时对句柄的后处理和结束操作必须调用SQLCompleteAsync

说明

只能在调用SQLPrepare后调用SQLNumParams

与StatementHandle相关的命令语没有参数时SQLNumParams将*ParameterCountPtr设置为0

SQLNumParams返回的参数数量与IPD的SQL_DESC_COUNT字段值相同

SQLNumResultCols

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLNumResultCols返回结果集的列的数量

语句

```
SQLRETURN SQLNumResultCols(
    SQLHSTMT       StatementHandle,
    SQLSMALLINT *   ColumnCountPtr);
```

参数

StatementHandle

【输入】命令语句柄

ColumnCountPtr

【输出】结果集中返回列的数量的缓冲区指针此数量不包含书签列

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	驱动程序的信息（函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY008	Operation canceled	可异步处理StatementHandle调用并完成执行此函数之前StatementHandle调用了SQLCancel或SQLCancelHandle函数此函数再次被StatementHandle调用 多线程应用程序中调用此函数并完成执行之前从其他线程在StatementHandle调用了SQLCancel或SQLCancelHandle

SQLSTATE	报错	说明
HY010	Function sequence error	StatementHandle调用SQLPrepare或SQLExecDirect之前调用了此函数 对与StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLNumResultsCols时异步执行的函数还在执行中 StatementHandle调用了异步执行函数调用此函数时异步执行的函数还在执行中 StatementHandle调用SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送data-at-execution参数或column的数据之前调用了此函数
HY013	Memory management error	无法访问内部内存或可使用的内存不足
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran**函数

SQLSTATE	报错	说明
HYT01	Connection timeout expired	数据源响应请求之前连接超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	与StatementHandle相关的驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄之前的函数调用返回SQL_STILL_EXECUTING提醒模式生效时为了句柄的后处理和结束操作必须调用SQLCompleteAsync

说明

只有在命令语处于准备执行定位状态时才能成功调用SQLNumResultCols

StatementHandle相关的命令语不返回列时SQLNumResultCols将*ColumnCountPtr设置为0

SQLNumResultCols返回的列的数量与IRD的SQL_DESC_COUNT字段值相同

SQLParamData

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLParamData为了执行命令时提供参数与SQLPutData一起使用

语句

```
SQLRETURN SQLParamData(
    SQLHSTMT       StatementHandle,
    SQLPOINTER *   ValuePtrPtr);
```

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_NO_DATA,

SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

检测

SQLSTATE	报错	说明
01000	General warning	驱动程序提供的信息（函数返回SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY010	Function sequence error	之前调用的函数不是返回SQL_NEED_DATA的SQLExecDirectSQLExecute 之前的函数为SQLParamData
HY013	Memory management error	无法访问内部内存或可使用的内存空间不足
HYT01	Connection timeout expired	数据源响应请求之前连接超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置

为了传送参数数据调用SQLParamData时可能返回SQLExecuteSQLExecDirect的SQLSTATE

说明

应用程序调用需要data-at-execution的SQLExecute或SQLExecDirect时驱动程序返回SQL_NEED_DATA应用程序为了决定要传送的数据调用SQLParamData驱动程序需要参数数据时返回应用程序输入的*ValuePtr值应用程序为了查看驱动程序请求的参数数据可以使用此值

应用程序为了传送参数数据调用SQLPutData传送参数的所有数据后应用程序再次调用SQLParamDataSQLParamData再次返回SQL_NEED_DATA时应用程序应重新调用SQLPutData传送其他参数数据如果已传送所有的参数数据时SQLParamData返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO不定义*ValuePtr的值并可执行SQL语句

SQLParamOptions

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 2.0 函数SQLParamOptions在ODBC 3.x中被替代为**SQLSetStmtAttr**

SQLPrepare

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLPrepare准备要执行的SQL字符串

语句

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

参数

StatementHandle

【输入】命令语句柄

StatementText

【输入】SQL文本字符串

TextLength

【输入】字符 *StatementText的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	驱动程序的信息（函数返回SQL_SUCCESS_WITH_INFO）
01S02	Option value changed	指定的命令属性值不适合执行操作因此临时被替代为相近值（通过调用SQLGetStmtAttr可查看临时变更的值）此替代值有效至关闭游标关闭游标时变更为原来的值 可变更的属性为如下 SQL_ ATTR_CONCURRENCY, SQL_ ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ ATTR_QUERY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR （函数返回 SQL_SUCCESS_WITH_INFO）
21S01	Insert value list does not match column list	*StatementText中有INSERT语句插入值的数量和衍生的表不一致
21S02	Degree of derived table does not match column list	*StatementText中有CREATE VIEW有语句指定的名称数量与查询明细定义的衍生的表不一致

SQLSTATE	报错	说明
22018	Invalid charcter value for cast specification	*StatementText中有包含字符串或参数的SQL命令值相关表的表列的数据类型不兼容
22019	Invalid escape character	StatementText参数包含WHERE子句的ESCAPE等LIKE谓词ESCAPE的控制字符长度不是1
22025	Invalid escape sequence	StatementText参数在WHERE自居包含LIKEpattern ESCAPEescape character模式值控制字符不是 % 或 -
24000	Invalid cursor state	StatementHandle中已打开游标并调用了SQLFetch或SQLFetchScroll函数
34000	Invalid cursor name	*StatementText包含指定位置DELETE或指定位置UPDATE 未打开已准备的命令语参考的游标
3D000	Invalid catalog name	StatementText中指定的目录名称无效
3F000	Invalid schema name	StatementText中指定的Schema名称无效
42000	Syntax error or access violation	*StatementText包含语句报错或无法准备的SQL命令语 *StatementText在命令语包含没有对应权限的用户
42S01	Base table or view already exists	*StatementText包含CREATE TABLE或CREATE VIEW语句已存在指定的表或视图的名

SQLSTATE	报错	说明
42S02	Base table or view not found	*StatementText包含DROP TABLE或DROP VIEW语句没有指定的表或视图名 *StatementText包含ALTER TABLE语句没有指定的表名 *StatementText包含CREATE VIEW语句没有查询明细中定义的表或视图的名 *StatementText包含CREATE INDEX语句没有指定的表名 *StatementText包含GRANT或REVOKE 语句没有指定的表或视图的名 *StatementText包含SELECT语句没有指定的表或视图的名 *StatementText包含DELETEINSERT或UPATE语句没有指定的表名 *StatementText包含CREATE TABLE语句没有约束条件指定的表名（参考其他表）
42S11	Index already exists	*StatementText中含有CREATE INDEX语句已存在指定的索引名

SQLSTATE	报错	说明
42S12	Index not found	*StatementText中有DROP INDEX语句不存在指定的索引名
42S21	Column already exist	*StatementText中有ALTER TABLE语句ADD语句中指定的列不是唯一的或识别基本表中的现有列
42S22	Column not foun	*StatementText中有CREATE INDEX语句一个以上的列名称不存在于指定的列目录中 *StatementText中有GRANT或REVOKE语句指定的列名称不存在 *StatementText中有SELECTDELETE或UPDATE语句指定的列名称不存在 *StatementText中有CREATE TABLE语句不存在约束条件中指定的列
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY008	Operation canceled	可异步处理StatementHandle调用并执行完此函数之前StatementHandle调用了SQLCancel或SQLCancelHandle函数（此函数再次被StatementHandle调用） 多线程应用程序中调用并结束此函数之前其他线程在StatementHandle调用了SQLCancel或SQLCancelHandle
HY009	Invalid use of null pointer	StatementText为空指针
HY010	Function sequence error	在StatementHandle调用SQLPrepare或SQLExecDirect之前调用了此函数 为了StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLPrepare时异步执行的函数仍在执行中 StatementHandle调用了异步执行的函数调用此函数时异步执行的函数仍在执行中 StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送data-at-execution或column的数据之前调用了此函数
HY013	Memory management error	无法访问内部内存或可使用的内存不足

SQLSTATE	报错	说明
HY090	Invalid string or buffer length	TextLength小于或等于0 或不等于SQL_NTS
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran** 函数
HYC00	Optional feature not implemented	对定义的游标类型并发性设置无效 SQL_ATTR_USE_BOOKMARKS命令属性设置为 SQL_UB_VARIABLESQL_ATTR_CURSOR_TYPE命令属性设置为驱动程序不支持书签游标类型
HYT00	Timeout expired	从数据源接收结果集之前语句执行超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置
HYT01	Connection timeout expired	数据源响应请求之前超时其限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	StatementHandle相关的驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的

SQLSTATE	报错	说明
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄之前的函数调用返回SQL_STILL_EXECUTING提醒模式生效时对句柄的后处理和结束操作必须调用SQLCompleteAsync

说明

应用程序为了准备工作通过调用SQLPrepare将SQL命令语传送给数据源应用程序在SQL命令语中可以包含一个以上的参数标记字符为了包含参数标记应用程序在SQL语句的适当位置中使用问号(?)

> Note:
> 如果应用程序为了准备操作使用SQLExecute传输SQLPrepare和COMMIT或ROLLBACK语句时数据库产品之间无法进行互操作

驱动程序为了使用数据源使用的SQL类型可以修改命令语也可以为了准备工作传送给数据源尤其是驱动程序可以修改为了定义特定函数的SQL语句的扩展bit列驱动程序中的命令语句柄类似于嵌入式SQL代码的命令语标识符如果数据源支持命令语标识符则驱动程序可以把命令语标识符和参数值传送给数据源

准备命令语后应用程序为了在后续调用的函数中参考命令语而使用命令语句柄应用程序通过SQL_DROP选项调用SQLFreeStmt解除命令语或命令语句柄使用SQLPrepareSQLExecDirect或目录函数（SQLColumns, SQLTables等）之前可调用SQLExecute再次执行与命令语句柄相关的已准

备的命令语一旦应用程序准备命令语则可以请求结果集合类型相关信息部分操作的情况调用SQLPrepare后调用SQLDescribeCol或SQLDescribeParam效率可能低于SQLExecute或SQLExecDirec之后调用

驱动程序在应用程序调用SQLPrepare时不能返回语法错误或违规访问驱动程序均可处理语法错误和违规访问或只能处理语法错误或均无法处理语法错误或违规访问因此应用程序调用后续相关函数（SQLNumResultColsSQLDescribeColSQLColAttribute与SQLExecute等后续函数）时需可处理这些条件

根据驱动程序与数据源的功能已准备命令语（绑定了所有参数时）或（未绑定所有参数时）执行时可以检查（数据类型等）参数信息应用程序在同一个命令语中准备新的SQL之前需要解除之前SQL语句绑定的所有参数这是为了防止之前的参数信息应用于新的命令语的错误

> Caution:
>
> 明示调用SQLEndTran或在自动提交模式下提交事务时数据源可能删除与连接相关的所有命令的访问计划更详细的内容参考**SQLGetInfo** 的信息类型SQL_CURSOR_COMMIT_BEHAVIOR和SQL_CURSOR_ROLLBACK_BEHAVIOR

SQLPrimaryKeys

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLPrimaryKeys返回由主键构成的column名称驱动程序以结果集返回信息此函数不支持通过单次调用从多个表返回主键

语句

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

参数

StatementHandle

【输入】命令语句柄

CatalogName

【输入】目录名称如果驱动程序仅支持部分表的目录则空字符("")表示没有目录的表

CatalogName不能包含字符串检索模式

SQL_ATTR_METADATA_ID属性为SQL_TRUE时CatalogName将作为不区分大小写的标识符

设置为SQL_FALSE时CatalogName区分大小写并作为普通的因子字符串

NameLength1

【输入】*CatalogName的字符长度

SchemaName

【输入】SCHEMA名称驱动程序仅支持部分表的Schema时空字符("")表示没有Schema的

表SchemaName不能包含字符串的检索模式

SQL_ATTR_METADATA_ID属性为SQL_TRUE时SchemaName将作为不区分大小写的标识

符设置为SQL_FALSE时SchemaName不区分大小写并作为模式值字符串

NameLength2

【输入】*SchemaName字符长度

TableName

【输入】表名此参数不能为空指针TableName不能包含字符串检索模式

SQL_ATTR_METADATA_ID命令属性为SQL_TRUE时TableName将作为不区分大小写的标

识符设置为SQL_FALSE时TableName不区分大小写并作为普通字符串

NameLength3

【输入】*TableName字符长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR,

SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	各驱动程序的信息（函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
24000	Invalid cursor state	调用了SQLFetchSQLFetchScroll已打开游标 未调用SQLFetchSQLFetchScroll而存在打开的游标
40001	Serialization failure	因其他事务的资源死锁此事务被回滚
40003	Statement completion unknown	相关连接在函数执行过程中失败无法查看事务状态
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY008	Operation canceled	可异步处理StatementHandle调用并完成执行完函数之前StatementHandle调用了或SQLCancelHandle函数而且StatementHandle再次调用了此函数 多线程应用程序中调用并结束此函数之前从其他线程在StatementHandle调用了SQLCancel或SQLCancelHandle
HY009	Invalid use of null pointer	TableName参数为空指针 SQL_ATTR_METADATA_ID命令属性设置为为SQL_TRUECatalogName未空指针返回包含SQL_CATALOG_NAME信息类型的SQLGetInfo支持目录名称的事实 SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUESchemaName参数为空指针

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLPrimaryKeys时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults 并返回了SQL_PARAM_DATA_AVAILABLE检查已连接的所有参数数据之前调用了此函数 对StatementHandle调用了异步执行函数调用此函数时异步执行的函数还在执行中 为了StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperator或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution或column数据之前调用了此函数
HY013	Memory management error	无法访问内部内存或可使用的内存不足
HY090	Invalid string or buffer length	名称长度参数中有一个值小于0但不是SQL_NTS名称参数不为空指针 名称长度参数中有一个值超过了与名称对应的最大长度值

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran** 函数
HYC00	Optional feature not implemented	指定了目录但驱动程序或数据源不支持目录 指定了Schema但驱动程序或数据源不支持Schema 驱动程序或数据源不支持SQL_ATTR_CONCURRENCY和SQL_ATTR_CURSOR_TYPE命令属性的当前设置组合 SQL_ATTR_USE_BOOKMARKS命令属性设置为SQL_UB_VARIABLESQL_ATTR_CURSOR_TYPE命令属性被设置为驱动程序不支持的书签的游标类型
HYT00	Timeout expired	从数据源接收结果集之前语句执行超时其限制时间可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置
HYT01	Connection timeout expired	数据源响应请求之前超时其限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	StatementHandle相关的驱动程序不支持此函数

SQLSTATE	报错	说明
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的 .
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄的之前的函数调用返回SQL_STILL_EXECUTING提醒模式生效时为了后处理和结束操作句柄必须调用QLCompleteAsync

说明

SQLPrimaryKeys以TABLE_CATTABLE_SCHEMATABLE_NAMEKEY_SEQ排序的标准结果集返回结果使用此信息的详细内容参考目录数据的使用

在ODBC 3.x中更改了以下column的名称应用程序以column编码绑定因此column名的变更不影响与之前版本的兼容性

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

为了查看TABLE_CATTABLE_SCHEMTABLE_NAME和OLUMN_NAMEclumn的实际长度与SQL_MAX_CATALOG_NAME_LENSQL_MAX_SCHEMA_NAME_LENSQL_MAX_TABLE_NAME_LENSQL_MAX_COLUMN_NAME_LEN选项同时调用SQLGetInfo

Note:

相关ODBC目录函数的一般用法参数以及返回的数据的详细内容参考 **目录函数**

以下表为结果集的column列表驱动程序可以定义column6以后增加的字段应用程序应该从结果集的尾开始计数访问相关column而不是指定位置更详细的内容参考**目录函数的数据返回**

Column name	Column number	数据类型	说明
TABLE_CAT (ODBC 1.0)	1	Varchar	主键表目录名如果不适用于数据源则为空(如驱动程序在其他数据库中检索数据)驱动程序仅支持部分表的目录时对不包含目录的表返回空字符("")
TABLE_SCHEM (ODBC 1.0)	2	Varchar	主键表Schema名如果不适用于数据源则为空(如驱动程序在其他数据库中检索数据)驱动程序仅支持部分表的Schema时对不包含Schema的表返回空字符("")
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	主键表名
COLUMN_NAME (ODBC 1.0)	4	Varchar not NULL	主键列名称对没有名称的列驱动程序返回空字符
KEY_SEQ (ODBC 1.0)	5	Smallint not NULL	Key的column序号（从1开始）
PK_NAME (ODBC 2.0)	6	Varchar	主键名称如果不适用于数据源则为空

SQLProcedureColumns

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLProcedureColumns不仅返回输入输出参数的目录也返回构成指定过程的结果集的列驱动程序作为指定命令的结果集返回信息

语句

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

参数

StatementHandle

【输入】命令语句柄

CatalogName

【输入】procedure目录的名称驱动程序不支持目录时返回空字符串("")其procedure不包含目录CatalogName不可包含检索模式将SQL_ATTR_METADATA_ID设置为SQL_TRUE时CatalogName作为标识符并不区分大小写如果设置为SQL_FALSE则CatalogName作为普通的字符串参数并区分大小写详细信息参考目录函数的参数

NameLength1

【输入】*CatalogName字符串的长度

SchemaName

【输入】procedure SCHEMA名称schema名称的字符串检索模式驱动程序不支持schema时返回空字符串("")其procedure不包含schemaSchemaName没有字符串检索模式SQL_ATTR_METADATA_ID属性设置为SQL_TRUE时SchemaName将作为不区分大小写的标识符 SQL_FALSE时SchemaName区分大小写并作为模式值字符串参数

NameLength2

【输入】*SchemaName字符串的长度

ProcName

【输入】procedure名称该参数不能使用NULL指针ProcName不能包含字符串检索模式将SQL_ATTR_METADATA_ID设置为SQL_TRUE时ProcName作为标识符并不区分大小写如果

设置为SQL_FALSE则作为模式值字符串参数并区分大小写

NameLength3

【输入】*ProcName字符串的长度

ColumnName

【输入】column名此参数不能为空指针ColumnName不能包含字符串检索模式SQL_ATTR_METADATA_ID命令属性为SQL_TRUE时ColumnName将作为不区分大小写的标识符 SQL_FALSE时ColumnName区分大小写并作为模式值字符串

NameLength4

【输入】*ColumnName字符串的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	各驱动程序的信息（函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败

SQLSTATE	报错	说明
24000	Invalid cursor state	StatementHandle中游标已打开并调用了SQLFetch或SQLFetchScroll 驱动程序中SQLFetch或SQLFetchScroll返回SQL_NO_DATA时驱动程序返回此报错；不返回SQL_NO_DATA时驱动程序管理器返回此报错 StatementHandle结果集已打开但未调用SQLFetch或SQLFetchScroll
40001	Serialization failure	由于其他事务的资源死锁此事务被回滚
40003	Statement completion unknown	相关连接在函数执行过程中失败无法查看事务状态
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY009	Invalid use of null pointer	TableName参数为空指针 SQL_ATTR_METADATA_ID命令属性值为SQL_TRUECatalogName参数为空指针包含SQL_CATALOG_NAME信息类型的SQLGetInfo返回支持目录名称的事实 The SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUESchemaNameProcName或ColumnName参数为空指针

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了异步执行的函数调用SQLProcedureColumns时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults 并返回了SQL_PARAM_DATA_AVAILABLE检查所有已连接的参数的数据之前调用了此函数 对StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution或column的数据之前调用了此函数 对StatementHandle调用了异步执行的函数调用此函数时异步执行的函数还在执行中
HY090	Invalid string or buffer length	名称长度参数小于0但不同于SQL_NTS 名称长度参数中有一个值超过了与名称对应的最大长度值

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran** 函数
HYC00	Optional feature not implemented	指定了目录但驱动程序或数据源不支持目录 指定了Schema但驱动程序或数据源不支持Schema 驱动程序或数据源不支持SQL_ATTR_CONCURRENCY和SQL_ATTR_CURSOR_TYPE命令属性的当前设置组合 SQL_ATTR_USE_BOOKMARKS命令属性设置为SQL_UB_VARIABLESQL_ATTR_CURSOR_TYPE命令属性设置为对驱动程序不支持的书签的游标类型.
HYT00	Timeout expired	从数据源接收结果集之前语句执行超时其限制时间通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置
HYT01	Connection timeout expired	数据源响应请求之前超时其限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	StatementHandle相关的驱动程序不支持函数

SQLSTATE	报错	说明
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄之前的函数调用返回SQL_STILL_EXECUTING提醒模式生效时对句柄的后处理和结束操作必须调用SQLCompleteAsync

说明

SQLProcedureColumns不仅返回输出入参数的目录也返回构成指定程序的结果集的column驱动程序作为指定命令的结果集返回信息

SQLProcedureColumns返回按照PROCEDURE_CATPROCEDURE_SCHEMPROCEDURE_NAMECOLUMN TYPE的顺序排列的标准结果集Column名称按照各个参数的名称（调用顺序）和procedure返回的结果集的各个column名的顺序进行返回

为了查看PROCEDURE_CATPROCEDURE_SCHEMPROCEDURE_NAMECOLUMN_NAME column的实际长度应用程序可以与SQLGetInfo函数同时调用SQL_MAX_CATALOG_NAME_LENSQL_MAX_SCHEMA_NAMESQL_MAX_PROCEDURE_NAME_LEN选项

ODBC 3.x中变更了以下column的名称应用程序以column编号绑定因此变更column名不影响与之

前版本的兼容性

ODBC 2.0 column	ODBC 3.x column
PROCEDURE_QUALIFIER	PROCEDURE_CAT
PROCDEURE_OWNER	PROCEDURE_SCHEM
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SACLE	DECIMAL_DIGITS
RADIX	NUM_PREC_RADIX

以下column添加至ODBC 3.x中SQLProcedureColumns返回的结果集

- COLUMN_DEF
- DATETIME_CODE
- CHAR_OCTET_LENGTH
- ORDINAL_POSITION
- IS_NULLABLE

以下表列出了结果集的column驱动程序可以定义column19（IS_NULLABLE）以后增加的行应用程序应该从结果集的尾开始倒计时访问特定驱动程序column而不是指定位置

Column name	Column number	数据类型	说明
PROCEDURE_CAT (ODBC 2.0)	1	Varchar	procedure目录名如果不适用于数据源则为空(如驱动程序在其他数据库中检索数据)驱动程序仅支持部分表的目录时对不包含目录的表返回空字符("")
PROCEDURE_SCHEM (ODBC 2.0)	2	Varchar	procedure Schema名如果不适用于数据源则为空(如驱动程序在其他数据库中检索数据)驱动程序仅支持部分表的Schema时对不包含Schema的表返回空字符("")
PROCEDURE_NAME (ODBC 2.0)	3	Varchar not NULL	procedure名没有procedure名时返回空字符串
COLUMN_NAME (ODBC 2.0)	4	Varchar not NULL	procedure column名对没有名的procedure驱动程序返回空字符串

Column name	Column number	数据类型	说明
COLUMN_TYPE(ODEBC 2.0）	5	Smallint not NULL	定义procedure类型 • SQL_PARAM_TYPE_UNKNOWN:无法知道procedure column的类型（ODBC 1.0) • SQL_PARAM_INPUT：procedure column是input参数（ODBC 1.0） • SQL_PARAM_INPUT_OUTPUT: procedure column是input/output参数（ODBC 1.0） • SQL_PARAM_OUTPUT: procedure column是ouput参数（ODBC 2.0) • SQL_RETURN_VALUES: procedure column是procedure的返回值（ODBC 2.0) • SQL_RESULT_COL: procedure column是结果集column（ODCB 1.0）
DATA_TYPE (ODBC 1.0)	6	Samllint not NULL	SQL数据类型Datatime及Interval数据类型时该column返回SQL_TYPE_DATESQL_INTERVAL_YEAR_TO_MONTH等concise数据类型

Column name	Column number	数据类型	说明
TYPE_NAME(ODBC 2.0)	7	Varchar not NULL	数据源从属数据类型名有CHAR()VARCHAR()MONEYLONG VARBINARY或CHAR ( ) FOR BIT DATA等应用程序必须在CREATE TABLE与ALTER TABLE命令中使用该名称
COLUMN_SIZE(ODBC 2.0)	8	Integer	服务器对数据类型支持的最大column大小数字型数据类型为最大Preciseion字符串数据类型为字符长度DATATIME数据类型为表示的字符长度INTERVAl数据类型为INTERVAL字面上的文字表示的字符长度对无法适用column大小的数据类型返回NULL
BUFFER_LENGTH(ODBC 1.0)	9	Integer	指定SQL_C_DEFAULT时SQLGetData或SQLFetch运算传送的字节长度Numeric数据的大小可能不同于数据源中存储的数字的大小字符串或二进制数据时此值与COLUMN_SIZE column相同
DECIMAL_DIGITS(ODBC 1.0)	10	Smallint	数据源中列的小数点位数无法适用数据类型的小数点位数时返回NULL

Column name	Column number	数据类型	说明
NUM_PREC_RADIX(ODBC 2.0)	11	Smallint	数字型数据类型时2或10 2时COLUMN_SIZE与DECIMAL_DIGITS为column中允许的bit数 10时COLUMN_SIZE与DECIMAL_DIGITS为column中允许的bit数 无法适用NUM_PREC_RADIX的数据类型返回NULL
NULLABLE (ODBC 2.0)	12	Smallint not NULL	数据类型是否允许NULL值 SQL_NULLABLE时数据类型允许NULL值 SQL_NO_NULLS时数据类型不允许NULL值 SQL_NULLABLE_UNKNOWN时无法知道column是否允许NULL值
REMAKRS (ODBC 2.0)	13	Varchar	Procedure column相关说明
COLUMN_DEF (ODBC 3.0)	14	Varchar	Column的默认值该值用引号引住时该column应解释为字符串

Column name	Column number	数据类型	说明
SQL_DATA_TYPE	15	Smallint not NULL	描述符的SQL_DESC_TYPE字段中显示的SQL数据类型的值该column与除INTERVAL和DATETIME数据类型外的DATA_TYPE column相同结果集的SQL_DATE_TYPE字段对INTERVAL与DATETIME数据类型返回SQL_INTERVAL或SQL_DATETIMESQL_DATETIME_SUB字段返回INTERVAL或DATETIME数据类型的子代码
SQL_DATETIME_SUB (ODBC 3.0)	16	Smallint	Datetime与interval数据类型的子类型代码其他类型时返回NULL
CHAR_OCTET_LENGTH (ODBC 3.0)	17	Integer	字符或二进制数据类型column的字节单位最大长度其他类型时返回NULL
ORDINAL_POSITION (ODBC 3.0)	18	Integer not NULL	表中column的位置
IS_NULLABLE (ODBC 3.0)	19	Varchar	• “YES”：column可包含NULL • “NO”：column不可包含NULL • 无法确定是否允许NULL时返回长度为0的字符串

SQLProcedures

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLProcedures返回存储在指定数据源的procedure名的目录procedure是用于说明可使用可执行的对象或输入/输出调用的对象的一般用语

语句

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

参数

StatementHandle

【输入】命令语句柄

CatalogName

【输入】procedure目录名驱动程序不支持目录时返回空字符串("")其procedure不具备目录目录名称不可包含字符串检索模式将SQL_ATTR_METADATA_ID设置为SQL_TRUE时CatalogName作为标识符并不区分大小写如果设置为SQL_FALSE则CatalogName作为普通的字符串参数并区分大小写详细信息参考**目录函数的参数**

NameLength1

【输入】*CatalogName字符串的长度

SchemaName

【输入】procedure SCHEMA名称schema名称的字符串检索模式驱动程序不支持schema时返回空字符串("")其procedure不具备schemaSchemaName不能包含字符串检索模式SQL_ATTR_METADATA_ID设置为SQL_TRUE时SchemaName将作为不区分大小写的标识符设置为SQL_FALSE时SchemaName区分大小写并作为模式值字符串参数

NameLength2

【输入】*SchemaName字符串的长度

ProcName

【输入】procedure名称该参数不能作为NULL指针procedure名称不能包含字符串检索模式将SQL_ATTR_METADATA_ID设置为SQL_TRUE时ProcName作为标识符并不区分大小写如果设置为SQL_FALSE则作为模式值字符串参数并区分大小写

NameLength3

【输入】*ProcName 字符串的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	各驱动程序的信息（函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
24000	Invalid cursor state	StatementHandle中游标已打开并调用了SQLFetch或SQLFetchScroll SQLFetch或SQLFetchScroll返回SQL_NO_DATA时驱动程序返回此报错未返回SQL_NO_DATA时驱动程序管理器返回此报错 StatementHandle中结果集已打开但未调用SQLFetch或SQLFetchScroll
40001	Serialization failure	由于其他事务的资源死锁此事务被回滚
40003	Statement completion unknown	相关连接在执行此函数的过程中失败并无法查看事务的状态

SQLSTATE	报错	说明
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY009	Invalid use of null pointer	TableName参数为空指针 SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUECatalogName参数为空指针包含SQL_CATALOG_NAME信息类型的SQLGetInfo返回支持目录名这一事实 The SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUESchemaNameProcName或ColumnName参数为空指针

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了异步执行的函数调用SQLProcedureColumns时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults并返回了SQL_PARAM_DATA_AVAILABLE检查所有已连接参数的数据之前调用了此函数 对StatementHandle调用了异步执行的函数调用此函数时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA在传送所有data-at-exection参数或column的数据之前调用了此函数
HY013	Memory management error	无法访问内部内存或可使用的内存容量不足
HY090	Invalid string or buffer length	名称长度参数小于0但不同于SQL_NTS 名称长度参数中有一个值超过了名称对应的最大长度值

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran**函数
HYC00	Optional feature not implemented	指定了目录但驱动程序或数据源不支持目录 指定了Schema但驱动程序或数据源不支持Schema 驱动程序或数据源不支持SQL_ATTR_CONCURRENCY和SQL_ATTR_CURSOR_TYPE命令属性的当前设置组合 SQL_ATTR_USE_BOOKMARKS命令属性设置为SQL_UB_VARIABLESQL_ATTR_CURSOR_TYPE命令属性被设置为驱动程序不支持的书签的游标类型.
HYT00	Timeout expired	从数据源接收结果集之前语句执行超时其限制时间通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置
HYT01	Connection timeout expired	数据源响应请求之前超时其限制时间通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置

SQLSTATE	报错	说明
IM001	Driver does not support this function	StatementHandle相关的驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的 .
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄之前的函数调用返回SQL_STILL_EXECUTING提醒模式生效时对句柄的后处理和结束操作必须调用SQLCompleteAsync

说明

SQLProcedure列出请求范围的所有procedure用户可能拥有或未拥有procedure执行权限查看访问权限可参考SQLGetInfo的SQL_ACCESSIBLE_PROCEDURES用户选择无法执行的procedure时应用程序应可以控制SQLProcedures返回标准结果集以PROCEDURE_CATPROCEDURE_SCHEMAPROCEDURE_NAME的顺序排列

以下column在ODBC 3.x中已被重命名应用程序通过column编号绑定因此column重命名不影响与之前版本的兼容性

ODBC 2.0 column	ODBC 3.x column
PROCEDURE_QUALIFIER	PROCEDURE_CAT
PROCDEURE_OWNER	PROCEDURE_SCHEM

为了查看PROCEDURE_CATPROCEDURE_SCHEMPROCEDURE_NAME column的实际长度应用程序可以与SQLGetInfo同时调用SQL_MAX_CATALOG_NAME_LENSQL_MAX_SCHEMA_NAMESQL_MAX_PROCEDURE_NAME_LEN选项

以下表列出了结果集的column驱动程序可以定义Column 8 (PROCEDURE_TYPE) 以后增加的行应用程序应该从结果集的尾开始倒计时访问特定驱动程序column而不是指定位置

Column name	Column number	数据类型	说明
PROCEDURE_CAT (ODBC 2.0)	1	Varchar	Procedure目录名如果不适用于数据源则为空(如驱动程序在其他数据库中检索数据)驱动程序仅支持部分表的目录时未包含目录的表返回空字符("")
PROCEDURE_SCHEM (ODBC 2.0)	2	Varchar	Procedure schema标识符如果不适用于数据源则为空(如驱动程序在其他数据库中检索数据)驱动程序仅支持部分表的Schema时未包含Schema的表返回空字符("")
PROCEDURE_NAME (ODBC 2.0)	3	Varchar not NULL	Procedure标识符
REMARK(ODBC 2.0)	4	Varchar	Procedure的说明

Column name	Column number	数据类型	说明
PROCEDURE_TYPE(ODBC 2.0)	5	Smallint	定义procedure类型 • SQL_PT_UNKNOWN：无法查看procedure是否返回值 • SQL_PT_PROCEDURE：返回的对象为procedure即没有返回值 • SQL_PT_FUNCTION：返回的对象为FUNCTION即有返回值

SQLPutData

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLPutData使应用程序在执行命令语的过程中向驱动程序传送参数或向column传送数据此函数为拥有字符二进制或各数据源数据类型（例如SQL_LONGVARBINARY或SQL_LONGVARCHAR类型的参数）的column可用于传送部分字符二进制数据即使默认驱动程序不支持UnicodeSQLPutData支持Unicode C数据类型的绑定

语句

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

参数

StatementHandle

【输入】命令语句柄

DataPtr

【输入】存储参数或column的实际数据的缓冲区指针数据应为SQLBindParameter的ValueType参数（参数数据）或SQLBindCol的TargetType参数中指定的C数据类型（column数据）

StrLen_or_Ind

【输入】*DataPtr的长度指定通过调用SQLPutData传送的数据量根据指定的参数或column的调用数据量发生变化如StrLen_or_Ind不满足以下条件中的一个则被忽略

• StrLen_or_Ind为SQL_NTSSQL_NULL_DATA或SQL_DEFAULT_PARAM

• SQLBindParameter或SQLBindCol中指定的C 数据类型为SQL_C_CHAR或SQL_C_BINARY

• C 数据类型为SQL_C_DEFAULT指定的SQL数据类型的C数据类型为SQL_C_CHAR或SQL_C_BINARY

对C数据的所有类型如果 StrLen_or_Ind非SQL_NULL_DATA或SQL_DEFAULT_PARAM则驱动程序认为*DataPtr的大小是ValueType或TargetType中指定的C数据类型的大小并传送所有数据值

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	各驱动程序提供的信息（函数返回SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
01004	String data, right truncated	向输入/输出或输出参数返回的字符串或二进制数据被截断字符串的右侧被截断（函数返回SQL_SUCCESS_WITH_INFO）
07006	Restricted data type attribute violation	对绑定的参数识别为SQLBindParameter的ValueType的数据值无法转换为SQLBindParameter的ParameterType参数识别的数据类型
07S01	Invalid use of default parameter	SQLBindParameter中设置的参数值为SQL_DEFAULT_PARAM对应的参数值不具备默认值
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
22001	String data, right truncation	字符串或二进制数据被截断 SQLGetInfo中SQL_NEED_LONG_DATA_LEN信息类型为"Y"对长的参数传送了比SQLBindParameter的StrLen_or_IndPtr参数指定的数据更多的数据 SQLGetInfo中SQL_NEED_LONG_DATA_LEN信息类型为"Y"对长的列传送了比通过SQLBulkOperation增加或更新或通过SQLSetPos更新的行数据对应的缓冲区长度指定的数据更多的数据

SQLSTATE	报错	说明
22003	Numeric value out of range	分配表column相关内存时绑定的数字参数或向列传送的数据导致数值被截断 由于输入/输出参数或输出参数返回的数值被截断
22007	Invalid datetime format	对DATETIME或TIMESTAMP结构中绑定的参数或列传送的数据对各类型无效 输入/输出或输出参数被绑定到DATETIME或TIMESTAMP C结构向参数返回的值对各类型无效（函数返回SQL_SUCCESS_WITH_INFO）
22008	Datetime field overflow	绑定的DATETIME 或TIMESTAMP C结构里DATETIME表达式无效
22012	Division by zero	输入输出或输出参数的计算式为除以0

SQLSTATE	报错	说明
22015	Interval field overflow	对准确的数字INTERVAL column或参数由于INTERVAL SQL数据类型传送的数据丢失了有效数字 向一个以上的INTERVAL column或参数传送的数据转换为数字数据类型但不能表现为数字数据类型 对column或参数数据传送的数据分配到了INTERVAL SQL类型分配但在INTERVAL SQL类型无法表现为C类型值 向准确的数字或INTERVAL C column或参数传送的数据丢失了有效数字 对column或参数数据传送的数据分配到了INTERVAL C结构但不能表现为INTERVAL数据结构
22018	Invalid character value for cast specification	结果集的字符column包含不能以C缓冲区的字符集表现的字符 C类型为准确numeric或近的numericdatetimeinterval数据类型SQL类型为字符数据类型时绑定到C类型的column值不是有效的字符
HY000	General error	无特定SQLSTATE的错误

SQLSTATE	报错	说明
HY001	Memory allocation error	内存分配错误
HY008	Operation canceled	可异步处理StatementHandle调用并完成执行此函数之前StatementHandle调用了SQLCancel或SQLCancelHandle函数之后此函数再次被StatementHandle调用 多线程应用程序中调用并结束此函数之前其他线程在StatementHandle调用了SQLCancel或SQLCancelHandle
HY009	Invalid use of null pointer	DataPtr参数为空指针StrLen_or_Ind参数不是SQL_DEFAULT_PARAM 或SQL_NULL_DATA
HY010	Function sequence error	之前的函数调用不是为了SQLPutData或SQLParamData StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLPrimaryKeys时异步执行的函数还在执行中 StatementHandle调用了异步执行的函数调用此函数时异步执行的函数仍然在执行 对StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-executing变量或column的数据之前调用了此函数

SQLSTATE	报错	说明
HY013	Memory management error	无法访问内部内存或可使用的内存不足
HY019	Non-character and non-binary data sent in pieces	对参数或column调用了一次以上的SQLPutData 未用于向字符二进制或有各数据源数据类型的column传送字符C数据或传送二进制C数据
HY020	Attempt to concatenate a null value	返回SQL_NEED_DATA后调用了一次以上的SQLPutData 其中一次调用的StrLen_or_Ind参数中包含SQL_NULL_DATA 或SQL_DEFAULT_PARAM
HY090	Invalid string or buffer length	DataPtr参数不为空指针StrLen_or_Ind小于0或不同于SQL_NTS或SQL_NULL_DATA
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran** 函数
HYT01	Connection timeout expired	数据源响应请求之前连接超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	StatementHandle相关的驱动程序不支持函数

SQLSTATE	报错	说明
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄之前的函数调用返回SQL_STILL_EXECUTING并激活提醒模式时为了句柄的后处理和完成操作必须调用SQLCompleteAsync

说明

SQLPutData为了提供data-at-execution数据有两种用途一是用作SQLExecute或SQLExecDirect调用的参数数据另一个是用于调用SQLBulkOperations的更新添加行或调用SQLSetPos更新行时的column数据

当应用程序为了决定传输什么数据而调用SQLParamData时返回用于判断应用程序传输什么参数数据或在哪里查找column数据的标识此函数还返回提示应用程序调用SQLPutData的SQL_NEED_DATA应用程序向SQLPutData的DataPtr参数传送存储参数或column的实际数据的缓冲区指针

当驱动程序对SQLPutData返回SQL_SUCCESS时应用程序再次调用SQLParamData如果还需要传送数据SQLParamData将返回SQL_NEED_DATA并应用程序再次调用SQLPutData传送所有data-at-execution数据后则返回SQL_SUCCESS之后应用程序再次调用SQLParamData驱动程序向SQL_NEED_DATA与*ValuePtrPtr返回其他标识时其请求其他参数或column的数据并再次调用SQLPutData如果驱动程序返回SQL_SUCCESS则传输所有data-at-execution并可执行SQL语句或可

以处理SQLBulkOperations或SQLSetPos

Note:

应用程序只能在向字符二进制或数据源指定的数据类型传送字符C数据或二进制C数据时使用SQLPutData如果在其他条件下调用一次以上SQLPutData则返回SQL_ERROR和SQLSTATE HY019(Non-character and non-binary data sent in pieces).

SQLRowCount

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLRowCount返回受UPDATEINSERT或DELETE命令语操作影响的行数(SQLBulkOperations中的SQL_ADDSQL_UPDATE_BY_BOOKMARK或SQL_DELETE_BY_BOOKMARK操作或SQLSetPos中的SQL_UPDATESQL_DELETE操作)

语句

```
SQLRETURN SQLRowCount(
    SQLHSTMT   StatementHandle,
    SQLLEN *   RowCountPtr);
```

参数

StatementHandle

【输入】命令语句柄

RowCountPtr

【输出】返回行数的缓冲区指针UPDATEINSERTDELETE语句SQLBulkOperations的

SQL_ADDSQL_UPDATE_BY_BOOKMARK和SQL_DELETE_BY_BOOKMARK操作SQLSetPos的SQL_UPDATE或SQL_DELETE操作向*RowCountPtr返回的值是请求影响的行数或没有任何影响时返回-1.

调用SQLExecuteSQLExecDirectSQLBulkOperationsSQLSetPos或SQLMoreResults时检测数据结构的SQL_DIAG_ROW_COUNT字段设置为行数行数以具体实现的方式缓存

SQLRowCount返回缓存的行数缓存的行数有效至命令语句柄重新设置为准备或分配状态或重新执行命令语或调用SQLCloseCursor设置SQL_DIAG_ROW_COUNT字段并调用此函数时由函数的调用SQL_DIAG_ROW_COUNT字段设置为0因此SQLRowCount返回的值可能不同于SQL_DIAG_ROW_COUNT字段的值

其他命令语和函数中驱动程序可以定义向*RowCountPtr返回的值例如部分数据源可以在获取数据之前返回SELECT 命令语返回的行数

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, 或SQL_INVALID_HANDLE.

检测

SQLSTATE	报错	说明
01000	General warning	各驱动程序的信息（函数返回SQL_SUCCESS_WITH_INFO）
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLRowCount时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults并返回了SQL_PARAM_DATA_AVAILABLE检查所有已连接的参数之前调用了此函数 对StatementHandle调用SQLExecuteSQLExecDirectSQLBulkOperations或SQLSetPos 之前调用了此函数 对StatementHandle调用了异步执行函数调用此函数时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution参数或column数据之前调用了此函数
HY013	Memory management error	无法访问内部内存或可使用的内存不足

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran**函数
HYT00	Timeout expired	从数据源接收结果集之前连接超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置
HYT01	Connection timeout expired	数据源响应请求之前连接超时该限制时间可通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	StatementHandle相关的驱动程序不支持函数

说明

对命令语句柄执行的最后一个SQL命令语不是UPDATEINSERTDELETE或调用SQLBulkOperations时operation参数不是SQL_ADDSQL_UPDATE_BY_BOOKMARKSQL_DELETE_BY_BOOKMARK或之前调用SQLSetPos时operation参数不是SQL_UPDATE或SQL_DELETE时*RowCountPtr的值是驱动程序定义的值

SQLSetConnectAttr

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLSetConnectAttr设置控制连接的属性

语句

```
SQLRETURN SQLSetConnectAttr(
    SQLHDBC      ConnectionHandle,
    SQLINTEGER   Attribute,
    SQLPOINTER   ValuePtr,
    SQLINTEGER   StringLength);
```

参数

ConnectionHandle

【输入】连接句柄

Attribute

【输入】用于设置的属性

ValuePtr

【输入】Attribute相关值的指针根据Attribute的值ValuePtr可以是unsigned整数值或指向空终止字符串Attribute参数的整数类型可以不是固定长度详细内容参考说明部分

StringLength

【输入】Attribute为ODBC定义的属性ValuePtr指向字符串或二进制时此参数应为*ValuePtr的长度为了字符串数据此参数应包含字符串的字节数

Attribute为ODBC定义的属性ValuePtr为整数时忽略StringLength

Attribute为驱动程序定义属性时应用程序设置通过StringLength参数向驱动程序管理器表示属性的特性StringLength可以有以下值

• ValuePtr为字符串指针时StringLength是字符串的长度或SQL_NTS

• ValuePtr为二进制缓冲区指针时应用程序把负数的SQL_LEN_BINARY_ATTR(length)macro的结果存储于StringLength

• ValuePtr不是字符串指针或二进制字缓冲区指针时StringLength应为SQL_IS_POINTER

• ValuePtr为固定长度数据类型时StringLength是SQL_IS_INTEGER或SQL_IS_UINTEGER

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

检测

SQLSTATE	报错	说明
01000	General warning	驱动程序提供的信息（函数返回SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
01S02	Option value changed	驱动程序不支持ValuePtr指定的值 替代为相近值（函数返回SQL_SUCCESS_WITH_INFO）
08002	Connection name in use	Attribute参数为SQL_ATTR_ODBC_CURSORS驱动程序已连接数据源
08003	Connection not open	Attribute值指定为请求已打开的连接但ConnectionHandle并非连接状态
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
24000	Invalid cursor state	Attribute参数为SQL_ATTR_CURRENT_CATALOG结果集延迟
25000	Illegal operation while in a local transaction	通过设置连接属性SQL_ATTR_ENLIST_IN_DTC尝试连接分布式务的过程中本地事务被连接 已尝试连接分布式事务 已尝试连接分布式事务将本地事务的SQL_ATTR_AUTOCOMMIT设置为SQL_AUTOCOMMIT_OFF后开始了本地事务
3D000	Invalid catalog name	Attribute参数为SQL_ATTR_CURRENT_CATALOG指定的目录名称无效
HY000	General error	无特定SQLSTATE的错误

SQLSTATE	报错	说明
HY001	Memory allocation error	内存分配错误
HY008	Operation canceled	对ConnectionHandle已激活异步处理调用并完成SQLSetConnectAttr函数之前ConnectonHandle调用了SQLCancelHandle函数SQLSetConnectAttr函数再次被ConnectionHandle调用 调用并结束SQLSetConnectAttr之前从多线程应用程序的其他线程在ConnectionHandle调用了SQLCancelHandle
HY009	Invalid use of null pointer	Attribute参数识别需要字符串值的连接属性ValuePtr参数为null指针

SQLSTATE	报错	说明
HY010	Function sequence error	对ConnectionHandle相关的StatementHandle调用了异步执行函数执行过程中调用了SQLSetConnectAttr 对ConnectionHandle调用了异步执行函数调用此函数时异步执行的函数还在执行中 对与ConnectionHandle相关的命令语句柄中的一个调用了SQLExecuteSQLExecDirect或SQLMoreResults并返回了 SQL_PARAM_DATA_AVAILABLE回收有流式参数的数据之前调用了此函数 对与ConnectionHandle相关的StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperations或SQLSetPos函数并返回了SQL_NEED_DATA传送所data-at-execution参数或column之前调用了此函数 ConnectionHandle调用了SQLBrowseConnect并返回了SQL_NEED_DATASQLBrowseConnect返回SQL_SUCCESS_WIRTH_INFO或SQL_SUCCESS之前调用了此函数
HY011	Attribute cannot be set now	Attribute参数为SQL_ATTR_TXN_ISOLATION事务已打开

SQLSTATE	报错	说明
HY013	Memory management error	无法访问内部内存或可使用的内存容量不足
HY024	Invalid attribute value	指定Attribute值ValuePtr中指定的值无效 Attribute参数为SQL_ATTR_TRACEFILE或SQL_ATTR_TRANSLATE_LIBValuePtr空字符
HY090	Invalid string or buffer length	ValuePtr为字符串StringLength参数小于0但不是SQL_NTS
HY114	Driver does not support connection-level asynchronous function execution	不支持异步连接的驱动程序中应用程序要通过SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE激活异步执行功能
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran** 函数
HY121	Cursor Library and Driver-Aware Pooling cannot be enabled at the same time	驱动程序不支持

SQLSTATE	报错	说明
HYC00	Optional feature not implemented	对Attribute参数指定的值在驱动程序指定的版本中是有效的ODBC连接或命令属性但驱动程序不支持
HYT01	Connection timeout expired	数据源响应请求之前超时其限制时间可通通过SQLSetStmtAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	ConnectionHandle相关的驱动程序不支持此函数
IM009	Unable to load translation DLL	驱动程序无法加载为连接指定的事务DDL此报错只有在Attribute为SQL_ATTR_TRANSLATE_LIB时返回
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的 .
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄之前的函数调用返回SQL_STILL_EXECUTING提醒模式生效时对句柄的后处理和结束操作必须调用SQLCompleteAsync
S1118	Driver does not support asynchronous notification	已设置SQL_ATTR_ASYNC_STMT_EVENT 但驱动程序不支持异步通知

说明

应用程序可以在分配和解除连接的过程中随时调用SQLSetConnectAttr应用程序为了连接成功设置的所有连接与命令属性保留到调用SQLFreeHandle例如应用程序在连接数据源之前调用SQLSetConnectAttr则在应用程序连接数据源时即使驱动程序调用SQLSetConnectAttr失败其属性仍有效如果应用程序设置各个驱动程序的属性则应用程序即使连接其他驱动程序其属性依然有效

> Note:
>
> ODBC 3.x不再通过调用SQLSetConnectAttr在连接级别设置命令属性的功能ODBC 3.x的应用程序不能在连接级别设置命令属性ODBC 3.x无法在连接级别设置除SQL_ATTR_METADATA_ID和SQL_ATTR_ASYNC_ENABLE外的其他命令属性这两个属性均为连接属性和命令属性可以在连接级别或命令语级别设置如果在连接级别需要通过设置ODBC 2.x命令选项的ODBC 2.x应用程序进行操作时ODBC 3.x需要支持此功能

部分连接属性只能在连接之前设置部分属性只能在连接之后设置以下表格说明这种连接属性

Attribute	Set before or after connection?
SQL_ATTR_ACCESS_MODE	Either[1]
SQL_ATTR_ASYNC_DBC_EVENT	Either
SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE	Either[4]
SQL_ATTR_ASYNC_DBC_PCALLBACK	Either
SQL_ATTR_ASYNC_DBC_PCONTEXT	Either

Attribute	Set before or after connection?
SQL_ATTR_ASYNC_ENABLE	Either[2]
SQL_ATTR_AUTO_IPD	Either
SQL_ATTR_AUTOCOMMIT	Either[5]
SQL_ATTR_CONNECTION_DEAD	After
SQL_ATTR_CONNECTION_TIMEOUT	Either
SQL_ATTR_CURRENT_CATALOG	Either[1]
SQL_ATTR_DBC_INFO_TOKEN	After
SQL_ATTR_ENLIST_IN_DTC	After
SQL_ATTR_LOGIN_TIMEOUT	Before
SQL_ATTR_METADATA_ID	Either
SQL_ATTR_OLDPWD	Before
SQL_ATTR_ODBC_CURSORS	Before
SQL_ATTR_PACKET_SIZE	Before
SQL_ATTR_QUIET_MODE	Either
SQL_ATTR_TRACE	Either
SQL_ATTR_TRACEFILE	Either
SQL_ATTR_TRANSLATE_LIB	After
SQL_ATTR_TRANSLATE_OPTION	After

Attribute	Set before or after connection?
SQL_ATTR_TXN_ISOLATION	Either[3]

[1] 根据驱动程序可以在连接之前或之后设置SQL_ATTR_ACCESS_MODE和SQL_ATTR_CURRENT_CATALOG但使用多种驱动程序的应用程序部分驱动程序不支持在连接后变更因此需要在连接之前设置

[2] 应在命令语激活前设置SQL_ATTR_ASYNC_ENABLE

[3] 连接中没有已打开的事务时才能设置SQL_ATTR_TXN_ISOLATION部分连接属性在数据源不支持*ValuePtr指定值时替代为相近值此时驱动程序返回SQL_SUCCESS_WITH_INFO和SQLSTATE 01S02 (Option value changed)例如attribute为SQL_ATTR_PACKET_SIZE *ValuePtr超过最大packet大小时驱动程序替代最大值为了查看替代值应用程序调用SQLGetConnectAttr

[4] 如果在连接打开之前设置SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE驱动程序管理器在调用SQLBrowseConnectQLConnect或SQLDriverConnect的过程中加载驱动程序时设置驱动程序的属性调用SQLBrowseConnectSQLConnect或SQLDriverConnect之前驱动程序管理器无法确定连接什么驱动程序此驱动程序是否支持异步连接操作因此驱动程序管理器返回SQL_SUCCESS但是驱动程序不支持异步连接操作时SQLBrowseConnectSQLConnect或SQLDriverConnect的调用将失败

[5] SQL_ATTR_AUTOCOMMIT设置为FALSE时如果API返回SQL_ERROR则应用程序为了保障事务的一致性需要调用SQLEndTran(SQL_ROLLBACK)

*ValuePtr缓冲区的信息类型取决于指定的attributeSQLSetConnectAttr以空终止字符或整数值的形式接收属性信息SQLSetConnectAttr的ValuePtr参数指向的字符串长度为 StringLength字节长度

长度由属性定义时与ODBC 2.x或之前介绍的所有属性相同忽略StringLength参数

Attribute	ValuePtr contents
SQL_ATTR_ACCESS_MODE (ODBC 1.0)	SQLUINTEGER值SQL_MODE_READ_ONLY为默认值 SQL_MORE_REAT_ONLY表示不会为了支持更新操作的SQL命令而请求连接用于驱动程序或数据源此模式可用于驱动程序或数据源的适当的加锁策略事务管理以及优化其他领域驱动程序不用防止这些内容传送到数据源只读连接中请求处理非只读SQL命令语时驱动程序和数据源以实现定义运行
SQL_ATTR_ASYNC_DBC_EVENT (ODBC 3.8)	驱动程序不支持

Attribute	ValuePtr contents
SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE (ODBC 3.8)	SQLUINTERGER值激活或禁用连接句柄所选的函数的异步执行 总是同步 SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE设置（不返回SQL_STILL_EXECUTING） 命令语操作的异步执行通过SQL_ATTR_ASYNC_ENABLE激活 • SQL_ASYNC_DBC_ENABLE_ON：激活指定的连接与相关函数的异步操作 • SQL_ASYNC_DBC_ENABLE_OFF：禁用指定的连接与相关函数的异步操作
SQL_ATTR_ASYNC_DBC_PCALLBACK (ODBC 3.8)	驱动程序不支持
SQL_ATTR_ASYNC_DBC_PCONTEXT (ODBC 3.8)	驱动程序不支持
SQL_ATTR_ASYNC_ENABLE (ODBC 3.0)	驱动程序不支持
SQL_ATTR_AUTO_IPD (ODBC 3.0)	驱动程序不支持

SQL_ATTR_AUTOCOMMIT (ODBC 1.0)	指定使用自动提交或手动提交模式的SQLUINTEGER值 部分数据源在连接提交命令语时删除访问计划并关闭游标自动提交模式下执行各Non-query命令语后或游标对语句关闭时可能发生这种情况更详细的内容参考**SQLGetInfo**的SQL_CURSOR_COMMIT_BEHAVIOR和SQL_CURSOR_ROLLBACK_BEHAVIOR信息类型 自动提交模式下执行批处理时有两种情况整个批处理作为一个可自动处理的单位处理或批处理的各命令语作为可自动处理的单位处理特定数据源可支持这种两种操作并需要选择一种由驱动程序定义是否批量自动提交或在批量处理中自动处理各个命令 • SQL_AUTOCOMMIT_OFF：驱动程序使用手动提交模式应用程序需要通过SQLEndTrans指定提交或回滚事务 • SQL_AUTOCOMMIT_ON：驱动程序使用自动提交模式各命令语执行后立即被提交连接中已打开的所有事务在SQL_ATTR_AUTOCOMMIT设置为SQL_AUTOCOMMIT_ON时被提交

Attribute	ValuePtr contents
SQL_ATTR_CONNECTION_DEAD(ODBC 3.5)	驱动程序不支持
SQL_ATTR_CONNECTION_TIMEOUT (ODBC 3.0)	驱动程序不支持
SQL_ATTR_CURRENT_CATALOG (ODBC 2.0)	驱动程序不支持
SQL_ATTR_DBC_INFO_TOKEN (ODBC 3.8	驱动程序不支持
SQL_ATTR_ENLIST_IN_DTC (ODBC 3.0)	驱动程序不支持
SQL_ATTR_LOGIN_TIMEOUT (ODBC 1.0)	返回到应用程序之前对应等待完成登录请求的时间（秒）的SQLUINTEGER值默认值取决于驱动程序 ValuePtr为0时禁用时间限制连接尝试无限等待 指定的登录请求时间限制超过最长登录时间限制时驱动程序替代其值并返回SQLSTATE 01S02 (Option value changed)

Attribute	ValuePtr contents
SQL_ATTR_METADATA_ID (ODBC 3.0)	决定目录函数的字符串参数处理方法的 SQLUINTEGER值默认值为SQL_FALSE SQL_TRUE时目录函数的字符串参数被认为标识符并不区分大小写未区分的字符串时驱动程序将删除所有后行的空白并将字符串转换为大写区分的字符串时驱动程序将删除前行或后行空格并使用原有分隔字符之间的字符这样的参数中一个为空指针时函数返回SQL_ERROR和SQLSTATE HY009 (invalid use of null pointer) SQL_FALSE时目录函数的字符串参数不会被认为标识符并区分大小写根据参数可处理为字符串模式与否 拥有值目录的SQLTables的TableType参数不受此属性的影响 SQL_ATTR_METADATA_ID也可在命令语阶段设置（是命令属性的唯一的连接属性） 更详细的内容参考**目录函数的参数**
SQL_ATTR_OLDPWD	之前密码字符串的SQLPOOINTER其值为只写专用应在连接服务器之前设置

Attribute	ValuePtr contents
SQL_ATTR_ODBC_CURSORS (ODBC 2.0)	驱动程序不支持
SQL_ATTR_PACKET_SIZE (ODBC 2.0)	驱动程序不支持
SQL_ATTR_QUIET_MODE (ODBC 2.0)	驱动程序不支持
SQL_ATTR_TRACE (ODBC 1.0)	驱动程序不支持
SQL_ATTR_TRACEFILE (ODBC 1.0)	驱动程序不支持
SQL_ATTR_TRANSLATE_LIB (ODBC 1.0)	驱动程序不支持
SQL_ATTR_TRANSLATE_OPTION (ODBC 1.0)	驱动程序不支持
SQL_ATTR_TXN_ISOLATION (ODBC 1.0)	设置当前连接的事务的隔离级别的32-bit mask应用程序为了提交或回滚连接中打开的所有事务在调用SQLSetConnectAttr之前使用此选项调用SQLSetConnectAttr InfoType可通过调用SQL_TXN_ISOLATION_OPTIONS的SQLGetInfo决定ValuePtr的有效值 事务隔离级别参考**SQLGetInfo**的SQL_DEFAULT_TXN_ISOLATION信息类型

Table 2-9 ODBC属性

[1] 这种函数只有在描述符为非应用程序描述符而是为定义描述符时才能异步调用

SQLSetConnectOption

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 2.0 的SQLSetConnectOption函数在ODBC 3.x中被替代为SQLSetConnectAttr详细内容参考

SQLSetConnectAttr

SQLSetCursorName

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLSetCursorName在已激活的目录中连接游标名如果应用程序不调用SQLSetCursorName则驱动程序则生成处理SQL命令所需的游标名

语句

```
SQLRETURN SQLSetCursorName(
    SQLHSTMT      StatementHandle,
    SQLCHAR *     CursorName,
    SQLSMALLINT   NameLength);
```

参数

SatatementHandle

【输入】命令语句柄

CursorName

【输入】游标名为了有效处理游标名前后空间不应包含空格如果游标名包含限制的标识

符则分隔符应位于游标名的第一个字符

NameLength

【输入】*CursorName 字符串的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	驱动程序提供的信息 (函数返回 SQL_SUCCESS_WITH_INFO)
01004	String data, right truncated	游标名长度超过了最大值仅使用最大允许数量的字符串
24000	Invalid cursor state	StatementHandle对应的命令语在执行中或用作位置指定游标
34000	Invalid cursor name	*CursorName指定的游标名超过了驱动程序的最大值或以SQLCUR或SQL_CUR开始而无效
3C000	Duplicate cursor name	*CursorName中指定的游标名已存在
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误
HY009	Invalid use of null pointer	CursorName参数为空指针

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了异步执行函数但调用SQLSetCursorName函数时此异步执行功能还在执行中 调用SQLSetCursorName后对StatementHandle调用了异步执行函数 对StatementHandle调用了SQLExecuteSQLExeDirectSQLBulkOperations或SQLSetPos函数并返回了SQL_NEED_DATA时在传送所有data-at-execution参数或column之前调用了此函数
HY013	Memory management error	无法访问内部内存或可使用的内存量不足
HY090	Invalid string or buffer length	NameLength参数小于0（不是SQL_NTS）
HY117	Connection is suspended due to unknown tracsaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran**

SQLSTATE	报错	说明
HYT01	Connection timeout expired	数据源响应请求之前连接超时连接超时周期可通过SQLSetConnectAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	驱动程序不支持该函数

说明

游标名仅用于更新位置指定命令和删除位置指定命令语（例如：UPDATE table-name ... WHERE CURRENT OF cursor-name）如果应用程序在执行语句命令时未调用SQLSetCursorName定义游标名则驱动程序生成不超过18个字符长度并以SQL_CUR开头的游标名

游标名在连接中必须是唯一的驱动程序定义游标名的最大长度为了最大相互兼容性建议游标名长度不超过18个字符ODBC 3.x中如果用双引号（"）标识符引住游标名则认为区分大小写并可以包含SQL语句不允许的字符或空格或关键字等特殊符号 如果一定要区分大小写游标名要用双引号引住

游标名维持到使用SQLFreeHandle删除相关命令语为止为了再次赋予已分配游标或准备（prepared）状态的命令语的游标名可调用SQLSetCursorName

SQLSetDescField

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLSetDescField设置描述符记录的一个字段值

语句

```
SQLRETURN SQLSetDescField(
    SQLHDESC      DescriptorHandle,
    SQLSMALLINT   RecNumber,
    SQLSMALLINT   FieldIdentifier,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    BufferLength);
```

参数

DescriptorHandle

【输入】描述符句柄

RecNumber

【输入】指包含应用程序要设置的字段的描述符记录描述符记录从0开始记录数字0为书签记录标头的字段忽略RecNumber参数

FieldIdentifier

【输入】指要设置的描述符的字段

ValuePtr

【输入】指包含描述符信息或整数值的缓冲区数据类型取决于FieldIdentifier值ValuePtr为整数值时根据FieldIdentifier参数值认为8字节(SQLLEN)4字节(SQLINTEGER)2字节(SQLSMALLINT)

BufferLength

【输入】FieldIdentifier为ODBC定义的字段并ValuePtr指向字符串或二进制缓冲区时此参数必须是*ValuePTr的长度对字符串数据此参数应包含字符串的字节数

FieldIdentifier为ODBC定义的字段ValuePtr为整数时忽略BufferLength

FieldIdentifier为驱动程序定义的字段时应用程序通过设置BufferLength参数在驱动程序管理器中显示字段的特性BufferLength可以有以下值

• ValuePtr为字符串指针时BufferLength是字符串长度或SQL_NTS

• ValuePtr为二进制缓冲区指针时应用程序在BufferLengfth中存储SQL_LEN_BINARY_ATTR(length) macro的结果此BufferLength存储负数值

• ValuePtr包含固定长度值时BufferLength为SQL_IS_INTEGERSQL_IS_UINTEGERSQL_IS_SMALLINT或SQL_IS_USMALLINT中的一个

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	Genaral warning	各驱动程序的信息（函数返回SQL_SUCCSS_WITH_INFO）
01S02	Option value changed	驱动程序（ValuePtr值为整数时）不支持*ValuePtr定义的值或ValuPtr的值或*ValuPtr 不符合实现操作条件而被驱动程序替代为相近值（函数返回SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
07009	Invalid descriptor index	FieldIdentifier参数为记录字段RecNumber参数为0时DescriptorHandle参数参考了IPD句柄 RecNumber参数小于0DescriptorHandle参数参考了ARD或APD RecNumber参数大于数据源支持的column或参数的最大值DescriptorHandle参数参考了APD或ARD FieldIdentifier参数为SQL_DESC_COUNT*ValuePtr参数小于0 RecNumber参数为0DescriptorHandle参数参考了内部分配的APD（该错误直到执行之前无法确定明确分配的应用程序描述符为APD或ARD因此对明确分配的应用程序描述符不会产生）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
22001	String data, right truncated	FieldIdentifier参数为SQL_DESC_NAMEBufferLength参数的值大于SQL_MAX_IDENTIFIER_LEN
HY000	General error	无特定SQLSTATE的错误
HY001	Memory allocation error	内存分配错误

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的DescriptorHandle调用了异步执行函数调用SQLSetDescField函数时异步执行函数仍处于执行状态 对DescriptorHandle相关的StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperations或SQLSetPos 并返回了SQL_NEED_DATA传送Data-at-execution参数或列的所有数据之前调用了此函数 对DescriptorHandle相关的连接句柄调用了异步执行函数调用SQLSetDescField时此异步执行功能还在执行中 对DescriptorHandle相关的命令语句柄中的一个调用了SQLExecuteSQLExecDirect或SQLMoreResults并返回了SQL_PARAM_DATA_AVAILABLE获取所有的流参数的数据之前调用了此函数
HY013	Memory management error	无法访问内部内存或可使用的内存容量不足
HY016	Cannot modify an implementation row descriptor	DescriptorHandle参数与IRD有关FieldIdentifier参数不是SQL_DESC_ARRAY_STATUS_PTR或SQL_DESC_ROWS_PROCESSED_PTR

SQLSTATE	报错	说明
HY021	Inconsistent descriptor information	SQL_DESC_TYPE和SQL_DESC_DATETIME_INTERVAL_CODE字段不是ODBC SQL驱动程序指定的SQL类型或ODBC C的有效类型 完整性检查中未查到描述符信息的一致性
HY090	Invalid string or buffer length	*ValuePtr为字符串BufferLength小于0（不是SQL_NTS） 驱动程序为ODBC 2.x描述符为ARDColumnNumber参数设置为0时BufferLength值并非4
HY091	Invalid descriptor field identifier	FieldIdentifier参数指定的值不是ODBC定义字段或实现定义值 FieldIdentifier参数DescriptorHandle参数无效 FieldIdentifier参数为ODBC定义的字段并为只读专用
HY092	Invalid attribute/option identifier	*ValuePtr对FieldIdentifier参数无效 FieldIdentifier参数为SQL_DESC_UNNAMEDValuePtr为SQL_NAMED

SQLSTATE	报错	说明
HY105	Invalid parameter type	SQL_DESC_PARAMETER_TYPE字段指定的值无效（详细内容参考SQLBindParameter的 **InputOutputType 参数** ）
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran** 函数
HYT01	Connection timeout expired	数据源响应请求之前超时连接超时周期可通过SQLSetConnectAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	驱动程序不支持此函数

说明

应用程序每次调用SQLSetDescField时均可设置一个描述符字段一次调用可在一个描述符中设置一个字段可设置的字段时可调用此函数设置描述符类型的字段

> Note:
>
> 调用SQLSetDescField失败时不定义识别为RecNumber参数的描述符记录的内容

为了通过一次调用函数设置多个描述符字段可以调用其他函数SQLSetDescRec函数可以设置绑

定数据类型的列或影响参数缓冲区的多种字段(SQL_DESC_TYPE,
SQL_DESC_DATETIME_INTERVAL_CODE, SQL_DESC_OCTET_LENGTH, SQL_DESC_PRECISION,
SQL_DESC_SCALE, SQL_DESC_DATA_PTR, SQL_DESC_OCTET_LENGTH_PTR,
SQL_DESC_INDICATOR_PTR)

SQLBindCol或SQLBindParameter可用于完成设置列和参数这些函数可以通过一次调用设置描述符字段的group

可在绑定指针添加offset后调用SQLSetDescField变更绑定缓冲区(SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR) 其类似于SQL_DESC_DATA_TYPE在不变更其他字段的情况下变更SQL_DESC_DATA_PTR应用程序在未调用SQLBindCol或SQLBindParameter的情况下变更绑定缓冲区

应用程序调用SQLSetDescField设置其他字段而不是SQL_DESC_COUNT或SQL_DESC_DATA_PTRSQL_DESC_OCTET_LENGTH_PTR等延迟字段时则将解除记录的绑定

可以与适当的FieldIdentifier同时调用SQLSetDescField设置描述符头部字段很多头部字段为命令属性因此也可以通过调用SQLSetStmtAttr设置应用程序可以在未获取描述符句柄的情况下优先设置描述符字段调用SQLSetDescField时设置头部字段时RecNumber参数被忽略

0 RecNumber用于设置书签字段

> Note:
> 为了设置书签字段需要在调用SQLSetDescField之前设置命令语属性SQL_ATTR_USE_BOOKMARKS这不是强制性的但是推荐事项

描述符字段的设置顺序

调用SQLSetDescField设置描述符字段时应用程序需要按以下顺序执行

1. 应用程序首先要设置SQL_DESC_TYPESQL_DESC_CONCISE_TYPE或SQL_DESC_DATETIME_INTERVAL_CODE字段
2. 设置这些字段中的一个后应用程序可以设置数据类型的属性驱动程序可以将数据类型设置字段设置为符合数据类型的默认值类型属性字段的自动默认化保证应用程序指定数据类型后始终为描述符可使用的准备状态应用程序明确设置数据类型属性时将覆盖默认属性
3. 设置第一阶段的字段中的一个与数据类型属性后应用程序可以设置SQL_DESC_DATA_PTR其立即检查描述符字段的一致性如果应用程序变更数据类型或属性后设置SQL_DESC_DATA_PTR字段则驱动程序将SQL_DESC_DATA_PTR设置为空指针并解除记录的绑定其在可使用描述符记录之前强制性的使应用程序按照顺序完成恰当的步骤

描述符字段的初始化

分配描述符时描述符字段可以初始化为默认值或没有默认值的初始化或可以初始化为未定义描述符类型的值以下表说明各描述符类型的各字段的初始化D表示拥有默认值的初始化字段ND表示没有默认值的初始化字段数字为字段的默认值以下表还说明字段是否为读/写或只读字段

准备（prepared）或执行命令语并生成IRD后IRD的字段才能拥有默认值而不是分配命令语句柄或描述符时生成IRD之前对字段的任何访问IRD均返回错误

部分描述符字段不是对所有描述符类型(ARD, IRD, APD, IPD)定义而对一个以上的描述符类型定义如果字段未对描述符的类型定义则使用其描述符的函数不需要这些

SQLGetDescField可访问的字段未必都能用SQLSetDescField设置SQLSetDescField可设置的字段

列如以下表格

以下表格说明头部字段的初始化

头部字段名称	类型	R/W	默认值
SQL_DESC_ALLOC_TYPE	SQLSMALLINT	ARD:R APD:R IRD:R IPD: R	ARD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit APD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit IRD: SQL_DESC_ALLOC_AUTO IPD: SQL_DESC_ALLOC_AUTO
SQL_DESC_ARRAY_SIZE	SQLULEN	ARD:R/W APD:R/W IRD:Unused IPD:Unused	ARD:[1] APD:[1] IRD: Unused IPD: Unused

头部字段名称	类型	R/W	默认值
SQL_DESC_ARRAY_STATUS_PTR	SQLUSMALLINT*	ARD:R/W APD:R/W IRD:R/W IPD:R/W	ARD:Null ptr APD:Null ptr IRD:Null ptr IPD:Null ptr
SQL_DESC_BIND_OFFSET_PTR	SQLLEN*	ARD:R/W APD:R/W IRD:Unused IPD:Unused	ARD:Null ptr APD:Null ptr IRD:Unused IPD:Unused
SQL_DESC_BIND_TYPE	SQLINTEGER	ARD:R/W APD:R/W IRD:Unused IPD:Unused	ARD: SQL_BIND_BY_COLUMN APD: SQL_BIND_BY_COLUMN IRD: Unused IPD: Unused
SQL_DESC_COUNT	SQLSMALLINT	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD:0 APD:0 IRD:D IPD: 0
SQL_DESC_ROWS_PROCESSED_PTR	SQLULEN*	ARD:Unused APD:Unused IRD:R/W IPD:R/W	ARD:Unused APD:Unused IRD:Null ptr IPD:Null ptr

[1] 只有在驱动程序自动生成IPD时定义字段否则不定义这些如果应用程序尝试设置这些字段则返回SQLSTATE HY091(Invalid descriptor field idenfitier)

以下表格说明记录字段的初始化

记录字段名称	类型	R/W	默认值
SQL_DESC_AUTO_UNIQUE_VALUE	SQLINTEGER	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD:Unused APD:Unused IRD:D IPD:Unused
SQL_DESC_BASE_COLUMN_NAME	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD:Unused APD:Unused IRD:D IPD:Unused
SQL_DESC_BASE_TABLE_NAME	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD:Unused APD:Unused IRD:D IPD:Unused
SQL_DESC_CASE_SENSITIVE	SQLINTEGER	ARD:Unused APD:Unused IRD:R IPD:R	ARD:Unused APD: Unused IRD: D IPD: D[1]

记录字段名称	类型	R/W	默认值
SQL_DESC_CATALOG_NAME	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD:Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_CHAR_LENGTH_UNITS	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD:Unused APD:Unused IRD:D IPD:Unused
SQL_DESC_CONCISE_TYPE	SQLSMALLINT	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD: SQL_C_ DEFAULT APD: SQL_C_ DEFAULT IRD: D IPD: ND
SQL_DESC_DATA_PTR	SQLPOINTER	ARD:R/W APD:R/W IRD:Unused IPD:Unused	ARD:Null ptr APD:Null ptr IRD: Unused IPD: Unused[2]
SQL_DESC_DATETIME_INTERVAL_CODE	SQLSMALLINT	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD: ND APD: ND IRD: D IPD: ND

记录字段名称	类型	R/W	默认值
SQL_DESC_DATETIME_INTERVAL_PRECISION	SQLINTEGER	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_DISPLAY_SIZE	SQLLEN	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_FIXED_PREC_SCALE	SQLSMALLINT	ARD:Unused APD:Unused IRD:R IPD:R	ARD: Unused APD: Unused IRD: D IPD: D[1]
SQL_DESC_INDICATOR_PTR	SQLLEN *	ARD:R/W APD:R/W IRD:Unused IPD:Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_LABEL	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD: Unused APD: Unused IRD: D IPD: Unused

记录字段名称	类型	R/W	默认值
SQL_DESC_LENGTH	SQLULEN	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_LITERAL_PREFIX	SQLCHAR *	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LITERAL_SUFFIX	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LOCAL_TYPE_NAME	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:R	ARD: Unused APD: Unused IRD: D IPD: D[1]
SQL_DESC_NAME	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:R/W	ARD: Unused APD: Unused IRD: D IPD: D[1]

记录字段名称	类型	R/W	默认值
SQL_DESC_NULLABLE	SQLSMALLINT	ARD:Unused APD:Unused IRD:R IPD:R	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_NUM_PREC_RADIX	SQLINTEGER	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_OCTET_LENGTH	SQLLEN	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_OCTET_LENGTH_PTR	SQLLEN *	ARD:R/W APD:R/W IRD:Unused IPD:Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_PARAMETER_TYPE	SQLSMALLINT	ARD:Unused APD:Unused IRD:Unused IPD: R/W	ARD: Unused APD: Unused IRD: Unused IPD: D =SQL_PARAM_INPUT

记录字段名称	类型	R/W	默认值
SQL_DESC_PRECISION	SQLSMALLINT	ARD:R/W APD:R/W IRD:R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_ROWVER	SQLSMALLINT	ARD:Unused APD:Unused IRD:R IPD:R	ARD: Unused APD: Unused IRD: ND IPD: ND
SQL_DESC_SCALE	SQLSMALLINT	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_SCHEMA_NAME	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_SEARCHABLE	SQLSMALLINT	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD: Unused APD: Unused IRD: D IPD: Unused

记录字段名称	类型	R/W	默认值
SQL_DESC_TABLE_NAME	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_TYPE	SQLSMALLINT	ARD:R/W APD:R/W IRD:R IPD:R/W	ARD: SQL_C_DEFAULT APD: SQL_C_DEFAULT IRD: D IPD: ND
SQL_DESC_TYPE_NAME	SQLCHAR *	ARD:Unused APD:Unused IRD:R IPD:R	ARD: Unused APD: Unused IRD: D IPD: D[1]
SQL_DESC_UNNAMED	SQLSMALLINT	ARD:Unused APD:Unused IRD:R IPD:R/W	ARD: ND APD: ND IRD: D IPD: ND

记录字段名称	类型	R/W	默认值
SQL_DESC_UNSIGNED	SQLSMALLINT	ARD:Unused APD:Unused IRD:R IPD:R	ARD: Unused APD: Unused IRD: D IPD: D[1]
SQL_DESC_UPDATABLE	SQLSMALLINT	ARD:Unused APD:Unused IRD:R IPD:Unused	ARD: Unused APD: Unused IRD: D IPD: Unused

[1] 只有在驱动程序自动生成IPD时定义字段否则不定义这些如果应用程序尝试设置字段则返回SQLSTATE HY091(Invalid descriptor field idenfitier)

[2] IPD的SQL_DESC_DATA_PTR字段强制检查一致性后续调用SQLGetDescField或SQLGetDescRec时驱动程序不用返回通过SQL_DESC_DATA_PTR设置的值

FieldIdentifier 参数

FieldIdentifier参数表示要设置的描述符字段描述符由描述符header的以下头部字段部分说明的描述符header组成描述符记录由以下头部字段中说明的描述符记录组成

头部字段

各个描述符由以下字段构成

SQL_DESC_ALLOC_TYPE[All] (只读)

只读SQLSMALLINT的此头部字段明示描述符是由驱动程序自动分配还是由应用程序指定分配应用程序可以获取此字段但无法修改当驱动程序自动分配描述符时字段由驱动程序设置为SQL_DESC_ALLOC_AUTO

SQL_DESC_ARRAY_SIZE[Application descriptors]

存在于ARD并是SQLULEN的此头部字段明示行集合的行数其是通过调用SQLFetchSQLFetchScroll返回或通过调用SQLBulkOperationsSQLSetPos运行的行数

存在于APD并是SQLULEN的此头部字段明示各参数值的数量

此字段的默认值为1当SQL_DESC_ARRAY_SIZE大于1时APD或ARD的SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTR及SQL_DESC_OCTET_LENGTH_PTR指数组各数组的常数与此字段的值相同

存在于ARD的此字段是SQL_ATTR_ROW_ARRAY_SIZE属性可通过调用SQLSetStmtAttr进行设置APD的字段也是SQL_ATTR_PARAMSET_SIZE属性可通过调用SQLSetStmtAttr进行设置

SQL_DESC_ARRAY_STATUS_PTR[All]

对各描述符类型SQLUSMALLINT*的头部字段指SQLUSMALLINT值的数组这些数组的名称是行状态数组(IRD)参数状态数组(IPD)行运算数组(ARD)参数运算数组(APD)

IRD的此头部字段指包含调用SQLBulkOperationsSQLFetchSQLFetchScroll或SQLSetPos后的状态值的行状态数组应用程序分配SQLUSMALLINT数组并使其字段指向数组此字段默认为空指针除将SQL_DESC_ARRAY_STATUS_PTR字段设置为空指针外驱动程序生成数组

Caution:

应用程序设置IRD的SQL_DESC_ARRAY_STATUS_PTR字段指向的行状态数组的要素时不定义驱动程序的行为

通过调用SQLBulkOperationsSQLFetchSQLFetchScroll或SQLSetPos初始化数组该调用不返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO时不定义该字段指向的数组内容数组的要素可以包含以下值

- SQL_ROW_SUCCESS：成功获取行最后获取行后没有任何变化
- SQL_ROW_SUCCESS_WITH_INFO：成功获取行最后获取行后没有任何变化但返回了对行的告警
- SQL_ROW_ERROR：获取行的过程中报错
- SQL_ROW_UPDATED:成功获取行最后获取行后被更新如果再次获取行则状态将变为SQL_ROW_SUCCESS
- SQL_ROW_DELETED：最后获取后行被删除
- SQL_ROW_ADDED：由SQLBulkOperations插入行如果再次获取行则状态为SQL_ROW_SUCCESS
- SQL_ROW_NOROW：行集合与结果集的最后重叠没有对应行状态数组要素返回的行

可通过SQL_ATTR_ROW_STATUS_PTR属性调用SQLSetStmtAttr设置IRD的此字段

IRD的SQL_DESC_ARRAY_STATUS_PTR字段仅在返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO后有效如果返回的代码不是这两个中的一个则不定义SQL_DESC_ROWS_PROCESSED_PTR的指向的内容

IPD的此头部字段指调用SQLExecute或SQLExecDirect后包含各参数状态信息的参数状态数组如果调用SQLExecute或SQLExecDirect后未返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO则不定义此字段指向的数组内容应用程序应分配SQLUSMALLINT数组并使其字段指向数组除了将SQL_DESC_ARRAY_STATUS_PTR字段设置为空指针外驱动程序生成数组数组的要素包含以下值

- SQL_PARAM_SUCCESS：对此参数集成功执行SQL命令语

- SQL_PARAM_SUCCESS_WITH_INFO：对此参数集成功执行SQL命令语但有对检测数据结构体可用的告警
- SQL_PARAM_ERROR：处理此参数集时报错详细错误信息可查看检测数据结构体
- SQL_PARAM_UNUSED：处理之前的几个参数时报错或APD的SQL_DESC_ARRAY_STATUS_PTR字段定义的数组的参数集中设置了SQL_PARAM_IGNORE因此未使用
- SQL_PARAM_DIAG_UNAVAILABLE：无法使用检测信息例如驱动程序把参数的数组识别为单一的单元因此不生成错误信息的level

可通过SQL_ATTR_PARAM_STATUS_PTR属性调用SQLSetStmtAttr设置IPD的此字段

ARD中的此字段为了设置在SQLSetPos运算中是否忽略对应行而指向应用程序设置的值的行运算数组数组的要素可包含以下值

- SQL_ROW_PROCEED：行包含于使用SQLSetPos的批量运算（此设置不保证运算发生在该行如果行是IRD行状态数组的SQL_ROW_ERROR状态则驱动程序无法执行行运算）
- SQL_ROW_IGNORE：在使用SQLSetPos的批量运算中排除行

未设置数组的要素时批量运算包含所有行ARD的SQL_DESC_ARRAY_STATUS_PTR字段中的值为空指针时批量运算包含所有行解释为如指针指向有效的数组并数组的所有要素为SQL_ROW_PROCEED相同如果将数组的所有要素设置为SQL_ROW_IGNORE则不变更被忽略的行状态数组中的值

可通过SQL_ATTR_ROW_OPERATION_PTR属性调用SQLSetStmtAttr设置ARD中的此字段

为了表示调用SQLExecute或SQLExecDirect时是否忽略此参数集ARD中的此头部字段指向应用程序可设置的值的参数运算数组数组的要素包含以下值

- SQL_PARAM_PROCEED：SQLExecute或SQLExecDirect调用包含参数集
- SQL_PARAM_IGNORE：SQLExecute或SQLExecDirect调用不包含参数集

未设置数组要素时数组的所有参数集用于调用SQLExecute或SQLExecDirectAPD的SQL_DESC_ARRAY_STATUS_PTR字段值为空指针时使用所有参数集解释为如指针指向有效的数组并所有要素为SQL_PARAM_PROCEED的数组相同

可与SQL_ATTR_PARAM_OPERATION_PTR属性同时调用SQLSetStmtAttr设置APD的此字段

SQL_DESC_BIND_OFFSET_PTR[Application descriptors]

SQLLEN*的此头部字段指绑定的offset其默认设置为空指针此字段不是空指针时驱动程序逆向参考指针并将逆向参考的值添加到描述符记录中(SQL_DESC_DATA_PTR_SQL_DESC_INDICATOR_PTR及SQL_DESC_OCTET_LENGTH_PTR)非空值的延迟的字段绑定时使用新的指针值

绑定offset总是直接追加于SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTR和SQL_DESC_OCTET_LENGTH_PTR字段如果变更offset值新的值继续直接追加于各描述符字段值新的offset并非追加于字段的之前的offset值

此字段为延迟的字段此字段不在设置时间点使用而是在后续需要查看数据缓冲区地址时由驱动程序使用

可通过SQL_ATTR_ROW_BIND_OFFSET_PTR属性调用SQLSetStmtAttr设置ARD中的此字段也通过SQL_ATTR_PARAM_BIND_OFFSET_PTR属性调用SQLSetStmtAttr设置ARD中的此字段

详细内容参考**SQLFetchScroll**或**SQLBindParameter**

SQL_DESC_BIND_TYPE[Application descriptors]

SQLUINTEGER的此头部字段用于设置绑定的方向

ARD中的此字段定义相关命令语句柄调用SQLFetchScroll或SQLFetch时的绑定方向

选择column的列式绑定需要将此字段设置为SQL_BIND_BY_COLUMN（默认值）

可通过SQL_ATTR_ROW_BIND_TYPE属性调用SQLSetStmtAttr设置ARD中的此字段

此字段指定用于动态参数的绑定方向

选择参数的列式绑定需要将此字段设置为SQL_BIND_BY_COLUMN（默认值）

可通过SQL_ATTR_PARAM_BIND_TYPE属性调用QLSetStmtAttr设置此字段

SQL_DESC_COUNT[All]

此SQLSMALLINT标头字段定义包含数据的最高级别记录的1-based索引驱动程序设置描述符的数据结构体时为了显示有多少条重要的记录必须设置SQL_DESC_COUNT应用程序分配数据结构体的实例时不用指定需要预留的记录存储空间如应用程序指定记录的内容驱动程序为了保证描述符句柄显示足够大小的数据结构体而执行必要的请求操作

SQL_DESC_COUNT不是绑定的所有数据列或所有参数的数量而是最高级别记录的数量解除最高级别column或参数的绑定时SQL_DESC_COUNT将变更为下一个最高级别的列或参数的数量如果解除小于最高级别column或参数的数的列或参数的绑定时

（TargetValuePtr参数设置为空指针调用SQLBindCol或将ParameterValuePtr参数设置为空指针后调用SQLBindParameter时）SQL_DESC_COUNT不变如果追加的列或参数绑定大于包含数据的最高级别记录的数时驱动程序自动增加SQL_DESC_COUNT字段的值通过SQL_UNBIND选项调用SQLFreeStmt后解除所有列的绑定时ARD及IRD中的SQL_DESC_COUNT字段设置为0通过SQL_RESET_PARAMS选项调用SQLFreeStmt时APD及IPD中的SQL_DESC_COUNT字段设置为0

应用程序可以调用SQLSetDescField明确设置SQL_DESC_COUNT值如果SQL_DESC_COUNT值明显减少则有效删除所有大于新的SQL_DESC_COUNT的值ARD的SQL_DESC_COUNT字段值明确设置为0时除了绑定的书签列外解除所有缓冲区ARD的此字段的记录数不包含绑定的书签列解除书签列绑定的唯一方法是将SQL_DESC_DATA_PTR字段设置为空指针

SQL_DESC_ROWS_PROCESSED_PTR[Implementation descriptors]

IRD的此SQLULEN*头部字段指包含调用SQLFetch或SQLFetchScroll后获取的行数或调用SQLBulkOperations或SQLSetPos执行的批量操作影响的行数以及出错行数的缓冲区

IPD的此SQLUINTEGER*头部字段指包含已处理的参数集合及出错数量的缓冲区如果为空指针则不返回数量

SQL_DESC_ROWS_PROCESSED_PTR（对IRD字段）只有在调用SQLFetch或SQLFetchScroll后或（对IPD字段）调用SQLExecuteSQLExecDirect或SQLParamData并返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO后有效如果之前的函数未返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO则不定义缓冲区的内容直到返回SQL_NO_DATA为止缓冲区的值设置为0

可通过SQL_ATTR_ROWS_FETCHED_PTR属性调用SQLSetStmtAttr设置ARD的此字段APD可通过SQL_ATTR_PARAMS_PROCESSED_PTR属性调用进行设置

应用程序分配此字段指向的缓冲区其是驱动程序设置的延迟的输出缓冲区默认设置为空指针

记录字段

各描述符包含一个以上的由基于描述符类型定义的column数据或动态参数中的一个的字段组成的记录各记录为单个column或参数的完整的明细

SQL_DESC_AUTO_UNIQUE_VALUE[IRDs]（只读专用）

column为自动增加的column时此只读专用SQLINTEGER记录字段为SQL_TRUE反之为SQL_FALSE此字段为只读专用但如果是自动增加column则不需要必须为只读

SQL_DESC_BASE_COLUMN_NAME[IRDs]（只读专用）

此只读SQLCHAR*记录字段包含结果集合column的默认column名如果没有默认column名

则此字段包含空字符串

SQL_DESC_TABLE_NAME[IRDs]（只读专用）

此只读专用SQLCHAR*记录字段包含结果集合column的默认表名如果不能定义或使用默认表名则此字段包含空字符串

SQL_DESC_CASE_SENSITIVE[Implementation descriptors]（只读专用）

排序或比较列或参数时如果区分大小写则此只读SQLINTEGER记录字段为SQL_TRUE不区分大小写或不是字符的column时包含SQL_FALSE

SQL_DESC_CATALOG_NAME[IRDs]（只读专用）

此只读专用SQLCHAR*记录字段包含有column的默认表的目录column为表达式或视图的一部分时返回值取决于驱动程序数据源不支持目录或无法查看目录时此字段包含空字符串

SQL_DESC_CHAR_LENGTH_UNITS[IPD]

此SQLSMALLINT记录字段描述SQL_DESC_LENGTH的单位

(SQL_CLU_NONE/SQL_CLU_CHARACTERS/SQL_CLU_OCTETS)

SQL_DESC_CONCISE_TYPE[All]

此SQLSMALLINT头部字段对包含datetime和interva数据类型的所有数据类型指定简洁的数据类型

SQL_DESC_CONCISE_TYPESQL_DESC_TYPE和SQL_DESC_DATETIME_INTERVAL_CODE字段的值是互相依赖的如果在一个字段设置time则其他字段也要设置

SQL_DESC_CONCISE_TYPE可通过调用SQLBindColSQLBindParameter或SQLSetDescField进行设置SQL_DESC_TYPE可通过调用SQLSetDescField或SQLSetDescRec设置

如果将SQL_DESC_CONCISE_TYPE设置为除interval或datetime数据类型外的简洁的数据类型时SQL_DESC_TYPE字段将被设置为相同的值SQL_DESC_DATETIME_INTERVAL_CODE字段被设置为0

如果将SQL_DESC_CONCISE_TYPE设置为简洁的datetime或interval数据类型SQL_DESC_TYPE字段设置为详细的类型(SQL_DATETIME或SQL_INTERVAL)SQL_DESC_DATETIME_INTERVAL_CODE字段设置适当的子码

SQL_DESC_DATETIME_INTERVAL_CODE[All]

此SQLSMALLINT记录字段包含SQL_DESC_TYPE字段为SQL_DATETIME或SQL_INTERVAL时定义datetime或interval数据类型的子码SQL和C也均相同代码包含代替datetime类型的TYPE或C_TYPEinterval类型的INTERVAL或C_INTERVAL的CODE的数据类型名称

将应用程序描述符的SQL_DESC_TYPE和SQL_DESC_CONCISE_TYPE设置为SQL_C_DEFAULT描述符和命令语句柄不相关时不定义SQL_DESC_DATETIME_INTERVAL_CODE的内容

此字段可设置以下表格列出的datetime数据类型

Datetime types	DATETIME_INTERVAL_CODE
SQL_TYPE_DATE/ SQL_C_TYPE_DATE	SQL_CODE_DATE
SQL_TYPE_TIME/ SQL_C_TYPE_TIME	SQL_CODE_TIME
SQL_TYPE_TIME_WITH_TIMEZONE/ SQL_C_TYPE_TIME_WITH_TIMEZONE	SQL_CODE_TIME_WITH_TIMEZONE

Datetime types	DATETIME_INTERVAL_CODE
SQL_TYPE_TIMESTAMP/ SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE/ SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE	SQL_CODE_TIMESTAMP_WITH_TIMEZONE

此字段可设置以下表格列出的interval数据类型

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_DAY/ SQL_C_INTERVAL_DAY	SQL_CODE_DAY
SQL_INTERVAL_DAY_TO_HOUR/ SQL_C_INTERVAL_DAY_TO_HOUR	SQL_CODE_DAY_TO_HOUR
SQL_INTERVAL_DAY_TO_MINUTE/ SQL_C_INTERVAL_DAY_TO_MINUTE	SQL_CODE_DAY_TO_MINUTE
SQL_INTERVAL_DAY_TO_SECOND/ SQL_C_INTERVAL_DAY_TO_SECOND	SQL_CODE_DAY_TO_SECOND
SQL_INTERVAL_HOUR/ SQL_C_INTERVAL_HOUR	SQL_CODE_HOUR
SQL_INTERVAL_HOUR_TO_MINUTE/ SQL_C_INTERVAL_HOUR_TO_MINUTE	SQL_CODE_HOUR_TO_MINUTE

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_HOUR_TO_SECOND/ SQL_C_INTERVAL_HOUR_TO_SECOND	SQL_CODE_HOUR_TO_SECOND
SQL_INTERVAL_MINUTE/ SQL_C_INTERVAL_MINUTE	SQL_CODE_MINUTE
SQL_INTERVAL_MINUTE_TO_SECOND/ SQL_C_INTERVAL_MINUTE_TO_SECOND	SQL_CODE_MONUTE_TO_SECOND
SQL_INTERVAL_MONTH/ SQL_C_INTERVAL_MONTH	SQL_CODE_MONTH
SQL_INTERVAL_SECOND/ SQL_C_INTERVAL_SECOND	SQL_CODE_SECOND
SQL_INTERVAL_YEAR/ SQL_C_INTERVAL_YEAR	SQL_CODE_YEAR
SQL_INTERVAL_YEAR_TO_MONTH/ SQL_C_INTERVAL_YEAR_TO_MONTH	SQL_CODE_YEAR_TO_MONTH

SQL_DESC_DATETIME_INTERVAL_PRECISION[All]

此SQLINTEGER记录字段的SQL_DESC_TYPE字段为SQL_INTERVAL时包含interval leading precision将SQL_DESC_DATETIME_INTERVAL_CODE字段设置为interval数据类型时此字段设置默认interval leading precision

SQL_DESC_DISPLAY_SIZE[IRDs]（只读专用)

此只读SQLINTEGER记录字段包含为了展示列数据而所需的最大字符数量

SQL_DESC_FIXED_PREC_SCALE[Implementation descriptors]（只读专用）

如果此只读SQLSMALLINT记录字段的column为exact numeric column并拥有非0的scale与固定的precision则为SQL_TRUE反之设置为SQL_FALSE

SQL_DESC_INDICATOR_PTR[Application descriptors]

ARD的此SQLLEN*记录字段表示指示符变量该变量的column值为空时包含SQL_NULL_DATEAPD的指示符变量为了把动态参数定义为NULL而设置为SQL_NULL_DATA否则变量为0

ARD的SQL_DESC_INDICATOR_PTR字段为空指针时驱动程序防止返回column是否为空column为空SQL_DESC_INDICATOR_PTR为空指针时驱动程序调用SQLFetch或SQLFetchScroll后生成缓冲区时返回SQLSTATE 22002 (Indicator variable required but not supplied)如果SQLFetch或SQLFetchScroll的调用不返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO则不定义缓冲区的内容

SQL_DESC_INDICATOR_PTR字段决定是否设置SQL_DESC_OCTET_LENGTH_PTR指向的字段column的数据值为空时驱动程序把指示符变量设置为SQL_NULL_DATA字段不在此时设置SQL_DESC_OCTET_LENGTH_PTR表示的字段获取数据时如果不接收空值则将SQL_DESC_INDICATOR_PTR指向的缓冲区设置为0将SQL_DESC_OCTET_LENGTH_PTR指向的缓冲区设置为数据的长度

APD的SQL_DESC_INDICATOR_PTR字段为空指针时应用程序为了把参数定义为NULL无法使用此描述符记录

此字段为延迟的字段设置时不使用此字段但驱动程序表示NULL可能性(ARD的情况)或决定NULL可能性(APD的情况)时使用

SQL_DESC_LABEL[IRDs]（只读专用）

此只读SQLCHAR*记录字段包含列的标签或标志列没有标签时此变量包含列的名称未命

名列或无法使用标签时此变量包含空字符串

SQL_DESC_LENGTH[All]

此SQLULEN记录字段是字符串的最大长度或实际长度或字节单位的二进制数据其为固定长度数据类型的最大长度或可变长数据类型的实际长度此值总是排除字符串最后的空终止字符SQL_TYPE_DATESQL_TYPE_TIMESQL_TYPE_TIMESTAMP或SQL interval数据类型的值时此字段拥有datetime或interval值转换为字符串时的字符长度

此字段的值可能不同于ODBC 2.x定义的length值

SQL_DESC_LITERAL_PREFIX[IRDs]（只读专用）

此只读SQLCHAR*记录字段包含字符或驱动程序识别为后缀的字符此变量包含对无法适用字符后缀的数据类型的空字符串

SQL_DESC_LOCAL_TYPE_NAME[Implementation descriptors]（只读专用）

此只读SQLCHAR*记录字段对数据类型包含本地化的名称这可能与数据类型的正规名称不同没有本地化的名称时返回空字符串此字段是仅以显示为目的的字段

SQL_DESC_NAME[Implementation descriptor]

此SQLCHAR*记录字段在行描述符中包含字段的别名如果不使用column的别名则返回column名两种情况均在设置SQL_DESC_NAME字段时驱动程序将SQL_DESC_UNNAMED字段设置为SQL_NAMED如果没有column名或column别名驱动程序则返回SQL_DESC_NAME字段的空字符串并将SQL_DESC_UNNAMED字段设置为SQL_UNNAMED

应用程序可以通过将IPD的SQL_DESC_NAME字段设置为参数名或别名按照名称指定存储的procedure参数IRD的SQL_DESC_NAME字段为只读字段应用程序药设置此字段时返回SQLSTATE HY091 (Invalid descriptor field identifier)

IPD的情况驱动程序不支持命名的参数时不定义此字段驱动程序支持命名的参数并其可

说明参数时此字段返回参数名称

SQL_DESC_NULLABLE[Implementation descriptors]（只读专用）

IRD的情况此只读SQLSMALLINT记录字段的column可以有空值时为SQL_NULLABLE不能有空值时为SQL_NO_NULLS无法查看是否允许空值时为SQL_NULLABLE_UNKNOWN此字段是为结果集的column而存在

在IPD情况此字段的动态参数可以总是为空 应用程序无法设置因此总是设置为SQL_NULLABLE

SQL_DESC_NUM_PREC_RADIX[All]

SQL_DESC_TYPE字段为approximate numeric数据类型时此SQLINTEGER字段值为2因为SQL_DESC_PRECISION字段包含bit数SQL_DESC_TYPE字段为exact numeric数据类型时SQL_DESC_PRECISION字段包含小数点位数因此此字段的值为10此字段在不是数字型数据类型时设置为0

SQL_DESC_OCTET_LENGTH[All]

此SQLLEN记录字段包含字符串或二进制数据类型的字节单位长度固定长度的字符或二进制类型时为字节单位的实际长度可变长度的字符或二进制类型时为字节单位的最大长度此值对执行描述符不包含空终止字符对应用程序描述符包含空终止字符应用程序数据的情况此字段包含缓冲区大小APD的情况此字段只对输出或输入/输出参数定义

SQL_DESC_OCTET_LENGTH_PTR[Application descriptors]

此SQLLEN*记录字段指包含（参数描述符的情况）动态参数或（行描述符的情况）绑定的column值的字节单位的总长度的变量

APD的情况此值忽略字符串和二进制以外的所有参数如果此字段为SQL_NTS则动态参数应以NULL终止为了显示Data-at-execution参数为要绑定的参数应用程序执行时将包含

SQL_DATA_AT_EXEC或SQL_LEN_DATA_AT_EXEC Macro结果的变量设置于APD记录中的此字段这种字段有一个以上时可将SQL_DESC_DATA_PTR设置为识别参数的值助于判断应用程序要求的参数

ARD的OCTET_LENGTH_PTR字段为空指针时驱动程序不返回column的长度信息APD的SQL_DESC_OCTET_LENGTH_PTR为空指针时驱动程序认为字符串和二进制值以NULL终止（二进制值不应该以NULL终止但为了避免数据被截断应赋予长度）

填充此字段指向的缓冲区的SQLFetch或SQLFetchScroll不返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO时不定义缓冲区的内容此字段为延迟的字段此字段不会立即使用而在后续驱动程序表示或决定数据的octet长度时使用

SQL_DESC_PARAMETER_TYPE[IRDs]

此SQLSMALLINT记录字段对输入参数设置为SQL_PARAM_INPUT对输入/输出参数设置为SQL_PARAM_INPUT_OUTPUT对输出参数设置SQL_PARAM_OUTPUT对流输入/输出参数设置为SQL_PARAM_INPUT_OUTPUT_STREAM对流输出参数设置为SQL_PARAM_OUTPUT_STREAM其默认设置为SQL_PARAM_INPUT

SQL_DESC_PRECISION[All]

此SQLSMALLINT记录字段对exact numeric类型包含有效的整数数量对approximate numeic包含mantissa（二进制精度）bit数又对SQL_TYPE_TIMESQL_TYPE_TIMESTAMPSQL_INTERVAL_SECOND数据类型包含fractional秒部分的位数此字段不在其他所有数据类型定义

此字段值不等于ODBC 2.x定义的precision值

SQL_DESC_ROWVER[Implementation descriptors]（只读专用）

此SQLSMALLINT记录字段在更新行（例如SQL Serever的timestamp）时表示数据库是否自动修改column此记录字段的值为行版本column时是SQL_TRUE否则设置为SQL_FALSE

此column属性类似于为了决定是否自动更新列在SQL_ROWVER的IdentifierType调用SQLSpecialColumn

SQL_DESC_SCALE[All]

此SQLSMALLINT记录字段对decimal与numeric数据类型包含定义的小数点位数此字段不在其他所有数据类型中定义

此字段的值可能不同于ODBC 2.x定义的scale值

SQL_DESC_SCHEMA_NAME[IRDs]（只读专用）

此只读SQLCHAR*记录字段包含包含column的默认表的SCHEMA名称列为表达式或视图的一部分时返回值取决于驱动程序数据源不支持SCHEMA或无法定义SCHEMA名称时此变量包含空字符串

SQL_DESC_SEARCHABLE[IRDs]（只读专用）

此只读SQLSMAINT记录字段设置以下值中的一个值

• Column不能用于WHERE子句时设置为SQL_PRED_NONE（与ODBC 2.X的SQL_UNSEARCHABLE相同）

• Column只能与WHERE子句的LIKE条件一起使用时设置为SQL_PRED_CHAR

• Column在WHERE子句中可以与除LIKE条件外的其他所有比较运算符一起使用时设置为SQL_PRED_BASIC（与ODBC 2.x的SQL_EXCEPT_LIKE值相同）

• Column能与WHERE子句的任何比较运算符一起使用时设置为SQL_PRED_SEARCHABLE

SQL_DESC_TABLE_NAME[IRDs]（只读专用）

此只读SQLCHAR*记录字段包含此column所属的默认表名column为表达式或视图的一部分时返回值取决于驱动程序

SQL_DESC_TYPE[All]

此SQLSMALLINT记录字段对除了datetime和interval数据类型外的所有数据类型包含缩略的SQL或C数据类型对datetime和interval数据类型此字段定义SQL_DATETIME或SQL_INTERVAL

此字段包含SQL_DATETIME或SQL_INTERVAL时SQL_DESC_DATETIME_INTERVAL_CODE字段必须包含缩略类型的恰当的Sub code对datetime数据类型SQL_DESC_TYPE包含SQL_DATETIMESQL_DESC_DATETIME_INTERVAL_CODE字段包含指定datetime数据类型的Sub code对interval数据类型SQL_DESC_TYPE包含SQL_INTERVALSQL_DESC_DATETIME_INTERVAL_CODE字段包含指定interval数据类型的Sub code

SQL_DESC_TYPE和SQL_DESC_CONCISE_TYPE字段的值是相互依赖的设置字段中的一个时也要设置其他字段SQL_DESC_TYPE可以通过调用SQLSetDescField或SQLSetDescRec设置SQL_DESC_CONCISE_TYPE可以通过调用SQLBindColSQLBindParameter或SQLSetDescField设置

SQL_DESC_TYPE设置为非interval或datetime数据类型的缩略的数据类型时SQL_DESC_CONCISE_TYPE字段设置为相同的值SQL_DESC_DATETIME_INTERVAL_CODE字段设置为0

SQL_DESC_TYPE设置为datetime或interval数据类型的长的数据类型（SQL_DATETIME或SQL_INTERVAL）SQL_DESC_DATETIME_INTERVAL_CODE字段设置为恰当的sub code时SQL_DESC_CONCISE_TYPE字段设置为对应于缩略的数据类型的值将SQL_DESC_TYPE设置为缩略的datetime或interval类型中的一个时返回SQLSTATE HY021 (Inconsistent descriptor information)

通过调用SQLBindColSQLBindParameter或SQLSetDescField设置SQL_DESC_TYPE时设置为如下表的默认值不定义相同记录的剩余的字段值

SQL_DESC_TYPE 值	以下字段的默认设置
SQL_CHAR, SQL_VARCHAR, SQL_C_CHAR, SQL_C_VARCHAR	SQL_DESC_LENGTH设置为1 SQL_DESC_PRECISION设置为0
SQL_DATETIME	SQL_DESC_DATETIME_INTERVAL_CODE设置为SQL_CODE_DATE或SQL_CODE_TIME时SQL_DESC_PRECISION设置为0设置为SQL_DESC_TIMESTAMP时SQL_DESC_PRECISION设置为6
SQL_DECIMAL, SQL_NUMERIC, SQL_C_NUMERIC	SQL_DESC_SCALE设置为0SQL_DESC_PRECISION设置为各数据类型执行定义的精密度
SQL_FLOAT, SQL_C_FLOAT	SQL_DESC_PRECISION设置为SQL_FLOAT执行定义的默认精密度
SQL_INTERVAL	SQL_DESC_DATETIME_INTERVAL_CODE设置为interval数据类型时SQL_DESC_DATETIME_INTERVAL_PRECISION设置为2（默认interval leading precision）interval有秒部分时SQL_DESC_PRECISION设置为6（默认interval seconds precision）

应用程序调用SQLSetDescField而不是SQLSetDescRec设置描述符的字段时应用程序务必先定义数据类型此时默认设置上述表中说明的其他字段如果不允许默认设置任何值应用程序可通过调用SQLSetDescField或SQLSetDescRec明确进行设置

SQL_DESC_TYPE_NAME[Implementation descriptors]（只读专用）

此只读SQLCHAR*记录字段包含依赖于数据源的类型名称（CHARVARCHAR等）如果数据类型名称为unknown时变量包含空字符串

SQL_DESC_UNNAMED[Implementation descriptors]

设置SQL_DESC_NAME字段时行描述符中的此SQLSMALLINT记录字段被驱动程序设置为SQL_NAME或SQL_UNNAMED中的一个SQL_DESC_NAME字段包含column的别名或不使用column的别名时驱动程序将SQL_DESC_UNNAMED字段设置为SQL_NAMED当应用程序将IPD的SQL_DESC_NAME字段设置为参数名称或别名时驱动程序将IPD的SQL_DESC_UNNAMED字段设置为SQL_NAMED没有column名称或别名时驱动程序将SQL_DESC_UNNAMED字段设置为SQL_UNNAMED

应用程序可以将IPD的SQL_DESC_UNNAMED字段设置为SQL_UNNAMED应用程序试图把IPD的SQL_DESC_UNANMED字段设置为SQL_NAMED时驱动程序返回SQLSTATE HY091 (Invalid descriptor field identifier)如果应用程序试图设置只读专用IRD的SQL_DESC_UNNAMED字段时驱动程序返回SQLSTATE HY091(Invalid descriptor field idenfitier)

SQL_DESC_UNSIGNED[Implemetation descriptors]（只读专用）

此只读SQLSMALLINT记录字段的column类型为unsigned或non-numeric时设置为SQL_TUREcolumn形式为signed时设置为SQL_FALSE

SQL_DESC_UPDATABLE[IRDs]（只读专用）

此只读SQLSMALLINT记录字段设置为以下值中的一个值

- 结果集合column为只读时设置为SQL_ATTR_READ_ONLY
- 结果集合column为可读可写时设置为SQL_ATTR_WRITE
- 无法查看是否更新结果集合column时设置为SQL_ATTR_READWRITE_UNKNOWN

SQL_DESC_UPDATABLE说明结果集合中的column（非默认表的字column）的更新与否

以此结果集合column为基础的默认表的column的更新与否可能与此column中的值不同

column的更新与否基于数据类型用户权限以及结果集合的自定义无法确定column的更新与否时返回SQL_ATTR_READWITE_UNKNOWN

SQL_DESC_CHAR_LENGTH_UNITS[Alls]

此SQLSMALLINT记录字段表示SQL类型为SQL_CHARSQL_VARCHAR或SQL_LONGVARCHAR的column的长度单位

- SQL_CLU_CHARACTERS：长度单位为CHARACTER例如编码方式为UHC (Unified Hangul Code)的称为“字符串”的数据的长度是3
- SQL_CLU_OCTETS：长度单位为OCTETS例如编码方式为UHC的称为“字符串”的数据的长度是6
- SQL_CLU_NONE：未定义长度单位除以上列举的SQL类型外的SQL类型时返回的值

查看一致性

应用程序每次把数据传给ARDAPD或IPD的SQL_DESC_DATA_PTR字段时驱动程序自动查看一致性部分字段与其他字段不一致时SQLSetDescField返回SQLSTATE HY021(inconsistent descriptor information)

详细内容参考**SQLSetDescRec**的**查看一致性**

SQLSetDescRec

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLSetDescRec函数设置数据类型或影响绑定于column或参数数据的缓冲区的多个描述符字段

语句

```
SQLRETURN SQLSetDescRec(
    SQLHDESC      DescriptorHandle,
    SQLSMALLINT   RecNumber,
    SQLSMALLINT   Type,
    SQLSMALLINT   SubType,
    SQLLEN        Length,
    SQLSMALLINT   Precision,
    SQLSMALLINT   Scale,
    SQLPOINTER    DataPtr,
    SQLLEN *      StringLengthPtr,
    SQLLEN *      IndicatorPtr);
```

参数

DescriptorHandle

【输入】描述符句柄无法设置IRD句柄

RecNumber

【输入】指包含要设置的字段的描述符记录描述符记录从0开始编号0号记录为书签记录此参数应大于或等于0RecNumber大于SQL_DESC_COUNT时SQL_DESC_COUNT转换为RecNumber值

Type

【输入】设置描述符记录的SQL_DESC_TYPE字段的值

SubType

【输入】对SQL_DATETIME或SQL_INTERVAL的数据类型设置SQL_DESC_DATETIME_INTERVAL_CODE字段的值

Length

【输入】设置描述符记录的SQL_DESC_OCTET_LENGTH字段的值

Precision

【输入】设置描述符记录的SQL_DESC_PRECISION字段的值

Scale

【输入】设置描述符记录的SQL_DESC_SCALE字段的值

DataPtr

[延迟的输入或输出] 设置描述符记录的SQL_DESC_DATA_PTR字段的值DataPtr可设置为空指针

为了将SQL_DESC_DATA_PTR设置为空指针DataPtr参数可以为空指针DescriptorHandle参数中的句柄与ARD有关时其解除column绑定

StringLengthPtr

[延迟的输入或输出] 设置描述符记录的SQL_DESC_OCTET_LENGTH_PTR字段的值为了把SQL_DESC_OCTET_LENGTH_PTR字段设置为空指针StringLengthPtr可以设置为空指针

IndicatorPtr

[延迟的输入或输出] 设置描述符记录的SQL_DESC_INDICATOR_PTR字段的值为了把SQL_DESC_INDICATOR_PTR设置为空指针IndicatorPtr可以为空指针

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	具体的驱动程序信息（函数返回SQL_SUCCESS_WITH_INFO）

SQLSTATE	报错	说明
07009	Invalid descriptor index	RecNumber参数设置为0DescriptorHandle指向IPD句柄 RecNumber参数大于数据源支持的column或参数的最大数DescriptorHandle参数为APDIPD或ARD RecNumber参数为0DescriptorHandle参数指向内部已分配的APD（对应用程序明确分配的描述符不报此错误因为直到执行无法知道分配的应用程序描述符为APD或ARD）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
HY000	General error	无特定SQLSTATE的错误*MessageText缓冲区的SQLGetDiagTec返回的错误信息说明错误及原因
HY001	Memory allocation error	无法分配驱动程序执行或完成函数所需的内存
HY010	Function sequence error	对DescriptorHandle相关的StatementHandle执行了异步执行函数执行SQLSetDescRec时异步执行的函数仍在执行中
HY013	Memory management error	由于内存不足无法访问内存对象因此无法执行函数
HY016	Cannot modify an implementation row descriptor	DescriptorHandle参数与IRD有关

SQLSTATE	报错	说明
HY021	Inconsistent descriptor information	与Type字段或描述符的SQL_DESC_TYPE字段相关的其他字段无效或不一致
HY090	Invalid string or buffer length	驱动程序为ODBC 2.x 驱动程序描述符为ARD ColumnNumber参数为0BufferLength参数指定的值不是4时
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed	挂起状态相关的详细内容参考**SQLEndTran**
HYT01	Connection timeout expired	数据源响应请求之前超时连接超时周期可通过SQLSetConnectAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	驱动程序不支持此函数

说明

应用程序可通过调用SQLSetDescRec设置单个column或参数的以下字段

- SQL_DESC_TYPE
- (SQL_DATETIME或SQL_INTERVAL类型记录的情况)

  SQL_DESC_DATETIME_INTERVAL_CODE

- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_DATA_PTR
- SQL_DESC_OCTET_LENGTH_PTR
- SQL_DESC_INDICATOR_PTR

> Note:
>
> 调用SQLSetDescRec失败时不定义RecNumber参数识别的描述符记录的内容

绑定列或参数时SQLSetDescRec不调用SQLBindCol或SQLBindParameter也未多次调用SQLSetDescField的情况下变更影响绑定的多个字段SQLSetDescRec可以设置与当前命令语无关的描述符的字段SQLBindParameter可以设置多于SQLSetDescRec的字段可通过一次调用设置APD和IPD的字段不要求描述符句柄

> Note:
>
> 为了设置书签以0的RecNumber参数调用SQLSetDescField之前必须设置命令语属性SQL_ATTR_USE_BOOKMARKS这不是强制性的但是推荐事项

查看一致性

每次应用程序设置ARDAPD或IPD的SQL_DESC_DATA_PTR字段时驱动程序均自动查看一致性存在与其他字段不一致的字段时SQLSetDescRec返回SQLSTATE HY021(Inconsistent descriptor information)

每次应用程序设置ARDAPD或IPD的SQL_DESC_DATA_PTR字段时驱动程序查看SQL_DESC_TYPE的字段值查看其值是否对SQL_DESC_TYPE字段有效或一致调用SQLBindParameter或SQLBindCol或APDARD或IPD调用SQLSetDescRec时查看一致性一致性查看包含以下描述符字段

- SQL_DESC_TYPE字段应为有效的ODBC CSQL Type或驱动程序定义的SQL type中的一个SQL_DESC_CONCISE_TYPE字段应为有效的ODBC CSQL Type或驱动程序定义的SQL type中的一个包含简洁的datetime和interval类型
- SQL_DESC_TYPE记录字段为SQL_DATETIME或SQL_INTERVAL时SQL_DESC_DATETIME_INTERVAL_CODE字段应为有效的datetime或interval code中的一个(参考**SQLSetDescField**的SQL_DESC_DATETIME_INTERVAL_CODE字段的说明）
- SQL_DESC_TYPE字段指向NUMERIC类型时查看SQL_DESC_PRECISION和SQL_DESC_SCALE字段是否为有效的值
- SQL_DESC_CONCISE_TYPE字段为time或timestamp数据类型并为适用second要素或time要素的interval数据类型时查看SQL_DESC_PRECISION字段是否为有效的second precision
- SQL_DESC_CONCISE_TYPE为interval数据类型时查看SQL_DESC_DATETIME_INTERVAL_PRECISION字段是否为有效的interval先行precision值

通常不设置IPD的SQL_DESC_DATA_PTR字段应用程序可以强制执行IPD字段的一致性查看在IRD无法执行一致性查看IPD的SQL_DESC_DATA_PTR字段值未实际存储无法通过SQLGetDescField或SQLGetDescRec检索设置为强制执行一致性查看

SQLSetEnvAttr

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLSetEnvAttr 设置环境管理属性

语句

```
SQLRETURN SQLSetEnvAttr(
    SQLHENV      EnvironmentHandle,
    SQLINTEGER   Attribute,
    SQLPOINTER   ValuePtr,
    SQLINTEGER   StringLength);
```

参数

EnvironmentHandle

【输入】环境句柄

Attribute

【输入】在说明部分列出

ValuePtr

【输入】联动于Attribute的值的指针根据Attribute 值ValuePtr可以为32位整数值或指向空终止字符

StringLength

【输入】ValuePtr指向字符串或二进制缓冲区时此参数应为*ValuePtr的长度字符串数据的情况此参数应包含字符串的字节数

ValuePtr 为整数时忽略StringLength

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	具体的驱动程序信息（函数返回SQL_SUCCESS_WITH_INFO）
01S02	Option value changed	驱动程序不支持ValuePtr指定的值 替代为相近值（函数返回SQL_SUCCESS_WITH_INFO）
HY000	General error	无特定SQLSTATE的错误*MessageText缓冲区的SQLGetDiagRec返回详细的错误信息和原因
HY001	Memory allocation error	无法分配驱动程序执行或完成函数所需的内存

SQLSTATE	报错	说明
HY009	Invalid use of null pointer	属性参数识别需要字符串值的环境属性ValuePtr参数为null指针
HY010	Function sequence error	连接句柄分配于EnvironmentHandle SQL_ATTR_ODBC_VERSION未设置在SQLSetEnvAttrAttribute与SQL_ATTR_ODBC_VERSION不同 如果使用SQLAllocHandleStd则不需要指定SQL_ATTR_ODBC_VERSION
HY013	Memory management error	由于内存不足无法访问内存对象而未执行函数
HY024	Invalid attribute value	考虑到指定的Attribute值 ValuePtr的值无效
HY090	Invalid string or buffer length	StringLength参数小于0（不是SQL_NTS）
HY092	Invalid attribute/option identifier	输入的Attribute在驱动程序支持的ODBC版本中无效
HY117	Connection is suspended due to unknown trasaction state. Only disconnect and read-only functions ard allowed	挂起状态相关的更多内容参考**SQLEndTran**函数

SQLSTATE	报错	说明
HYC00	Optional feature not implemented	指定的Attribute参数在驱动程序支持的ODBC版本的ODBC环境属性中有效但驱动程序不支持 Attribute参数为SQL_ATTR_OUTPUT_NTSValuePtr为SQL_FASE

说明

应用程序只有在环境句柄中没有分配的连接句柄时才可调用SQLSetEnvAttr应用程序设置的环境属性有效至对环境句柄调用SQLFreeHandle为止建议只分配使用一个环境句柄

通过ValuePtr设置的信息的格式取决于指定的AttributeSQLSetEnvAttr在两个不同的格式（NULL终止字符串或32bit整数值）中接受一种属性信息属性部分说明了各格式

不存在各驱动程序的环境属性

无法通过调用SQLSetEnvAttr设置连接属性尝试设置则返回SQLSTATE HY092 (Invalid attribute/option identifier)

Attribute	ValuPtr 内容
SQL_ATTR_CONNECTION_POOLING (ODBC 3.8)	驱动程序不支持
SQL_ATTR_CP_MATCH (ODBC 3.0)	驱动程序不支持

SQL_ATTR_ODBC_VERSION
(ODBC 3.0)
	表示特定功能以ODBC 2.x还是以ODBC 3.x执行的32位整数
	
	以下值用于设置属性
	
	应用程序在调用拥有SQLHENV参数的函数之前应设置此环境属性值否则返回SQLSTATE HY010(Function sequence error)对这些环境标志是否存在额外的行为取决于各个驱动程序
	
	• SQL_OV_ODBC3_80: 驱动程序管理器和驱动程序具有以下ODBC 3.8行为
	  ○ 驱动程序对DATETIMETIMESTAMP预计并返回ODBC 3.x代码值
	  ○ 驱动程序调用SQLErrorSQLGetDiagField或SQLGetDiagRec时返回ODBC 3.x SQLSTATE代码
	  ○ SQLTables的CatalogName参数允许模式检索
	• SQL_OV_ODBC3: 驱动程序管理器和驱动程序具有以下ODBC 3.x行为
	  ○ 驱动程序对DATETIMETIMESTAMP预计并返回ODBC 3.x代码值
	  ○ 驱动程序调用SQLErrorSQLGetDiagField或SQLGetDiagRec时返回ODBC 3.x SQLSTATE代码
	  ○ SQLTables的CatalogName 参数允许模式检索
	  ○ 驱动程序管理器不支持C数据类型的扩展性

Attribute	ValuPtr 内容

	• SQL_OV_ODBC2: 驱动程序管理器和驱动程序具有以下ODBC 2.x行为其非常有利于在ODBC 3.x 驱动程序下运行ODBC 2.x应用程序
	  ○ 驱动程序对DATETIMETIMESTAMP预计并返回ODBC 2.x代码值
	  ○ 驱动程序调用SQLErrorSQLGetDiagField SQLGetDiagRec时返回ODBC 2.x SQLSTATE代码
	  ○ SQLTables的CatalogName参数不允许模式检索
	  ○ 驱动程序管理器不支持C数据类型的扩展性

SQL_ATTR_OUTPUT_NTS (ODBC 3.0)	驱动程序不支持

SQLSetParam

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 1.0函数SQLSetParam在ODBC 2.0中被替代为SQLBindParameter

详细内容参考**SQLBindParameter**

SQLSetPos

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLSetPos设置行集合中的游标位置并允许应用程序更新行集合的数据或更新/删除结果集中的数据

语句

```
SQLRETURN SQLSetPos(
    SQLHSTMT        StatementHandle,
    SQLSETPOSIROW   RowNumber,
    SQLUSMALLINT    Operation,
    SQLUSMALLINT    LockType);
```

参数

StatementHandle

【输入】命令语句柄

RowNumber

【输入】执行指定为Operation参数的运算的行集合中行的位置RowNumgber为0时运算适用于行集合中的所有行

详细内容参考**说明**部分

Operation

【输入】要执行的运算是SQL_POSITIONSQL_REFRESHSQL_UPDATESQL_DELETE

ODBC 3.x中不再使用Operation参数的SQL_ADD值ODBC 2.x驱动程序为了与低版本的兼容性需支持SQL_ADD此功能被替代为使用SQL_ADD运算调用SQLBulkOperaions以ODBC 2.x驱动程序执行ODBC 3.x应用程序时驱动程序管理器将使用SQL_ADD运算的SQLBulkOperations调用映射到使用SQL_ADD运算的SQLSetPos调用

详细内容参考**说明**部分

LockType

【输入】定义执行Operation参数中定义的运算后锁定行的方法

SQL_LOCK_NO_CHANGE, SQL_LOCK_EXCLUSIVE, SQL_LOCK_UNLOCK

详细内容参考**说明**部分

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	各个驱动程序的信息（函数返回SQL_SUCCSS_WITH_INFO）
01001	Cursor operation conflict	Operation参数为SQL_DELETE或SQL_UPDATE删除或更新一个以上的行或未删除或更新任何行 Operation参数为SQL_DELETE或SQL_UPDATE由于同时执行而操作失败（函数返回SQL_SUCCESS_WITH_INFO）
01004	String data, right truncation	Operation参数为SQL_REFRESH对数据类型为SQL_C_CHAR或SQL_C_BINARY类型返回的非字符换或二进制数据的空白的字符或非null的二进制数据被截断
01S01	Error in row	RowNumber参数为0执行Operation参数定义的运算的过程中一条以上的行数据发生错误 （多种运算中非所有的一条以上的行数据发生错误时返回SQL_SUCCESS_WITH_INFO但一条行运算发生错误时返回SQL_ERROR） （驱动程序为ODBC 2.x并不使用游标库时在SQLExtendedFetch之后调用SQLSetPos时返回此SQLSTATE）

SQLSTATE	报错	说明
01S07	Fractional truncation	Operation参数为SQL_REFRESH应用程序缓冲区的数据类型不是SQL_C_CHAR或SQL_C_BINARY向应用程序缓冲区返回的一个以上的列被截断numeric数据类型时截断小数点部分timetimestamp以及包含time的interval数据类型时截断time的fractional部分
07006	Restricted data type attribute violation	结果集的列数据值调用SQLBindCol 时不能转换为TargetType 定义的数据类型
07009	Invalid descriptor index	Operation参数为SQL_REFRESH 或SQL_UPDATE绑定了比结果集的列数量更多的列
21S02	Degree of derived table does not match column list	Operation参数为SQL_UPDATE解除所有列的绑定或由于只读或绑定的长度/指示符缓冲区为SQL_COLUMN_IGNORE因此没有可更新的列
22001	String data, right truncation	Operation 参数为SQL_UPDATE由于向column分配字符或二进制值（对字符）而非空字符（对二进制）非NULL的字符或二进制被截断
22003	Numeric value out of range	Operation 参数为SQL_UPDATE由于向column分配numeirc值所有数字被截断 Operation 参数为SQL_REFRESH由于返回的一个以上绑定列的numeric值丢失了有效数字

SQLSTATE	报错	说明
22007	Invalid datetime format	Operation 参数为SQL_UPDATE由于向结果集的列分配date 或timestamp值超出了年月或日字段的范围 Operation 参数为SQL_REFRESH由于对一个以上绑定列返回date或timestamp超出了年月或日字段的范围
22008	Date/time field overflow	Operation参数为SQL_UPDATE由于对向结果集的column发送的数据执行了datetime运算使datetime字段（年月日时分秒）超出允许范围或违反公历年规则 Operation参数为SQL_REFRESS由于对结果集中检索到的数据执行了datetime运算使datetime字段（年月日时分秒）超出允许范围或违反公历年的规则

SQLSTATE	报错	说明
22015	Interval Field overflow	Operation参数为SQL_UPDATE由于将exact numeric 或interval C值分配为interval SQL数据类型导致丢失有效数字 Operation参数为SQL_UPDATE由于分配到interval SQL类型the interval SQL type中没有表示为C type的值 Operaion参数为SQL_REFRESH由于将exact numeric或interval SQL type分配到interval C 类型导致丢失先行字段的有效数字 Operation参数为SQL_REFRESH由于分配到interval C typeinterval C type中没有表示为SQL type的值
22018	Invalid character value for cast specification	Operation参数为SQL_REFRESHC 类型为exact或approximate numericdatetime或interval数据类型列的SQL类型为字符数据类型列的值对绑定的C类型无效 Operation参数为SQL_UPDATESQL类型为exact或approximate numericdatetime 或interval 数据类型C column的类型为SQL_C_CHAR列的值对绑定的SQL类型无效
23000	Intergrity constraint violation	Operation参数为SQL_DELETE或SQL_UPDATE违反了intergrity约束事项

SQLSTATE	报错	说明
24000	Invalid cursor state	StatementHandle为执行状态但结果与StatementHandle无关 StatementHandle中已打开游标但未调用SQLFetch或SQLFetchScroll StatementHandle中已打开游标也调用了SQLFetch或SQLFetchScroll但游标位于结果集的开始之前或结束之后 Operation参数为SQL_DELETESQL_REFRESH 或SQL_UPDATE
40001	Serialization failure	由于与其他事务的资源发生死锁此事务被回滚
40003	Statement completion unknown	函数执行过程中连接失败无法查看事务状态
42000	Syntax error or access violation	驱动程序无法锁定Operation参数请求的运算所需的row 驱动程序无法锁定LockType参数请求的row
44000	WITH CHECK OPTION violation	Operation参数为SQL_UPDATE为了受到更新影响的一个以上的row不存在于视图表更新了通过WITH_CHECK OPTION生成的视图表或从视图表衍生的表

SQLSTATE	报错	说明
HY000	General error	无特定SQLSTATE的报错*MessageText缓冲区的SQLGetDiagRec返回的错误信息说明错误及原因
HY001	Memory allocation error	驱动程序无法分配函数的执行或完成所需的内存
HY008	Operation canceled	可异步处理StatementHandle调用并完成执行此函数之前StatementHandle调用了SQLCancel或SQLCancelHandle此函数再次被StatementHandle调用 执行函数后完成执行之前从多线程应用程序的其他线程在StatementHandle调用了SQLCancel 或SQLCancelHandle

SQLSTATE	报错	说明
HY010	Function sequence error	执行SQLSetPos时StatementHandle调用了异步执行的函数调用SQLSetPos时该异步执行的函数还在执行中 StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults并返回了SQL_PARAM_DATA_AVAILABLE回收所有流式参数的数据之前调用了此函数 StatementHandle调用了异步执行函数调用此函数时该异步执行函数并仍在执行中 StatementHandle调用SQLExecuteSQLExecDirectSQLBulkOperation并返回了SQL_NEED_DATA传送所有数据之前调用了此函数 驱动程序为ODBC 2.x调用SQFetch 之后对StatementHandle调用了SQLSetPos
HY011	Attribute cannot be set now	驱动程序版本为ODBC 2.x在设置了SQL_ATTR_ROW_STATUS_PTR命令属性的情况下调用SQLFetch, SQLFetchScroll 或SQLExtendedFetch 之前调用了SQLSetPos
HY013	Memory management error	由于内存不足无法访问内存对象而函数执行失败

SQLSTATE	报错	说明
HY090	Invalid string or buffer length	Operation 参数为 SQL_UPDATE
HY092	Invalid attribute identifier	Operation 参数指定的值无效 LockType 参数指定的值无效 Operation参数为SQL_UPDATE 或 SQL_DELETESQL_ATTR_CONCURRENCY命令属性为 SQL_ATTR_CONCUR_READ_ONLY
HY107	Row value out of range	RowNumber参数指定的值大于行集合里的行数
HY109	Invalid cursor position	与StatementHandle相关的游标设置为forward-only因此无法位于行集合中参考SQLSetStmtAttr 的 SQL_ATTR_CURSOR_TYPE 属性说明 Operation参数为SQL_UPDATESQL_DELETE或 SQL_REFRESH 识别为RowNumber参数的行被删除或回收 RowNumber参数为0Operation参数为SQL_POSITION 调用SQLBulkOperaions后调用SQLFetchScroll或 SQLFetch之前调用了SQLSetPos

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran**函数
HYC00	Optional feature not implemented	驱动程序或数据源不支持Operation参数或LockType参数请求的运算
HYT00	Timeout expired	数据源返回结果集之前语句执行超时其限制时间可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置
HYT01	Connection timeout expired	数据源响应请求之前超时其限制时间可通过SQLSetConnectAttr的SQL_ATTR_CONNECTION_TIMEOUT设置
IM001	Driver does not support this function	与StatementHandle相关的驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时无法使用轮询

SQLSTATE	报错	说明
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄的之前的函数调用返回SQL_STILL_EXECUTING提醒模式生效时对句柄的后处理和结束操作必须调用SQLCompleteAsync

说明

RowNumber 参数

RowNumber参数定义Operation参数指定的运算执行影响的行集合中的行数RowNumber为0时适用于行集合中的所有行RowNumber必须为从0到行集合中的行数量

C语言的数组是从0开始但RowNumber参数从1开始例如更新行集合的第5条行时应用程序修改数组索引4的行集合缓冲区但RowNumber指定为5

所有运算将游标位于RowNumber指定的行以下运算要求游标位置

- positioned更新或删除命令语
- 调用SQLGetData
- 通过SQL_DELETESQL_REFRESHSQL_UPDATE选项调用SQLSetPos

Operation 参数

Operation参数支持以下运算为了查看数据源支持的选项应用程序根据游标类型以

SQL_DYNAMIC_CURSOR_ATTRIBUTES1SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1SQL_KEYSET_CURSOR_ATTRIBUTES1或SQL_STATIC_CURSOR_ATTRIBUTES1信息形式调用SQLGetInfo

Operation 参数	运算
SQL_POSITION	驱动程序将游标定位于RowNumber指定的行 SQL_ATTR_ROW_OPERATION_PTR属性所指的行状态数组内容被 SQL_POSITION忽略
SQL_REFRESH	驱动程序不支持
SQL_UPDATE	驱动程序不支持
SQL_DELETE	驱动程序不支持

LockType 参数

LockType参数提供应用程序可控制并发性的方法大部分情况下提供并发性级别和事务的数据源仅支持LockType参数的SQL_BLOCK_NO_CHANGE值

LockType参数定义执行SQLSetPos后的行锁状态如果驱动程序无法为了执行请求的操作或满足LockType参数而锁定行时返回SQL_ERROR或SQLSTATE 42000 (Syntax error or access violation)

即使在一个命令语中定义LockType参数也会适用于以相同权限连接的所有命令连接的一个命令语获取的锁可以被相同连接中的其他命令语解锁

通过SQLSetPos锁定的行有效至对LockType参数设置为SQL_LOCK_UNLOCK的行调用SQLSetPos或应用程序通过SQL_CLOSE选项调用SQLFreeHandle或对命令调用SQLFreeStmt驱动程序支持事务时通过SQLSetPos锁定的行（像SQLGetInfo返回的SQL_CURSOR_COMMIT_BEHAVIOR或SQL_CURSOR_ROLLBACK_BEHAVIOR信息类型显示的内容提交或回滚事务时游标关闭的情况）

在应用程序调用SQLEndTran提交或回滚事务时被解锁

LockType参数支持以下锁类型为了查看数据源支持的锁应用程序以SQL_DYNAMIC_CURSOR_ATTRIBUTES1SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1SQL_KEYSET_CURSOR_ATTRIBUTES1或SQL_STATIC_CURSOR_ATTRIBUTES1信息类型调用SQLGetInfo

LockType 参数	锁类型
SQL_LOCK_NO_CHANGE	驱动程序或数据源保证与调用SQLSetPos之前相同的行锁或解锁为了可使用当前并发性喝事务隔离级别请求的任何级别此LockType值不允许定义row级别的锁
SQL_LOCK_EXCLUSIVE	驱动程序或数据源排他性的锁定行其他连接或命令不能用于获取该row的锁
SQL_LOCK_UNLOCK	驱动程序或数据源解锁

SQLSetPos执行更新和删除操作时应用程序使用以下LockType参数

- 为了保证回收行后不被变更应用程序使用operation集合以SQL_REFRESH调用SQLSetPos以SQL_LOCK_EXCLUSIVE调用LockType
- 如果应用程序将LockType设置为SQL_LOCK_NO_CHANGE则驱动程序仅在应用程序对SQL_ATTR_CONCURRENCY命令属性定义SQL_CONCUR_LOCK时保证成功执行更新或删除操作
- 如果应用程序对SQL_ATTR_CONCURRENCY命令定义SQL_CONCUR_ROWVER或SQL_CONCUR_VALUES则驱动程序在应用程序回收row后变更此row时对比row的版本或值后拒绝操作
- 如果应用程序对SQL_ATTR_CONCURRENCY命令属性定义SQL_CONCUR_READ_ONLY则驱动

程序拒绝执行任何更新或删除操作

相关SQL_ATTR_CONCURRENCY属性的详细内容参考**SQLSetStmtAttr**

状态及运算数组

调用SQLSetPos时使用以下状态及运算数组

- 行状态数组包含行集合中的各个行数据的状态值驱动程序调用SQLSetchSQLFetchScrollSQLBulkOperations或SQLSetPos后设置此数组中的状态值SQL_ATTR_ROW_STATUS_PTR命令属性指定此数组
- 行操作数组包含表示是否忽略或执行批量操作的SQLSetPos调用的行集合中的各个行值数组的各个元素设置为SQL_ROW_PROCEED或SQL_ROW_IGNORE中的一个SQL_ATTR_ROW_OPERATION_PTR命令属性指定此数组

状态及运算数组中的元素数量必须（与SQL_ATTR_ROW_ARRAY_SIZE命令属性定义的相同）与行集合中的行数相同

行状态数组的详细内容参考**SQLFetch** 行操作数组相关的详细内容参考**批量操作中忽略行**

SQLSetPos用法

应用程序使用SQLSetPos前需执行以下步骤

1. 应用程序使用operation集合以SQL_UPDATE调用SQLSetPos时为了指定列的数据类型并绑定列数据和长度的缓冲区对各列调用SQLBindCol(或SQLSetDescRec)
2. 应用程序使用Operation集合以SQL_DELETE或SQL_UPDATE调用SQLSetPos时调用SQLColAttribute更新删除或更新的列

3. 调用SQLExecDirectSQLExecute或目录函数创建结果集
4. 调用SQLFetch或SQLFetchScroll回收数据

通过SQLSetPos删除数据

为了使用SQLSetPos删除数据应用程序使用RowNumber参数以要删除的行编号调用SQLSetPos并以SQL_DELETE调用operation

删除数据后驱动程序通过SQL_ROW_DELETED(或SQL_ROW_ERROR)变更该行的执行行状态数组值

通过SQLSetPos更新数据

应用程序通过绑定的缓冲区或一次以上的SQLPutData调用向列传送数据通过SQLPutData接收数据的列就是data-at-execution column 通常用于向SQL_LONGVARBINARY与SQL_LONGVARCHAR column传送数据可与其他列混合使用

应用程序中使用SQLSetPos更新数据

1. 通过数据和SQLBindCol向绑定的长度/指示符缓冲区分配值

- 普通列的情况应用程序将新的列分配在*TargetValuPtr缓冲区值的长度分配于*StrLen_or_IndPtr缓冲区如果不更新行则应用程序在行操作数组中将对用行的元素分配于SQL_ROW_IGNORE
- Data-at-execution column的情况应用程序将列编号等应用程序定义的值放在*TargetValuePtr缓冲区此值将用于后续的列识别

  应用程序将SQL_LEN_DATA_AT_EXEC(length) macro的结果放在*StrLen_or IndPtr 缓冲区如果列的SQL数据类型为SQL_LONGVARBINARYSQL_LONGVAR_CHAR或源定义的长数据类型

并驱动程序在SQLGetInfo的SQL_NEED_LONG_DATA信息类型中返回"Y"时length是向参数传送的数据的字节数否则必须不能是负数并忽略

2. 为了更新行数据通过Operation参数以SQL_UPDATE调用SQLSetPos

- 没有Data-at-execution column时完成处理
- 有Data-at-execution column时此函数返回SQL_NEED_DATA并执行第三步骤

3. 为了检索要处理的第一个data-at-execution column的*TargetValuePtr缓冲区地址调用SQLParamDataSQLParamData返回SQL_NEED_DATA应用程序在*TargetValuPtr缓冲区检索应用程序定义的值

Note:

* Data-at-execution参数与Data-at-execution column类似但在SQLParamData返回的值不相同

* SQL命令中的data-at-execution参数在命令语以SQLExecDirect或SQLExecute执行时传送到SQLPutData这些通过SQLBindParameter绑定或通过SQLSetDescRec设置描述符后绑定SQLParamData返回的值为32位值传送到SQLBindParameter的ParameterValuePtr参数

* Data-at-execution column是通过SQLSetPos更新行时以SQLPutData传送数据的行集合的column这些通过SQLBindCol绑定SQLParamData返回的值为处理中的*TargetValuePtr缓冲区的行地址

4. 为了传送列的数据调用一次以上的SQLPutData如果SQLPutData指定的*TargetValuePtr缓冲区无法返回所有数据值则需要调用多次只有在将C语言的字符数据以字符二进制或数据源数据类型传送到列或将C语言的二进制数据以字符二进制或数据源定义的数据类型传送到

列时可以对同一个列调用多次SQLPutData

5. 通过调用SQLParamData通知所有的数据已传送到列

- 还有Data-at-execution列时SQLParamData返回SQL_NEED_DATA和下一个处理的data-at-execution列的TargetValuePtr缓冲区地址应用程序反复第四和第五步骤
- 没有data-at-execution列时结束处理如果成功执行命令语则SQLParamData返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO执行失败时返回SQL_ERROR此时SQLParamData返回SQLSetPos可返回的SQLSTATE

如果数据被更新则驱动程序将对应行的实现行状态数组的值变更为SQL_ROW_UPDATED

如果取消操作或SQLParamData或SQLPutData中发生报错则SQLSetPos返回SQL_NEED_DATA后传送所有data-at-execution列之前应用程序可对命令语或命令语相关的连接调用SQLCancelSQLGetDiagFieldSQLGetDiagRecSQLGetFunctionsSQLParamData或SQLPutData此时调用其他函数将返回SQL_ERROR和SQLSATATE HY010 (Function sequence error)

驱动程序仍需要data-at-execution列的数据时如果应用程序调用SQLCancel则驱动程序将取消操作此时应用程序可再次调用SQLSetPos取消不影响游标的状态或当前游标的位置

由驱动程序定义游标相关的查询明细的SELECT-list对同一个列包含一个以上的参照时是否发生报错或忽略重复参照并继续执行操作

执行批量操作

RowNumber参数为0时驱动程序对SQL_ATTR_ROW_OPERATION_PTR命令属性设置的行操作数组的字段中有SQL_ROW_PROCEED值的行集合的所有行执行Operation参数定义的操作

其值对SQL_DELETESQL_REFRESH或SQL_UPDATE的operation参数的RowNumber参数有效但operation参数为SQL_POSITION时无效Operation参数为SQL_POSITIONRowNumber参数为0的

SQLSetPos返回SQLSTATE HY109 (Invalid curosr position)

如SQLSTATE HYT00(timeout expired)所有的行集合中有错误时驱动程序将返回SQL_ERROR和恰当的SQLSTATE不定义行集合缓冲区的内容不变更游标位置

如果单个行发生报错则驱动程序如下运行

- 把SQL_ATTR_ROW_STATUS_PTR命令属性指向的行状态数组的行元素设置为SQL_ROW_ERROR
- 在报错队列的报错中增加一个以上的SQLSTATE设置检测数据结构体的SQL_DIAG_ROW_NUMBER字段

处理完报错或告警后驱动程序完成行集合中剩余的行操作时返回SQL_SUCCESS_WITH_INFO因此对返回报错的行报错队列包含0个或0个以上的SQLSTATE驱动程序处理完报错或告警后暂停运算并返回SQL_ERROR

驱动程序返回SQLSTATE 01004 (Data truncated) 等告警时在返回特定行的错误信息之前返回整个行集合或行集合的未知行的告警返回这些行的错误信息的同时返回特定行的告警

RowNumber为0Operation为SQL_UPDATESQL_REFRESH或SQL_DELETE时SQL_ATTR_ROWS_FETCHED_PTR命令属性设置SQLSetPos运行的行数

RowNumber 为0 运算为SQL_DELETESQL_REFRESH或SQL_UPDATE时运算后的当前行与运算前的当前行相同

批量操作中忽略行

行操作数组可表示通过SQLSetPos进行批量操作时在当前行集合中需要忽略的行在批量操作过

程中驱动程序为了忽略一个以上的行应用程序需要执行以下步骤

1. 调用SQLSetStmtAttr后SQL_ATTR_ROW_OPERATION_PTR命令属性设置SQLUSMALLINT数组此字段也可以通过调用SQLSetDescField后设置ARD的SQL_DESC_ARRAY_STATUS_PTR头部字段的方式进行设置此时应用程序需要先获取描述符句柄

2. 行操作数组的各元素设置为以下两个值中的一个

- 设置为SQL_ROW_IGNORE时表示在批量操作中排除行
- 设置为SQL_ROW_PROCEED时表示在批量操作中包含行

3.调用SQLSetPos执行批量操作

此时以下规则适用于行操作数组

- SQL_ROW_IGNORE和SQL_ROW_PROCEED仅影响以SQL_DELETE或SQL_UPDATE operation使用SQLSetPos的批量操作这些不影响以SQL_REFRESH或SQL_POSITION operation调用SQLSetPos
- 指针默认设置为NULL
- 指针为NULL时所有元素如设置为SQL_ROW_PROCEED一样更新所有行
- 将元素设置为SQL_ROW_PROCEED也无法保证对特定行产生操作例如行集合中的特定行明确为SQL_ROW_ERROR状态时不管应用程序是否指定SQL_ROW_PROCEED驱动程序均无法更新行应用程序为了查看操作是否成功始终需要检查行状态数组
- SQL_ROW_PROCEED在头部文件中定义为0应用程序为了处理所有行可以将行操作数组初始化为0
- 将行状态数字的第n个元素设置为SQL_ROW_IGNORE并调用SQLSetPos执行批量更新或删除操作时行集合的第n个行在调用SQLSetPos后也不会被变更

- 应用程序应该自动将只读列设置为SQL_ROW_IGNORE

批量操作中忽略列

为了避免试图更新一个以上的只读列而产生不必要的处理检测应用程序可以将绑定的长度/指示符缓冲区的值设置为SQL_COLUMN_IGNORE

详细内容参考**SQLBindCol**

SQLSetScrollOptions

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 2.0的函数SQLSetScrollOptions在ODBC 3.x中被替代为调用SQLGetInfo 和SQLSetStmtAttr

> Note:
>
> 驱动程序对以不支持SqlSetScrollOptions的ODBC 3.x驱动程序运行的应用程序映射SQLSetScrollOptions时驱动程序管理器将SQL_ROW_SET_SIZE命令选项设置为非SQL_ATTR_ROW_ARRAY_SIZE命令属性的SQLSetScrollOptins 的RowsetSize参数因此不能在调用SQLFetch或SQLFetchScroll获取多行数据时使用SQLSetScrollOptions只有在调用SQLExtendedFetch获取多行数据时使用

SQLSetStmtAttr

兼容性

导入版本：ODBC 3.0

遵从标准：ISO 92

概要

SQLSetStmtAttr设置命令语相关属性

语句

```
SQLRETURN SQLSetStmtAttr(
    SQLHSTMT      StatementHandle,
    SQLINTEGER    Attribute,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    StringLength);
```

参数

StatementHandle

【输入】命令语句柄

Attribute

【输入】要设置的属性参考说明部分

ValuePtr

【输入】属性相关的值根据属性值ValutPtr为以下中的一个

•ODBC 描述符句柄

•SQLUINTEGER值

•SQLULEN值

• 以下中的一个指针

◦ Null终止字符

◦ 二进制缓冲区

◦ SQLLENSQLULENSQLUSMALLINT值或值的数组

◦ 驱动程序定义值

Attribute 参数为特定驱动程序值时ValuePtr为整数

StringLength

【输入】Attribute为ODBC定义属性ValuePtr指向字符串或二进制缓冲区时此参数为*ValuePtr的长度Attribute为ODBC定义属性ValuePtr为整数时忽略此参数StringLength可以为以下值

Attribute为驱动程序定义属性时应用程序通过设置StringLength参数表示驱动程序管理器属性的固有值

• ValuePtr为字符串指针时StringLength为字符串的长度或SQL_NTS

• ValuePtr为二进制缓冲区指针时应用程序在StringLength存储SQL_LEN_BINARY_ATTR(length) macro的结果在StringLength存储负数值

• ValuePtr为非字符串或二进制缓冲区的其他值的指针时StringLength应拥有SQL_IS_POINTER的值

• ValuePtr包含固定长度值时StringLength为SQL_IS_INTEGER或SQL_IS_UINTEGER

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	具体的驱动程序信息（函数返回SQL_SUCCESS_WITH_INFO）
01S02	Option value changed	驱动程序不支持ValuePtr指定的值或由于执行条件ValuePtr指定的值已失效而被替代为相近值的情况（通过调用SQLGetStmtAttr可查看临时替代的值）此相近值对StatementHandle有效至关闭游标并在此时命令语属性返回至之前的值命令属性可变更为如下值 SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_ROW_ARRAY_SIZE, SQL_ATTR_SIMULATE_CURSOR （函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
24000	Invalid cursor state	Attribute为 SQL_ATTR_CONCURRENCYSQL_ATTR_CURSOR_TYPESQL_ATTR_SIMULATE_CURSOR 或 SQL_ATTR_USE_BOOKMARKS游标已打开

SQLSTATE	报错	说明
HY000	General error	无特定SQLSTATE的错误*MessageText缓冲区的SQLGetDiagRec返回的错误信息说明错误及原因
HY001	Memory allocation error	无法分配驱动程序执行或完成函数所需的内存
HY009	Invalid use of null pointer	Attribute参数识别的命令语属性需要字符串属性ValuePtr参数为null指针
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了异步执行的函数调用SQLSetStmtAttr时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults 并返回了SQL_PARAM_DATA_AVAILABLE回收所有流式参数的数据之前调用了此函数 对StatementHandle调用了异步执行函数调用此函数时异步执行的函数还在执行中 对StatementHandle调用SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution参数或column的数据之前调用了此函数
HY011	Attribute cannot be set now	Attribute 为SQL_ATTR_CONCURRENCYSQL_ATTR_CURSOR_TYPESQL_ATTR_SIMULATE_SURSOR或 SQL_ATTR_USE_BOOKMARKS命令语为准备状态的情况

SQLSTATE	报错	说明
HY013	Memory management error	由于内存不足无法访问内存对象因此函数未能执行
HY017	Invalid use of an automatically allocated descriptor handle	Attribute 参数为SQL_ATTR_IMP_ROW_DESC或SQL_ATTR_IMP_PARAM_DESC Attribute 参数为SQL_ATTR_APP_ROW_DESC或SQL_ATTR_APP_PARAM_DESCValuePtr值并非原来分配于ARD或APD的值而是默认分配的描述符句柄
HY024	Invalid attribute value	考虑已指定的Attribute值时ValuePtr指定的值无效（驱动程序管理器仅向SQL_ATTR_ACCESS_MODE或SQL_ATTR_ASYNC_ENABLE等允许个别集合的命令语属性和连接返回SQLSTATE其他连接与命令语属性的情况驱动程序需验证ValuePtr指定的值） Attribute参数为SQL_ATTR_APP_ROW_DESC或SQL_ATTR_APP_PARAM_DESCValuePtr是指定分配的描述符句柄与SatetementHandle参数在不同的连接线
HY090	Invalid string or buffer length	*ValuePtr为字符串StringLength参数小于0时（不是SQL_NTS）
HY092	Invalid attribute/option identifier	驱动程序支持的ODBC版本不支持Attribute参数指定的值 Attribute参数指定的值为只读属性的情况

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown tracsaction state. Only disconnect and read-only functions are allowed	挂起状态相关的详细内容参考**SQLEndTran**
HYC00	Optional feature not implemented	驱动程序支持的ODBC支持Attribute参数指定的值但驱动程序不支持 Attribute参数为SQL_ATTR_ASYNC_ENABLE以SQL_ASYNC_MODE的InforType调用SQLGetInfo时返回SQL_AM_CONNECTION的情况 Attribute参数为SQL_ATTR_ENABLE_AUTO_IPD连接属性SQL_ATTR_AUTO_IPD的值为SQL_FALSE的情况
HYT01	Connection timeout expired	响应数据源请求之前连接超时连接超时周期可通过SQL_ATTR_CONNECTION_TIMEOUT的SQLSetConnectAttr进行设置
IM001	Driver does not support this function	驱动程序不支持此函数

SQLSTATE	报错	说明
S1118	Driver does not support asynchronous notification	调用SQLSetStmtAttr设置SQL_ATTR_ASYNC_STMT_EVENT时驱动程序不支持异步提醒

说明

命令语属性有效至由调用其他的SQLSetStmtAttr修改或调用SQLFreeHandle删除为止以SQL_CLOSESQL_UNBIND或SQL_RESET_PARAMS选项调用SQLFreeStmt时不初始化命令属性

如果驱动程序不支持部分命令属性的ValuePtr指定的值则用相近值代替此时驱动程序返回SQL_SUCCESS_WITH_INFO和SQLSTATE 01S02(Option value changed)例如Attribute为SQL_ATTR_CONCURRENCYValuePtr为SQL_CONCUR_ROWVER时如果数据源不支持这些驱动程序用SQL_CONCUR_VALUES替代并返回SQL_SUCCESS_WITH_INFO为了查询替代值应用程序调用SQLGetStmtAttr

ValuePtr设置的信息集合类型取决于指定的AttributeSQLSetStmtAttr允许字符串或整数值中的一个的属性信息类型每个类型的详细内容参考属性的说明部分此格式适用于对SQLGetStmtAttr的各个属性返回的信息StringLength的ValuePtr参数表示的字符串的长度为StringLength

> Note:
>
> * ODBC 3.x 不再使用通过调用SQLSetConnectAttr在连接级别设置属性的功能ODBC 3.x的应用程序绝对不可以在连接级别设置属性ODBC 3.x命令属性中除了SQL_ATTR_METADATA_ID外不能在连接级别设置SQL_ATTR_ASYNC_ENABLE即是连接属性也是命令属性在连接级别或命令语级别均可设置

* ODBC 3.x驱动程序需要与ODBC 2.x应用程序同时运行时ODBC 3.x驱动程序需要在连接级别将驱动程序版本设置为ODBC 2.x 的选项功能

设置描述符字段的命令属性

很多命令属性与描述符的头部字段相对应设置这些属性的结果与设置描述符的字段的结果相同设置字段时相比SQLSetDescField 调用SQLSetStmtAttr不需要获取调用函数的描述符句柄因此更加有效

Caution:

即使对一个命令语调用SQLSetStmtAttr也可能影响其他命令语这种情况在命令语指定分配APD或ARD并且其与其他命令语相关时发生SQLSetStmtAttr修改APD或ARD因此将适用于与该描述符相关的所有命令语如果这个不是故意的操作则应用程序应在调用SQLSetStmtAttr之前（调用SQLSetStmtAttr在其他描述符句柄设置SQL_ATTR_APP_ROW_DESC或SQL_ATTR_APP_PARAM_DESC字段）需要将其他命令语与此描述符进行分离

描述符字段设置为如设置对应的命令属性的结果字段仅适用于可适用于识别为StatementHandle参数的当前命令语的描述符设置的属性不影响后续相关的任何命令语描述符与命令句柄i相关的描述符字段设置为SQLSetDescField时也设置对应的命令属性如果指定分配的描述符与命令语分离头部字段对应的命令语属性将默认还原为分配的描述符字段值

分配命令语时将自动分配四个描述符句柄并与命令语关联指定分配的描述符句柄以SQL_HANDLE_DESC的HandleType调用SQLAllocHandle并分配可调用SQLSetStmtAttr关联命令语

对应描述符头部字段的命令属性为如下

命令语属性	标头字段	说明
SQL_ATTR_PARAM_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	APD
SQL_ATTR_PARAM_BIND_TYPE	SQL_DESC_BIND_TYPE	APD
SQL_ATTR_PARAM_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_PARAM_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IPD
SQL_ATTR_PARAMS_PROCESSED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IPD
SQL_ATTR_PARAMSET_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_ARRAY_SIZE	SQL_DESC_ARRAY_SIZE	ARD
SQL_ATTR_ROW_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	ARD
SQL_ATTR_ROW_BIND_TYPE	SQL_DESC_BIND_TYPE	ARD
SQL_ATTR_ROW_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	ARD
SQL_ATTR_ROW_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IRD
SQL_ATTR_ROWS_FETCHED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IRD

命令语属性

以下表格说明当前定义的属性与导入其的ODBC版本

属性	ValuePtr 内容
SQL_ATTR_APP_PARAM_DESC (ODBC 3.0)	驱动程序不支持
SQL_ATTR_APP_ROW_DESC (ODBC 3.0)	驱动程序不支持
SQL_ATTR_ASYNC_ENABLE (ODBC 1.0)	驱动程序不支持
SQL_ATTR_ASYNC_STMT_EVENT (ODBC 3.8)	驱动程序不支持
SQL_ATTR_ASYNC_STMT_PCALLBACK (ODBC3.8)	驱动程序不支持
SQL_ATTR_ASYNC_STMT_PCONTEXT (ODBC 3.8)	驱动程序不支持
SQL_ATTR_ATOMIC_EXECUTION	SQLUSMALLINT : Atomic insert的运行与否 • SQL_ATOMIC_EXECUTION_OFF • SQL_ATOMIC_EXECUTION_ON

SQL_ATTR_CONCURRENCY
(ODBC 2.0)
	SQL_ATTR_CONCURRENCY的默认值为SQL_CONCUR_READ_ONLY
	
	SQL_ATTR_CURSOR_TYPE Attribute被变更为SQL_ATTR_CONCURRENCY不支持的值时
	SQL_ATTR_CONCURRENCY 的值在执行时变更调用SQLExecDirect 或SQLPrepare 时
	告警
	
	驱动程序支持SELECT FOR UPDATE 命令语执行这些命令语的过程中如果
	SQL_ATTR_CONCURRENCY变更为SQL_CONCUR_READ_ONLY则报错如果
	SQL_ATTR_CONCURRENCY变更为驱动程序支持的SQL_ATTR_CURSOR_TYPE值或不支
	则SQL_ATTR_CURSOR_TYPE值在执行时被变更并在执行SQLExecDirect 或SQLPrepar
	错SQLSTATE 01S02 (option value changed)
	
	如果数据源不支持指定的并发性时驱动程序用其他的并发性替代并返回SQLSTATE 0
	(option value changed)驱动程序把SQL_CONCUR_VALUES替代为SQL_CONCUR_ROWV
	之亦然另外SQL_CONCUR_LOCK以SQL_CONCUR_ROWVERSQL_CONCUR_VALUES顺序
	值的有效性直到执行时间点才能查看
	
	• SQLULEN: 指定游标并发性的值
	  ◦ SQL_CONCUR_READ_ONLY: 只读游标不允许更新
	  ◦ SQL_CONCUR_LOCK: 游标使用可完成更新行的最低级别的锁
	  ◦ SQL_CONCUR_POWVER: 如SQLBase ROWID或Sybase TIMESTAMP等
	  用比较行版本的并发控制
	  ◦ SQL_CONCUR_VALUES: 游标使用控制并比较值的并发性

属性	ValuePtr 内容
SQL_ATTTR_CURSOR_SCROLLABLE (ODBC 3.0)	• SQLULEN: 定义应用程序要求的支持级别的值设置此属性影响后续SQLE 和SQLExecute ○ SQL_NONSCROLLABLE: 其为默认值命令语句柄不要求滚动游标如果 序在此句柄调用SQLFetchScroll则FetchOrientation的唯一有效值为 SQL_FETCH_NEXT ○ SQL_SCROLLABLE: 命令语句柄要求滚动游标调用SQLFetchScroll 时 序在除顺序模式外的其他模式下定位游标并定义FetchOrientation的
SQL_ATTR_CURSOR_SENSITIVITY (ODBC 3.0)	• SQLULEN: 是否允许命令语句柄的游标查看其他游标结果集变化的值设 影响后续的SQLExecDirect和SQLExecute的调用为了获取应用程序最近设 始状态或状态值应用程序可重新读取此属性值 ○ SQL_UNSPECIFIED: 未定义是否允许游标的类型和命令语句柄的游标 他游标结果集变化命令语句柄的游标可能一点都看不到或看到部分 为默认值使用 ○ SQL_INSENSITIVE: 命令语句柄的游标无法查看其他游标生成的任何 能看到结果insensitive游标为只读游标其属于只读的支持并发性的静 ○ SQL_SENSITIVE: 命令语句柄的游标可以查看其他游标的结果集的所

属性	ValuePtr 内容

SQL_ATTR_CURSOR_TYPE
(ODBC 2.0)
	默认值为SQL_CURSOR_FORWARD_ONLY此属性不能在准备的(prepared)SQL命令语中
	如果数据源不支持指定的游标类型则驱动程序用其他类型代替并返回SQLSTATE 01S02(Option value changed)驱动程序将混合的游标或动态游标替代为keyset-driven游标驱动程序用静态游标替代keyset-driven游标
	• SQLULEN：指定游标类型的值
	  ○ SQL_CURSOR_FORWARD_ONLY: 游标只能向前移动
	  ○ SQL_CURSOR_KEYSET_DRIVEN: 驱动程序只存储并使用与SQL_ATTR_KEYSET_SIZE属性定义的行相同数量的KEY
	  ○ SQL_CURSOR_DYNAMIC: 驱动程序存储并使用行集合的行的KEY

SQL_ATTR_ENABLE_AUTO_IPD(ODBC 3.0)	驱动程序不支持
SQL_ATTR_EXPLAIN_PLAN_OPTION	SQLUSMALLINT : 是否生成plan信息 • SQL_EXPLAIN_PLAN_OFF : 不生成plan信息 • SQL_EXPLAIN_PLAN_ON : 执行SQL语句并生成plan信息 • SQL_EXPLAIN_PLAN_ONLY : 不执行SQL语句只生成plan信息
SQL_ATTR_EXPLAIN_PLAN_TEXT	生成的plan字符串（只读）

属性	ValuePtr 内容
SQL_ATTR_FETCH_BOOKMARK_PTR (ODBC 3.0)	驱动程序不支持
SQL_ATTR_FETCH_FAILOVER	SQLUSMALLINT：是否使用fetch failover • SQL_FETCH_FAILOVER_OFF：不使用fetch failover • SQL_FETCH_FAILOVER_ON：使用fetch failover
SQL_ATTR_IMP_PARAM_DESC (ODBC 3.0)	驱动程序不支持
SQL_ATTR_IMP_ROW_DESC (ODBC 3.0)	驱动程序不支持
SQL_ATTR_KEYSET_SIZE (ODBC 2.0)	驱动程序不支持
SQL_ATTR_MAX_LENGTH (ODBC 1.0)	驱动程序不支持

属性	ValuePtr 内容
SQL_ATTR_MAX_ROWS (ODBC 1.0)	此属性的目的为减少网络消耗理论上生成结果集时适用此属性限于第一个ValuePtr行 集合结果集的行数量大于ValuePtr 时结果集会减少 SQL_ATTR_MAX_ROWS适用于包含目录函数返回的命令语的所有结果集 SQL_ATTR_MAX_ROWS设置游标行数量值的最大值 无法保证SQL_ATTR_MAX_ROWS是否正确执行时（数据源无法限制结果集大小时）驱 不能对SQLFetch或SQLFetchScroll模仿SQL_ATTR_MAX_ROWS操作 由驱动程序定义是否对SELECT 命令语（目录函数等）外的其他命令语适用 SQL_ATTR_MAX_ROWS 此属性值可设置于已打开的游标但不会立即生效此时驱动程序返回SQLSTATE 01S02 value changed) 并把属性设置为原来的值 • SQLULEN：SELECT 命令语返回的最大行数量的值*ValuePtr为0时驱动程 所有行

属性	ValuePtr 内容
SQL_ATTR_METADATA_ID (ODBC 3.0)	SQL_TRUE时目录函数将字符串参数识别为标识符此时不区分大小写对未指定范围的 驱动程序将去掉后置空白并将字符串转换为大写对指定范围的字符串驱动程序去掉前 和后置空白并直接使用分隔符之间的字面上的内容此参数中的一个设置为空指针时函 SQL_ERROR和SQLSTATE HY009 (Invalid use of null pointer) SQL_FALSE时 不将目录函数的字符串参数识别为标识符此时区分大小写根据因数参 不包含字符串检索模式 默认值为SQL_FALSE 使用值目录的SQLTables的TableType参数值不受此属性的影响 SQL_ATTR_METADATA_ID也可在连接级别设置（其与SQL_ATTR_ASYNC_ENABLE为中 命令属性也是属性） 详细内容参考**目录函数的参数** • SQLULEN：设置怎样使用目录函数的字符串参数的值
SQL_ATTR_NOSCAN (ODBC 1.0)	驱动程序不支持

属性	ValuePtr 内容
SQL_ATTR_PARAM_BIND_OFFSET_PTR (ODBC 3.0)	绑定offset始终直接添加至 SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTRSQL_DESC_OCTET_LENGTH_PTR字 改offset值则新的值直接添加至描述符字段值新的offset字段值不会添加至之前的offs 通过设置此目录属性设置APD头部中的SQL_DESC_BIND_TYPE字段 • SQLULEN*：指为了变更动态参数的绑定添加至指针的offset的值此字段 NULL时驱动程序反向参考指针并反向参考描述符记录(SQL_DESC_DATA_ SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR) 中延迟的 添加反向参考的值绑定时使用新的指针值其默认设置为NULL
SQL_ATTR_PARAM_BIND_TYPE (ODBC 3.0)	此字段设置为列式绑定用的SQL_PARAM_BIND_BY_COLUMN（默认值） 选择行式绑定时此字段应设置为结构体的长度或动态参数集合要绑定的缓冲区实例此 含绑定参数和结构体填充所需的空间或绑定参数的地址以指定的长度增加时应缓冲至 果必须指向下一个参数的开始使用ANSI C 的sizeof运算符时保证其操作 通过设置此属性设置APD头部的SQL_DESC_BIND_TYPE字段 • SQLULEN：表示动态参数中要使用的绑定方向

属性	ValuePtr 内容
SQL_ATTR_PARAM_OPERATION_PTR (ODBC 3.0)	设置APD中的SQL_DESC_ARRAY_STATUS_PTR指向的数组状态值可在处理的过程中忽 集合参数集合在此状态值设置为SQL_PARAM_PROCEED或未设置数组要素时进行处理 此目录属性可以设置为空指针此时驱动程序不返回参数状态值可随时设置此属性但设 个SQLExecDirect 或SQLExecute之前不使用新的值 没有绑定的参数时忽略此属性 通过设置此目录属性设置APD头部的SQL_DESC_ARRAY_STATUS_PTR字段 • SQLUSMALLINT*：执行SQL命令语过程中指向用于忽略参数的SQLUSMA 的数组的值各值为SQL_PARAM_PROCEED(为了执行参数)或 SQL_PARAM_IGNORE(忽略参数)

属性	ValuePtr 内容
SQL_ATTR_PARAM_STATUS_PTR (ODBC 3.0)	此属性可以设置为空指针此时驱动程序不返回参数的状态值可以随时设置此属性但 个SQLExecDirect 或 SQLExecute之前不使用新的值设置此属性可能影响驱动程序输 行为 通过设置此目录属性设置IPD头部的SQL_DESC_ARRAY_STATUS_PTR字段 • SQLUSMALLINT*：调用SQLExecute或SQLExecDirect 后指向包含参数的 状态信息值的SQLUSMALLINT值的数组的值此字段仅在PARAMSET_SIZE 使用状态值可以包含以下值 ○ SQL_PARAM_SUCCESS：SQL命令语对此参数集合成功执行 ○ SQL_PARAM_SUCCESS_WITH_INFO：SQL命令语对此参数集合成功 检测数据结构体有告警信息 ○ SQL_PARAM_ERROR：处理参数集合时发生错误详细的错误信息在 结构体中 ○ SQL_PARAM_UNUSED：由于之前的参数集合出现中断处理的错误或 SQL_ATTR_PARAM_OPERATION_PTR指定的数组参数设置为 SQL_PARAM_IGNORE而未使用参数集合 ○ SQL_PARAM_DIAG_UNAVAILABLE：由于不生成错误信息级别驱动 组作为一个整体单元进行处理

属性	ValuePtr 内容
SQL_ATTR_PARAMS_PROCESSED PTR (ODBC 3.0)	通过设置此目录属性设置IPD头部的SQL_DESC_ROWS_PROCESSED_PTR字段 用于填充此属性中表示的缓冲区的SQLExecDirect或SQLExecute不返回SQL_SUCCESS SQL_SUCCESS_WITH_INFO时不定义缓冲区的内容 • SQLULEN*：作为指向返回已处理的参数集合数量的缓冲区的记录字段包 集合不返回空指针
SQL_ATTR_PARAMSET_SIZE (ODBC 3.0)	如果没有绑定的变量则忽略此属性 通过设置此命令属性设置APD头部的SQL_DESC_ARRAY_SIZE字段 • SQLULEN：指定各参数值数量的值SQL_ATTR_PARAMSET_SIZE大于1时A SQL_DESC_DATA_PTRSQL_DESC_INDICATOR_PTRSQL_DESC_OCTET_LEN 指向数组每个数组常数等于字段的值

属性	ValuePtr 内容
SQL_ATTR_QUERY_TIMEOUT (ODBC 1.0)	如果指定的超时值超过数据源的最大值或小于最小值时SQLSetStmtAttr替代值并返回 SQLSTATE 01S02(Option value changed) 即使SELECT命令语超时应用程序为了再次使用不需要调用SQLCloseCursor 设置此命令属性的查询超对同步方式与异步方式均有效 • SQLULEN：执行SQL命令语后向应用程序返回之前等待的秒单位的时间 ValuePtr为0（默认值）时没有超时
SQL_ATTR_RETRIEVE_DATA (ODBC 2.0)	驱动程序不支持
SQL_ATTR_ROW_ARRAY_SIZE (ODBC 3.0)	指定的行集合大小超过数据源支持的行集合大小的最大值时驱动程序将替代值并返回 SQLSTATE 01S02 (option value changed) 通过设置此命令属性设置ARD头部的SQL_DESC_ARRAY_SIZE字段 • SQLULEN：指定调用SQLFetch或SQLFetchScroll 后返回的行数量的值其 SQLBulkOperations中用于批量书签运算的书签数组的行数量默认值为1

属性	ValuePtr 内容
SQL_ATTR_ROW_BIND_OFFSET_PTR (ODBC 3.0)	通过设置此命令属性设置ARD头部的SQL_DESC_BIND_OFFSET_PTR字段 • SQLULEN：为了表示列数据绑定的变更而添加的offset值此字段不为空时 序将反向参考指针并在描述符记录(SQL_DESC_DATA_PTR, SSQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR)的各个字 反向参考值绑定时使用新的指针值默认值为空
SQL_ATTR_ROW_BIND_TYPE (ODBC 1.0)	如果指定长度则为了column的地址增加一定长度时指向相同的column的开始需要包 充所有已绑定column和结构体或缓冲区的空间可在ANSI C的结构体或共用体中使用s 算保证其行为 SQLFetch和SQLFetchScroll的默认绑定方向为列式绑定 通过设置此命令属性设置ARD头部的SQL_DESC_BIND_TYPE字段 • SQLULEN：相关命令语调用SQLFetch 或SQLFetchScroll时设置绑定方向 设置为SQL_BIND_BY_COLUMN则采用列式绑定 如果结果column设置要 构体或缓冲区的实例长度的值则采用行式绑定
SQL_ATTR_ROW_NUMBER (ODBC 2.0)	驱动程序不支持

属性	ValuePtr 内容
SQL_ATTR_ROW_OPERATION_PTR (ODBC 3.0)	驱动程序不支持
SQL_ATTR_ROW_STATUS_PTR (ODBC 3.0)	此命令属性可设置为空指针此时驱动程序不返回行状态值可以随时设置此属性但直到 SQLBulkOperationsSQLFetchSQLFetchScroll或SQLSetPos 之前不使用新的值 通过设置此命令属性设置IRD头部的SQL_DESC_ARRAY_STATUS_PTR字段 此属性在ODBC 2.x 驱动程序中映射到SQLExtendedFetch 的rgbRowStatus 数组 • SQLUSMALLINT*：指向调用SQLFetch或SQLFetchScroll后包含行状态值的 SQLUSMALLINT数组值的值数组拥有与包含在行集合的行数相同数量的元
SQL_ATTR_ROWS_FETCHED_PTR (ODBC 3.0)	通过设置此属性设置IRD头部的SQL_DESC_ROWS_PROCESSED_PTR字段 调用此属性指向的填充缓冲区的SQLFetch 或SQLFetchScroll时 如果不返回SQL_SUCC SQL_SUCCESS_WITH_INFO则不定义缓冲区的内容 • SQLULEN*：指返回调用SQLFetch或SQLFetchScroll后回收的行数量的缓 SQL_REFRESH的Operation因数调用SQLSetPos后执行批量操作影响的行 SQLBulkOperations执行批量操影响的行数量此行数量包含报错行

属性	ValuePtr 内容
SQL_ATTR_SIMULATE_CURSOR (ODBC 2.0)	驱动程序不支持
SQL_ATTR_USE_BOOKMARKS (ODBC 2.0)	驱动程序不支持

SQLSetStmtOption

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 3.x中ODBC 2.0的函数SQLSetStmtOption被替代为SQLSetStmtAttr

详细内容参考**SQLSetStmtAttr**

SQLSpecialColumns

兼容性

导入版本：ODBC 1.0

遵从标准：Open group

概要

SQLSpecialColumns函数对指定表中的列检索以下信息

- 唯一识别表的行的最佳column集合
- 更行行值时事务自动更行的column

语句

```
SQLRETURN SQLSpecialColumns(
    SQLHSTMT      StatementHandle,
    SQLSMALLINT   IdentifierType,
    SQLCHAR *     CatalogName,
    SQLSMALLINT   NameLength1,
    SQLCHAR *     SchemaName,
    SQLSMALLINT   NameLength2,
    SQLCHAR *     TableName,
    SQLSMALLINT   NameLength3,
    SQLSMALLINT   Scope,
```

```
    SQLSMALLINT   Nullable);
```

参数

StatementHandle

【输入】命令语句柄

IdentifierType

【输入】返回的列的类型应为以下值中的一个

• SQL_BEST_ROWID：在column(s)检索值后返回在指定表唯一识别所有行的最佳的列或列集合列可以是按照特定目的设计的pseudo column（ORACLE的ROWID或INGERS的TID）普通列或表的固有索引column

• SQL_ROWVER：一个事务更新行值（SQLBase ROWID或Sybase TIMESTAMP等）时返回由数据源自动更新的指定表的column

CatalogName

【输入】表目录驱动程序不支持目录时返回空支付穿("")其表没有目录CatalogName不能包含字符串检索pattern

将SQL_ATTR_METADATA_ID设置为SQL_TRUE时CatalogName识别为标识符并不区分大小写如果设置为SQL_FALSECatalogName识别为普通字符串参数并区分大小写详细内容参考**目录函数的参数**

NameLength1

【输入】*CatalogName字符串的长度

SchemaName

【输入】表SCHEMA的名称schema名称的字符串检索模式驱动程序不支持Schema时返回空字符串("")其表没有SchemaSchemaName不能包含字符串检索pattern

将SQL_ATTR_METADATA_ID设置为SQL_TRUE时SchemaName识别为标识符并不区分大小写设置为 SQL_FALSE时SchemaName将作为普通的字符串参数并区分大小写

NameLength2

【输入】*SchemaName 字符串的长度

TableName

【输入】表名此参数不能使用NULL pointerTableName不能包含字符串检索模式

将SQL_ATTR_METADATA_ID设置为SQL_TRUE时TableName将作为标识符并不区分大小写设置为SQL_FALSE时TableName将作为普通的参数字符串并区分大小写

NameLength3

【输入】*TableName 字符串的长度

Scope

【输入】所需的最小的ROWID范围返回的ROWID范围可以为更大的范围其应为以下值中的一个

• SQL_SCOPE_CURROW：ROWID仅在位于当前行时有效被其他事务更新或删除时不能查询使用中的ROWID

• SQL_SCOPE_TRANSACTION：ROWID仅在维持当前事务时有效

• SQL_SCOPE_SESSION：ROWID在（超出事务的界限）维持当前会话时有效

Nullable

【输入】查看指定列是否可以为NULL值应为以下中的一个

• SQL_NO_NULLS：指定列不能为NULL值部分驱动程序不支持SQL_NO_NULLS这些驱动程序在指定SQL_NO_NULLS时将返回空结果集为了这种情况应用程序必须在需要时请求SQL_NO_NULLS

• SQL_NULLABLE：指定列可以有NULL值

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	各驱动程序信息(函数返回SQL_SUCCESS_WITH_INFO)
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败

SQLSTATE	报错	说明
24000	Invalid cursor state	StatementHandle中已打开游标并调用了SQLFetch或SQLFetchScroll SQLFetch或SQLFetchScroll返回SQL_NO_DATA时驱动程序返回该错误如果SQLFetch或SQLFetchScroll未返回SQL_NO_DATA则由驱动程序管理器返回该错误 StatementHandle中已打开游标但未调用SQLFetch或SQLFetchScroll时
40001	Serialization failure	因其他事务的资源产生死锁而事务被回滚的情况
40003	Statement completion unknown	相关连接在函数执行过程中失败因此无法查看事务状态
HY000	General error	没有特定SQLSTATE的报错*MessageText缓冲区的SQLGetDiagRec返回的错误信息中说明了详细的错误及原因
HY001	Memory allocation error	驱动程序无法分配函数执行或结束所需的必要内存

SQLSTATE	报错	说明
HY008	Operation canceled	可在StatementHandle使用异步处理调用并完成执行此函数之前StatementHandle调用了SQLCancel 或SQLCancelHandle函数之后 此函数再次被StatementHandle调用 调用并完成此函数之前多线程应用程序从其他线程在StatementHandle调用了SQLCancel或SQLCancelHandle
HY009	Invalid use of null pointer	TableName参数为空指针 SQL_ATTR_METADATA_ID语句属性设置为SQL_TRUEColumnName参数为空指针 SQL_CATALOG_NAME InfoType返回支持目录名 SQL_ATTR_METADATA_ID设置为SQL_TRUESchemaName参数为空指针

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了SQL异步执行函数并在调用SQLSpecialColumns时此异步执行的函数还在执行中 对StatementHandle执行了SQLExecuteSQLExecDirect,或SQLMoreResults 并返回了 SQL_PARAM_DATA_AVAILABLE回收流式参数的数据之前调用了此函数 对StatementHandle调用了异步执行函数并在调用此函数时该异步执行函数仍在执行中 对StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution参数和column的数据之前执行了此函数
HY013	Memory management error	由于内存不足无法访问内存对象而函数执行失败
HY090	Invalid string or buffer length	长度参数中有一个值小于0时（但不是SQL_NTS） 名称长度参数中有一个值超过了最大长度
HY097	Column type out of range	指定了无效的标识符
HY098	Scope type out of range	指定了无效的范围值

SQLSTATE	报错	说明
HY099	Nullable type out of range	指定了无效的Nullable值
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	挂起状态相关的详细内容参考**SQLEndTran**
HYC00	Optional feature not implemented	驱动程序或数据源不支持指定的目录 驱动程序和数据源不支持指定的Schema 目录名表schema表名数据源所有或其中的一个以上不支持指定的字符串检索模式 驱动程序和数据源不支持SQL_ATTR_CONCURRENCY和SQL_ATTR_CURSOR_TYPE的当前状态组合 SQL_ATTR_USE_BOOKMARKS设置为SQL_UB_VARIABLESQL_ATTR_CURSOR_TYPE在驱动程序设置为不支持书签的游标类型
HYT00	Timeout expired	从数据源接收请求的结果集之前语句执行超时超时周期可通过SQLSetStmtAttr的SQL_ATTR_QUERY_TIMEOUT设置

SQLSTATE	报错	说明
HYT01	Connection timeout expired	数据源返回请求的结果集之前语句执行超时其连接超时周期可通过SQL_ATTR_CONNECTION_TIMEOUT的SQLSetConnectAttr设置
IM001	Driver does not support this function	驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	使用提醒模式时无法使用轮询
IM018	SQLCompleteAsync had not been called to complete the previous asynchronous on this handle.	之前的函数调用对句柄返回SQL_STILL_EXECUTING并告警模式生效时为了后处理和完成操作应对句柄调用SQLCompleteAsync

说明

IdentifierType参数为SQL_BEST_ROWID时SQLSpecialColumns返回可唯一识别列或表的各行的列这些列可用于select-list或WHERE子句SQLColumns返回表column的多种信息但不需要返回更新行时自动更新的唯一识别列或各行的列例如SQLColumns不返回Oracle pseudo column的ROWID这就是SQLSpecialColumns 用于返回特殊列的信息的理由详细内容参考**目录数据的使用**

> Note:
>
> 相关ODBC目录函数的一般用法参数返回数据的详细内容参考**31.4 目录函数**

如果没有唯一识别表的各行的列时SQLSpecialColumns不返回任何列之后对命令语调用SQLFetch或SQLFetchScroll时返回SQL_NO_DATA

如果数据源不支持IdentifierTypeScopeNullable参数指定的属性则SQLSpecialColumn返回空结果集

SQL_ATTR_METADATA_ID属性设置为SQL_TRUE时CatalogNameSchemaNameTableName参数被当作标识符因此无法使用空指针（详细内容参考 **目录函数的参数** ）

SQLSpecialColumns函数返回以SCOPE排序的标准结果集合

在ODBC 3.x中重命名以下column应用程序以column编码绑定因此column名的变更不影响与之前版本的兼容性

ODBC 2.0 列	ODBC 3.x 列
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SCALE	DECIMAL_DIGITS

为了查看COLUMN_NAME列的实际长度应用程序可以与SQL_MAX_COLUMN_NAME_LEN选项同时调用SQLGetInfo 函数

以下表格列出了结果集的列驱动程序可以定义column 8(PSEUDO_COLUMNN)以后添加的column应用程序为了访问驱动程序定义的列应该从结果集的尾开始倒计时而不是指定一个明确的位置详细内容参考**目录函数的数据返回**

字段名	字段编号	数据类型	说明
SCOPE (ODBC 1.0)	1	Smallint	ROWID的实际范围包含 SQL_SCOPE_CURROWSQL_SCOPE_TRANSACTION或 SQL_SCOPE_SESSION值中的一个 IdentifierType为SQL_ROWVER时返回NULL 各个值的详细内容参考语句的scope说明部分
COLUMN_NAME (ODBC 1.0)	2	Varchar not NULL	Column名如果列没有名称则驱动程序返回空字符串
DATA_TYPE (ODBC 1.0)	3	Smallint not NULL	SQL 数据类型其可以为ODBC SQL数据类型或驱动程序指定的SQL 数据类型所有ODBC SQL数据类型均有效驱动程序指定的SQL数据类型的详细内容参考驱动程序的文档
TYPE_NAME (ODBC 1.0)	4	Varchar not NULL	依赖于数据源的数据类型名称例如有 CHARVARCHARMONEYLONG VARBINARYCHAR () FOR BIT DATA等
COLUMN_SIZE (ODBC 1.0)	5	Integer	数据源的列的长度
BUFFER_LENGTH (ODBC 1.0)	6	Integer	指定SQL_C_DEFAULT时SQLGetData或SQLFetch运算所传送的字节单位的数据长度Numeric数据大小可能与数据源存储的数据大小不一致其值为字符串或binary数据时此值与COLUMN_SIZE列的值相同

字段名	字段编号	数据类型	说明
DECIMAL_DIGITS (ODBC 1.0)	7	Smallint	数据源中列的小数点位数如果不能适用数据类型的小数点位数则返回NULL
PSEUDO_COLUMN (ODBC 2.0)	8	Smallint	如Oracle ROWID表示列是否为pseudo-column 为了兼容性pseudo-column不能引用SQLFetInfo返回的标识符quote • SQL_PC_UNKNOWN • SQL_PC_NOT_PSEUDO • SQL_PC_PSEUDO

应用程序检索SQL_BEST_ROWID值后可以使用这些值在定义的范围内重新选择行SELECT语句保证返回no rows和one row中的一个

如果应用程序基于ROWID或column重新查询行时如果未查询到行则对应的行可能被删除或ROWID列可能被修改即使ROWID未发生变更对应行的其他列可能被变更

应用程序需要在行的集合中检索最新数据的结果集合内前后移动时对column类型SQL_BEST_ROWID返回的column非常有用定位对应行时ROWID的column(s)不会被变更

即使游标不位于行ROWID的列仍然有效应用程序可以通过检查结果集的SCOPE列查看这些

SQLStatistics

兼容性

导入版本：ODBC 1.0

遵从标准：ISO 92

概要

SQLStatistics函数检索一个表和表相关索引的统计目录驱动程序以结果集返回结果

语句

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

参数

StatementHandle

【输入】命令语句柄

CatalogName

【输入】目录名驱动程序不支持目录时返回空支付穿("")其表没有目录目录名不能包含字符串检索模式

SQL_ATTR_METADATA_ID设置为SQL_TRUE时CatalogName将作为不区分大小写的标识符设置为SQL_FALSE时CatalogName区分大小写并作为普通的字符串参数详细内容参考**目录函数的参数**

NameLength1

【输入】*CatalogName字符串的长度

SchemaName

【输入】SCHEMA名称驱动程序不支持Schema时空字符串返回("")其表没有Schema

SQL_ATTR_METADATA_ID设置为SQL_TRUE时SchemaName将作为不区分大小写的标识符设置为SQL_FALSE时SchemaName区分大小写并作为模式值字符串参数

NameLength2

【输入】*SchemaName 字符串的长度

TableName

【输入】表名此参数不能为空指针TableName不包含字符串检索模式

SQL_ATTR_METADATA_ID设置为SQL_TRUE时SchemaName将作为不区分大小写的标识

符设置为SQL_FALSE时SchemaName区分大小写并作为模式值字符串参数

NameLength3

【输入】*TableName 字符串的长度

Unique

【输入】索引的类型SQL_INDEX_UNIQUE或SQL_INDEX_ALL

Reserved

【输入】表示结果集的CARDINALITY和PAGES column的重要性以下选项仅影响CARDINALITY和PAGES column的结果即使不返回CARDINALITY和PAGES也返回索引信息

• SQL_ENSURE：请求驱动程序无条件检索统计信息（遵守open group标准但不支持的ODBC扩展的驱动程序无法支持SQL_ENSURE）

• SQL_QUICK：仅在服务器方便使用的情况请求驱动程序无条件检索CARDINALITY和PAGES此时驱动程序不保证检索的值一定为最新值（登记到open group的应用程序从遵守ODBC 3.x的驱动程序获取SQL_QUICK操作）

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SSQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	Genera warning	各个驱动程序的信息(函数返回SQL_SUCCESS_WITH_INFO)
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
24000	Invalid cursor state	StatementHandle中已打开游标并调用了SQLFetch或SQLFetchScroll SQLFetch 或SQLFetchScroll返回SQL_NO_DATA时驱动程序返回此错误SQLFecth或SQLFetchScroll未返回SQL_NO_DATA时驱动程序管理器返回此错误 StatementHandle中已打开游标但未调用SQLFetch 或SQLFetchScroll
40001	Serialization failure	由于其他事务的资源产生死锁事务被回滚
40003	Statement completion unknown	执行此函数的过程中相关连接失败无法查看事务状态
HY000	General error	无特定SQLSTATE的错误详细的错误信息和原因返回于*MessageText缓冲区的SQLGetDiagRec

SQLSTATE	报错	说明
HY001	Memory allocation error	驱动程序无法分配函数的执行或完成所需的内存
HY008	Operation canceled	可异步处理StatementHandle调用并完成执行此函数之前StatementHandle调用了SQLCancel 或SQLCancelHandle函数之后此函数再次被StatementHandle调用 调用并完成此函数之前多线程应用程序从其他线程在StatementHandle调用了SQLCancel或SQLCancelHandle
HY009	Invalid use of null pointer	TableName 参数为空指针 SQL_ATTR_METADATA_ID语句属性设置为SQL_TRUECAtalogName为空指针SQL_CATALOG_NAME InfoType返回支持目录名称 SQL_ATTR_METADATA_ID设置为SQL_TRUESchemaName参数为空指针

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的连接句柄调用了异步执行函数调用SQLStatistics时该异函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults并返回了SQL_PARAM_DATA_AVAILABLE回收所有流式参数的数据之前调用了此函数 对StatementHandle调用了异步执行函数调用此函数时异步执行的函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirectSQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution参数或column数据之前调用了SQLStatistics
HY013	Memory management error	由于内存不足无法访问内存对象而函数执行失败
HY090	Invalid string or buffer length	长度参数中有一个值小于0（但不是SQL_NTS） 名称长度参数中有一个值超过了最大长度
HY100	Uniqueness option type out of range	指定了无效的唯一值

SQLSTATE	报错	说明
HY101	Accuracy option type out of range	制定了无效的预留字值
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	静止状态相关的更多内容参考**SQLEndTran**函数
HYC00	Optional feature not implemented	驱动程序或数据源不支持指定的目录 驱动程序或数据源不支持指定的Schema 目录名表schema表名数据源中的一个或全部不支持指定的字符串检索模式 驱动程序和数据源不支持SQL_ATTR_CONCURRENCY和SQL_ATTR_CURSOR_TYPE的当前状态组合 SQL_ATTR_USE_BOOKMARKS设置为SQL_UB_VARIABLESQL_ATTR_CURSOR_TYPE设置为不支持书签的游标类型

SQLSTATE	报错	说明
HYT00	Timeout expired	数据源返回请求的结果集数据之前查询超时超时周期可通过SQL_ATTR_QUERY_TIMEOUT的SQLSetStmtAttr设置
HYT01	Connection timeout expired	数据源响应之前连接周期超时该连接超时周期通过SQL_ATTR_CONNECTION_TIMEOUT的SQLSetConnectAttr设置
IM001	Driber does not support this function	驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	句柄的之前的函数调用返回SQL_STILL_EXECUTING并提醒模式生效时为了后处理和结束操作句柄必须调用SQLCompleteAsync

说明

SQLStatistics以NON_UNIQUETYPEINDEX_QUALIFIERINDEX_NAMEORDINAL_POSITION顺序的标准结果集返回一个表的信息结果集结合各个索引的信息和表的统计信息（结果集的CARDINALITY和PAGES行）详细内容参考**目录数据的使用**

应用程序中为了决定TABLE_CATTABLE_SCHEMTABLE_NAMECOLUMN_NAME行的实际长度与SQL_MAX_CATALOG_NAME_LENSQL_MAX_SCHEMA_NAME_LENSQL_MAX_TABLE_NAME_LENSQL_MAX_COLUMN_NAME_LEN同时调用SQLGetInfo函数

> Note:
>
> ODBC目录函数的一般用法参数返回数据的详细内容参考 **目录函数**

在ODBC 3.x中重命名以下column应用程序以column编码绑定因此column名的变更不影响与之前版本的兼容性

ODBC 2.0 行	ODBC 3.x 行
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM
SEQ_IN_INDEX	ORDINAL_POSITION
COLLATION	ASC_OR_DESC

以下表格列出结果集的列驱动程序可以自定义column 13(FILTER_CONDITION) 以后添加的列应用程序应该从结果集的尾开始计数访问特定的驱动程序字段而不是指定一个明确的序数位置详细内容参考**目录函数的数据返回**

行名	行编号	数据类型	说明
TABLE_CAT (ODBC 1.0)	1	Varchar	目录名无法转换数据源时为NULL驱动程序不支持目录时返回空字符串("")其表没有目录

行名	行编号	数据类型	说明
TABLE_SCHEM (ODBC 1.0)	2	Varchar	schema名无法适用于数据源时为NULL驱动程序不支持schema时返回空字符串("")其表没有shcema
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	适用统计或索引的表的名称
NON_UNIQUE (ODBC 1.0)	4	Smallint	表示索引是否允许值的重复 TYPE为SQL_TABLE_STAT时返回NULL • SQL_TRUE：索引值可以为nonunique • SQL_FALSE：索引值必须为unique
INDEX_QUALIFIER (ODBC 1.0)	5	Varchar	执行DROP INDEX并规定索引名称的标识符 不支持索引规则的数据源或TYPE为SQL_TABLE_STAT时返回NULL如果此行返回非NULL的值时此值用于定义在DROP INDEX识别的索引否则必须适用TABLE_SCHEM规定索引名
INDEX_NAME (ODBC 1.0)	6	Varchar	索引名称TYPE为SQL_TABLE_STAT时返回NULL

行名	行编号	数据类型	说明
TYPE (ODBC 1.0)	7	Smallint not NULL	返回的信息的类型 • SQL_TABLE_STAT：表示（CARDINALITY或PAGES row的)表统计 • SQL_INDEX_BTREE：表示B-Tree索引 • SQL_INDEX_CLUSTERED：表示集群索引 • SQL_INDEX_CONTENT：表示索引的内容 • SQL_INDEX_HASHED：表示Hash索引 • SQL_INDEX_OTHER：表示其他类型的索引
ORDINAL_POSITION (ODBC 1.0)	8	Smallint	索引的行编号（从1 开始）TYPE为SQL_TABLE_STAT时返回NULL
COLUMN_NAME (ODBC 1.0)	9	Varchar	列的名称行为类似SALARY + BENEFITS的表达式时返回表达式如果无法查看表达式则返回空字符串TYPE为SQL_TABLE_STAT时返回NULL
ASC_OR_DESC (ODBC 1.0)	10	Char(1)	column排序顺序A表示升序D表示降序数据源不支持行排列顺序或TYPE为SQL_TABLE_STAT时返回NULL
CARDINALITY (ODBC 1.0)	11	Integer	表或索引的CardinalityTYPE为SQL_TABLE_STAT时是表行的数量TYPE为非SQL_TABLE_STAT时是索引的唯一值的数量从数据源无法使用值时返回NULL

行名	行编号	数据类型	说明
PAGES (ODBC 1.0)	12	Integer	索引或表存储的页数TYPE为SQL_TABLE_STAT时是表的页数TYPE为非SQL_TABLE_STAT时是索引的页数无法使用或转换数据源的值时返回NULL
FILTER_CONDITION (ODBC 2.0)	13	Varchar	索引为类似SALARY > 30000的已过滤的索引时是索引条件如果无法查看过滤状态时为空字符串如果不是索引则无法查看是否为过滤索引或TYPE为SQL_TABLE_STAT

结果集的行对应于表时驱动程序将TYPE设置为SQL_TABLE_STAT将NON_UNIQUEINDEX_QUALIFIERINDEX_NAMEORDINAL_POSITIONCOLUMNASC_OR_DESC设置为NULL如果从数据源无法使用CARDINALITY或PAGES则驱动程序将其设置为NULL

SQLTablePrivileges

兼容性

导入版本：ODBC 1.0

遵从标准：ODBC

概要

SQLTablePrivileges 返回表的列表及各表相关的权限驱动程序在指定语句返回结果集类型的信息

语句

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

参数

StatementHandle

【输入】检索结果的命令语句柄

CatalogName

【输入】表目录驱动程序不支持目录时返回空字符串("")其表没有目录CatalogName不能包含字符串检索模式

SQL_ATTR_METADATA_ID设置为SQL_TRUE时CatalogName将作为不区分大小写的标识符设置为SQL_FALSE时CatalogName区分大小写并作为普通的参数字符串详细内容参考**目录函数的参数**

NameLength1

【输入】*CatalogName的字符串长度

SchemaName

【输入】SCHEMA名称的字符串检索模式驱动程序不支持Schema时返回空字符串("")其表没有Schema

SQL_ATTR_METADATA_ID设置为SQL_TRUE时SchemaName将作为不区分大小写的标识符设置为SQL_FALSE时SchemaName区分大小写并作为模式值字符串参数

NameLength2

【输入】*SchemaName 字符串长度

TableName

【输入】表名字符串检索模式

SQL_ATTR_METADATA_ID设置为SQL_TRUE时TableName将作为不区分大小写的标识符设置为SQL_FALSE时TableName区分大小写并作为模式值字符串参数

NameLength3

【输入】*TableName 字符串的长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General warning	各驱动程序的信息（函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败
24000	Invalid curosr state	StatementHandle中已打开游标并调用了SQLFetch或SQLFetchScroll SQLFetch或SQLFetchScroll返回SQL_NO_DATA时驱动程序返回其错误如果SQLFetch或SQLFetchScroll未返回SQL_NO_DATA则驱动程序管理器返回其错误 StatementHandle中已打开游标但未调用SQLFetch或SQLFetchScroll的情况
40001	Serialization failure	由于其他事务的资源产生死锁事务被回滚

SQLSTATE	报错	说明
40003	Statement completion unknown	相关连接在执行函数的过程中失败因此无法查看事务状态
HY000	General error	没有特定SQLSTATE的错误详细的错误信息和原因返回于*MessageText缓冲区的SQLGetDiagRec
HY001	Memory allocation error	驱动程序无法分配执行或完成函数所需的内存
HY008	Operation canceled	StatementHandle中可使用异步处理调用并完成执行此函数之前StatementHandle调用SQLCancel或SQLCancelHandle之后函数再次被StatementHandle调用 调用并结束此函数之前多线程应用程序从其他线程在StatementHandle调用SQLCancel 或SQLCancelHandle的情况
HY009	Invalid use of null pointer	SQL_ATTR_METADATA_ID语句属性设置为SQL_TRUEColumnName为空指针SQL_CATALOG_NAME InfoType返回支持目录名称 SQL_ATTR_METADATA_ID设置为SQL_TRUESchemaName或TableName参数为空指针

SQLSTATE	报错	说明
HY010	Function sequence error	对StatementHandle相关的连接句柄调用SQL异步执行函数调用SQLTablePrivilege时此函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults并返回了SQL_PARAM_DATA_AVAILABLE从所有流式参数回收数据之前调用了此函数 对StatementHandle调用了异步执行函数调用此函数时异步执行函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirect,SQLBulkOperation或SQLSetPos并返回了SQL_NEED_DATA传送所有data-at-execution参数和column的数据之前调用了SQLTableprevileges
HY013	Memory management error	由于内存不足无法访问内存对象而函数执行失败
HY090	Invalid string or buffer length	长度参数中有一个值小于0（不是SQL_NTS） 名称长度参数中有一个值超过最大长度

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	挂起状态相关的详细内容参考**SQLEndTran**
HYC00	Optional feature not implemented	驱动程序或数据源不支持指定的目录 驱动程序或数据源不支持指定的schema 目录名表schema表名称数据源的全部或其中一个不支持指定的字符串检索模式 驱动程序或数据源不支持SQL_ATTR_CONCURRENCY和SQL_ATTR_CURSOR_TYPE的当前状态组合 SQL_ATTR_USE_BOOKMARKS设置为SQL_UB_VARIABLESQL_ATTR_CURSOR_TYPE在驱动程序设置为不支持书签的游标类型
HYT00	Timeout expired	接收数据源请求的结果集之前语句执行超时其超时周期可通过SQL_ATTR_CONNECTION_TIMEOUT的SQLSetStmtAttr设置

SQLSTATE	报错	说明
HYT01	Connection timeout expired	响应数据请求之前超时其连接超时周期可通过SQL_ATTR_CONNECTION_TIMEOUT的SQLSetConnectAttr设置
IM001	Driver does not support this function	驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	对句柄的之前的函数调用返回SQL_STILL_EXECUTING提醒模式生效时为了后处理和完成操作应对句柄调用SQLCompleteAsync

说明

SchemaName和TableName参数允许检索模式

有效的检索模式的详细内容参考**pattern值参数**

SQLTablePrivileges以TABLE_CATTABLE_SCHEMTABLE_NAMEPRIVILEGEGRANTEE顺序的标准结果集返回结果

应用程序中为了查看TABLE_CATTABLE_SCHEMTABLE_NAME column的实际长度与SQL_MAX_CATALOG_NAME_LENSQL_MAX_SCHEMA_NAME_LENSQL_MAX_TABLE_NAME_LEN同

时调用SQLGetInfo函数

> Note:
>
> ODBC目录函数的一般用法参数返回数据的详细内容参考目录函数

在ODBC 3.x中重命名以下column应用程序以column的编号绑定因此column名的变更不影响与之前版本的兼容性

ODBC 2.0 列	ODBC 3.x 列
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

以下表列出了结果集的列驱动程序可以定义Column 7(IS_GRANTABLE)之后添加的column应用程序应该从结果集的尾开始倒计时访问特定的驱动程序column而不是指定一个明确位置详细内容参考目录函数的数据返回

列名	列编号	数据类型	说明
TABLE_CAT (ODBC 1.0)	1	Varchar	目录名称无法适用于数据源时为NULL驱动程序不支持目录时返回空字符串("")其表没有目录
TABLE_SCHE (ODBC 1.0)	2	Varchar	schema名无法适用于数据源时为NULL驱动程序不支持schema时返回空字符串("")其表没有schema
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	表名

列名	列编号	数据类型	说明
GRANTOR (ODBC 1.0)	4	Varchar	赋予权限的用户的名称无法适用于数据源时为NULL GRANTEE列的值为对象的所有者的所有行时GRANTOR column为_SYSTEM
GRANTEE (ODBC 1.0)	5	Varchar not NULL	被赋予权限的用户名称
PRIVILEGE (ODBC 1.0)	6	Varchar not NULL	表的权限是下列中的一个或数据源指定的权限 被赋予表权限的GRANTEE被允许范围取决于数据源例如UPDATE权限是grantee可以在一个数据源中更新表的所有column只能更新grantor拥有更新权限的其他数据源的column • SELECT：允许GRANTEE查询一个以上的表列 • INSERT：允许GRANTEE向表插入拥有一个以上的column数据的新数据 • UPDATE：允许GRANTEE更新一个以上的表列 • DELETE：允许GRANTEE删除表的数据 • REFERENCES：允许GRANTEE在约束条件中参考一个以上的表column（例：查看uniquereferential表的约束条件）

列名	列编号	数据类型	说明
IS_GRANTABLE (ODBC 1.0)	7	Varchar	表示GRANTEE是否可以向其他用户赋予权限其值为YES或"NO如果无法适用于数据源或无法知道则为NULL 只能为可赋予或不可赋予中的一个 SQLColumnPrivileges返回的结果集在除了IS_GRANTABLE外的所有列不能包含有相同值的两个行

SQLTables

兼容性

导入版本：ODBC 1.0

遵从标准：Open group

概要

SQLTables返回指定数据源存储的表的列表目录或Schema名称表类型驱动程序以结果集返回信息

语句

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

参数

StatementHandle

【输入】检索结果的命令语句柄

CatalogName

【输入】目录名称SQL_ODBC_VERSION环境属性为SQL_OV_ODBC3时CatalogName参数允许检索模式如果驱动程序只支持部分表的目录如驱动程序从另一个数据库中检索数据时空字符串("")表示没有目录的表

SQL_ATTR_METADATA_ID命令属性设置为SQL_TRUE时CatalogName作为不区分大小写的标识符设置为SQL_FALSE时CatalogName为模式值参数即按照字面处理并区分大小写详细内容参考**目录函数的参数**

NameLength1

【输入】*CatalogName的字符长度

SchemaName

【输入】Schema名称的字符串检索模式与CatalogName相同空字符串("")表示没有Schema的表另外与CatalogName相同参数的处理取决于SQL_ATTR_METADATA_ID命令属性

NameLength2

【输入】*SchemaName的字符长度

TableName

【输入】表名的字符串检索模式与CatalogName相同参数的处理取决于

SQL_ATTR_METADATA_ID命令属性

NameLength3

【输入】*TableName的字符长度

TableType

【输入】匹配的表类型的列表

SQL_ATTR_METADATA_ID命令属性不影响TableType参数TableType与

SQL_ATTR_METADATA_ID的设置无关是值列表参数

NameLength4

【输入】*TableType的字符长度

返回

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTEING, SQL_ERROR, or

SQL_INVALID_HANDLE

检测

SQLSTATE	报错	说明
01000	General waring	各驱动程序的信息（函数返回SQL_SUCCESS_WITH_INFO）
08S01	Communication link failure	在函数完成处理之前驱动程序和数据源之间的连接失败

SQLSTATE	报错	说明
24000	Invalid cursor state	StatementHandle中已打开游标调用了SQLFetch或SQLFetchScroll SQLFetch或SQLFetchScroll返回SQL_NO_DATA时驱动程序返回此错误如果SQLFetch或SQLFetchScroll未返回SQL_NO_DATA则驱动程序管理者返回此错误 StatementHandle中已打开游标但未调用SQLFetch或SQLFetchScroll
40001	Serialization failure	由于其他事务该资源发生死锁而事务被回滚
40003	Statement completion unknown	执行此函数的过程中相关连接失败无法查看事务的状态
HY000	General error	没有特定SQLSTATE的错误*MessageText 缓冲区的SQLGetDiagRec中返回详细的错误信息和原因
HY001	Memory allocation error	驱动程序无法分配执行或完成函数所需的内存

SQLSTATE	报错	说明
HY008	Operation canceled	可异步处理StatementHandle调用并完成执行此函数之前StatementHandle调用了SQLCancel或SQLCancelHandle之后StatementHandle再次调用了此函数 调用并完成此函数之前多线程应用程序从其他线程在函数在StatementHandle调用了SQLCancel 或SQLCancelHandle
HY009	Invalid use of null pointer	SQL_ATTR_METADATA_ID语句属性设置为SQL_TRUECatalogName参数为空指针 SQL_CATALOG_NAME InfoType返回支持目录名称 SQL_ATTR_METADATA_ID设置为SQL_TRUESchemaName 或TableName参数为空指针

SQLSTATE	报错	说明
HY010	Function sequence error	StatementHandle相关的连接句柄调用了异步执行的函数但调用SQLTables时此函数还在执行中 对StatementHandle调用了SQLExecuteSQLExecDirect或SQLMoreResults并返回了SQL_PARAM_DATA_AVAILABLE传送所有流式参数的数据之前调用了此函数 对StatementHandle调用了异步执行的函数调用SQLTables函数时此异步执行的函数还在执行中 对StatementHandle调用SQLExecuteSQLExecDirect或SQLMoreResults并返回了SQL_NEED_DATA传送所有data-at-execution参数或column的数据之前调用了此函数
HY013	Memory management error	由于内存不足无法访问内存而函数执行失败
HY090	Invalid string or buffer length	长度参数中有一个值小于0（不是SQL_NTS） 名称长度参数中有一个值超过了最大长度

SQLSTATE	报错	说明
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed	静止状态相关的更多内容参考 **SQLEndTran**
HYC00	Optional feature not implemented	驱动程序或数据源不支持指定的目录 驱动程序或数据源不支持指定的Schema 目录名表schema表名称数据源的全部或其中一个不支持指定的字符串检索模式 驱动程序或数据源不支持SQL_ATTR_CONCURRENCY和SQL_ATTR_CURSOR_TYPE的当前状态组合 SQL_ATTR_USE_BOOKMARKS设置为SQL_UB_VARIABLESQL_ATTR_CURSOR_TYPE在驱动程序设置为不支持书签的游标类型
HYT00	Timeout expired	返回数据源请求的结果集之前语句执行超时其超时周期可通过SQL_ATTR_CONNECTION_TIMEOUT的SQLSetStmtAttr设置

SQLSTATE	报错	说明
HYT01	Connection timeout expired	数据源响应请求之前连接超时其连接超时周期可通过SQL_ATTR_CONNECTION_TIMEOUT的SQLSetConnectAttr设置
IM001	Driver does not support this function	驱动程序不支持此函数
IM017	Polling is disabled in asynchronous notification mode	当使用提醒模式时轮询是禁用的
IM018	SQLCompleteASync has not been called to complete the previous asynchronous operation on this handle.	句柄的之前的函数调用返回SQL_STILL_EXECUTING提醒模式生效时为了后处理和结束操作必须对句柄调用SQLCompleteAsync

说明

SQLTables列出请求范围内的所有表用户对此表拥有或未拥有SELECT权限应用程序如下查看访问性

- 调用SQLGetInfo查看SQL_ACCESSIBLE_TABLES信息类型
- 调用SQLTablePrivileges查看对各个表权限

否则应用程序应该能够应对用户再没有SELECT权限的情况下执行select的情况

SchemaName与TableName参数可使用检索模式SQL_ODBC_VERSION为SQL_OV_ODBC3 时

CatalogName参数也能使用检索模式

有效的检索模式相关详细内容参考**pattern值参数**

> Note:
>
> ODBC目录函数的用法参数返回数据的详细内容参考**目录函数**

为了支持列出目录Schema表类型如下定义SQLTables的

CatalogNameSchemaNameTableNameTableType参数的特殊含义

- 如果CatalogName为SQL_ALL_CATALOGSSchaemaName和TableName为空字符串时结果集包含数据源的有效目录 (除TABLE_CAT column外的所有列均包含NULL)
- 如果SchemaName为SQL_ALL_SCHEMASCatalogName和TableName为空字符串时结果集包含数据源的有效Schema (除TABLE_SCHE column外的所有列均包含NULL)
- 如果TableType为SQL_ALL_TABLE_TYPESCatalogNameSchemaNameTableName为空字符串时结果集包含数据源的有效的表类型(除TABLE_TYPE column外的所有列均包含NULL)
- 如果TableType不是空字符串应该包含以逗号区分相关类型的值的目录各值应以'TABLE''VIEW'TABLEVIEW等形式用单引号引住或不使用引号应用程序始终用大写指定表类型驱动程序按照数据源请求的形式转换TableType如果数据源不支持指定的表类型则SQLTables不返回对应类型的数据结果

SQLTables以TABLE_TYPETABLE_CATTABLE_SCHETABLE_NAME顺序的标准结果集返回结果详细内容参考**目录数据的使用**

应用程序中为了决定TABLE_CATTABLE_SCHEMTABLE_NAME column的实际长度与SQL_MAX_CATALOG_NAME_LENSQL_MAX_SCHEMA_NAME_LENSQL_MAX_TABLE_NAME_LENSQL_MAX_COLUMN_NAME_LEN同时调用SQLGetInfo

在ODBC 3.x中重命名了以下column应用程序以column编号绑定因此字段名的变更不影响与之前版本的兼容性

ODBC 2.0列	ODBC 3.x列
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

以下表列出了结果集的列驱动程序可以定义Column 5(REMARKS)之后添加的列应用程序应该从结果集的尾开始倒计时访问特定的驱动程序column而不是指定一个明确的序数位置

详细内容参考**目录函数的数据返回**

字段名	字段编号	数据类型	说明
TABLE_CAT (ODBC 1.0)	1	Varchar	目录名称无法适用于数据源时为NULL驱动程序不支持目录时返回空字符串("")其表没有目录
TABLE_SCHEM (ODBC 1.0)	2	Varchar	schema名称无法适用于数据源时为NULL驱动程序不支持schema时返回空字符串("")其表没有schema
TABLE_NAME (ODBC 1.0)	3	Varchar	表名
TABLE_TYPE (ODBC 1.0)	4	Varchar	表类型名称是"TABLE""VIEW""SYSTEM TABLE""GLOBAL TEMPORARY""LOCAL TEMPORARY""ALIAS""SYNONYM"或数据源指定的名称中的一个 各个驱动程序的"ALIAS"和"SYNONYM" 的含义均不相同

字段名	字段编号	数据类型	说明
REMARKS (ODBC 1.0)	5	Varchar	表的说明

SQLTransact

兼容性

导入版本：ODBC 1.0

遵从标准：无

概要

ODBC 3.x中ODBC 2.x的SQLTransact 函数被替代为SQLEndTran函数

详细内容参考**SQLEndTran**函数

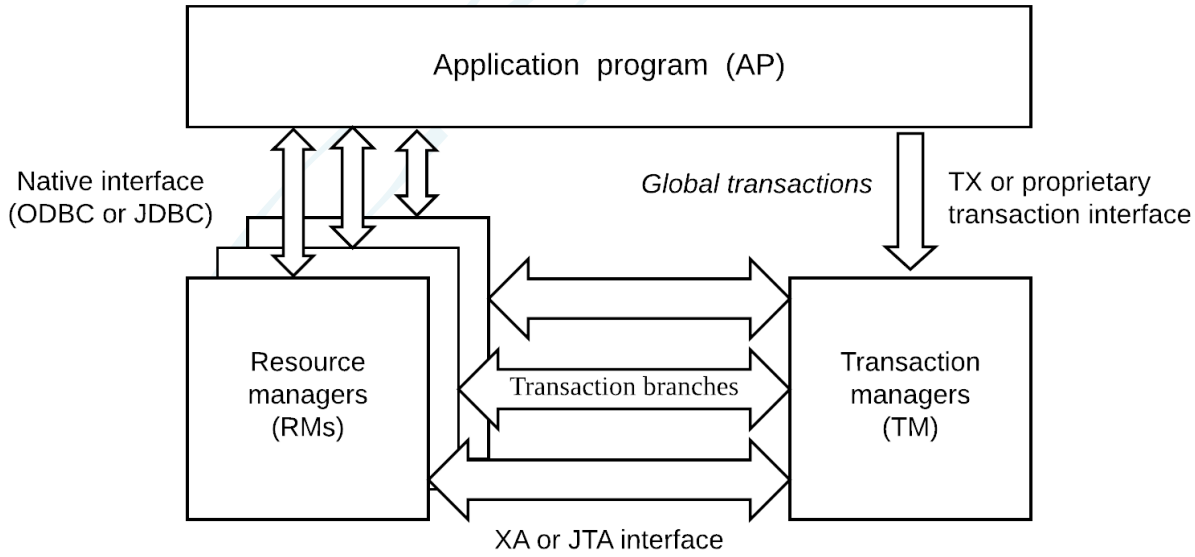
2.7 XA API References

概要

X/Open XA是X/Open制定的处理分布式事务的标准一般规定多事务管理器与本地资源管理器之间的接口XA规定资源管理器处理事务所需的内容

SUNDB XA是在X/Open CAE文档Distributed Transaction Processing: The XA Specification (http://www.opengroup.org/public/catalog/c193.htm)基础上实现的

X/Open Distributed Transaction Processing (DTP)模型定义其他电脑型号不同的数据库之间的事务管理



- Appication Program (AP)： 定义由事务构成的操作
- Resource Managers (RM)： 管理分布式事务访问的共享资源指SUNDB等数据库管理系统
- Transaction Manager (TM)： 分配分布式事务的ID(XID)管理分布式事务的执行负责分布式

事务的终止与恢复

AP可以是通过预编译或使用ODBC开发的应用程序AP不直接使用XA接口而是使用RM的native interface或使用TM提供的TX接口控制事务

XA接口

XA接口是RM与TM之间的接口规定SUNDB不单独提供XA接口的库而是包含在SUNDB提供的ODBC库文件中

switch_t结构体

拥有XA接口的进入点（entry point）与RM信息的结构体

SUNDB提供的xa_switch_t是SUNDB_xa_switch

变量名称	说明
char name[RMNAMESZ]	RM的名称
long flags	RM提供的选项 • 不支持MIGRATE选项（设置为TMNOMIGRATE）
long version	RM版本
int (*xa_open_entry)(char *, int, long);	xa_open函数指针
int (*xa_close_entry)(char *, int, long);	xa_close函数指针
int (*xa_start_entry)(XID *, int, long);	xa_start函数指针
int (*xa_end_entry)(XID *, int, long);	xa_end函数指针
int (*xa_rollback_entry)(XID *, int, long);	xa_rollback函数指针

变量名称	说明
int (*xa_prepare_entry)(XID *, int, long);	xa_prepare 函数指针
int (*xa_commit_entry)(XID *, int, long);	xa_commit函数指针
xint (*xa_recover_entry)(XID *, long, int, long);	xa_recover函数指针
int (*xa_forget_entry)(XID *, int, long);	xa_forget函数指针
int (*xa_complete_entry)(int *, int *, int, long);	xa_complete函数指针 • SUNDB不提供

XA相关ODBC函数

说明除标准ODBC外为了使用XA接口而添加的函数

SQLGetXaSwitch

获取ODBC函数提供的xa_switch_t

```
xa_switch_t * SQLGetXaSwitch( void );
```

返回

返回ODBC提供的sw_switch_t结构体的指针不能返回NULL

SQLGetXaConnectionHandle

获取与当前XA会话相关的连接句柄（connection handle）

```
SQLHANDLE SQLGetXaConnectionHandle( void );
```

返回

如果有连接当前线程的XA会话则返回相关连接句柄否则返回NULL

XA函数

详述xa_switch_t结构体的XA相关函数内容

xa_open

连接RM

> Note:
>
> 如已连接调用xa_open的线程则忽视

```
int xa_open(
    char * xa_info,
    int    rmid,
    long   flags );
```

参数

xa_info

【输入】包含访问信息的字符串最大长度为256字节详细内容参考**SQLDriverConnect**的InConnectionString

rmid

【输入】要访问的RM的固有ID忽略此参数

flags

【输入】访问标志应设置为TMNOFLAGS

检测

返回值	说明
XA_OK	操作正常执行
XAER_RMFAIL	RM为不可用状态时发生
XAER_RMERR	由于资源不足等RM无法在事务branch执行操作
XAER_INVAL	传递异常参数时发生

xa_close

终止RM的连接释放连接句柄

```
int xa_close(
    char * xa_info,
    int    rmid,
    long   flags );
```

参数

xa_info

【输入】包含访问信息的字符串忽略此参数

rmid

【输入】要访问的RM的固有ID忽略此参数

flags

【输入】终止标志忽略此参数

检测

返回值	说明
XA_OK	操作正常执行
XAER_RMFAIL	RM为不可用状态时发生
XAER_RMERR	由于资源不足等RM无法在事务branch执行操作
XAER_INVAL	传递异常参数时发生

xa_start

生成新的事务分支或开始现有的事务分支

```
int xa_start(
    XID *  xid,
    int    rmid,
    long   flags );
```

参数

xid

【输入】 要开始的事务ID

rmid

【输入】要访问的RM的固有ID忽略此参数

flags

【输入】开始标志

flags可使用以下值

- TMASYNC：以异步模式开始事务分支（不支持）
- TMNOFLAGS：表示不使用FLAG如果不使用任何FLAG需指定此FLAG
- TMNOWAIT：如果指定的事务分支已被其他会话占用则无需等待并返回XA_RETRY错误
- TMRESUME：继续执行之前中断的事务分支不能与TMJOIN同时使用
- TMJOIN：连接现有的事务分支不能与TMRESUME同时使用

检测

返回值	说明
XA_OK	操作正常执行
XAER_RMFAIL	使用中的会话异常结束时发生
XAER_RMERR	由于资源不足等RM无法在事务branch执行操作
XAER_PROTO	未按照XA协议的顺序执行时发生
XAER_INVAL	传递异常参数时发生
XAER_DUPID	已有拥有相同XID的事务分支
XAER_NOTA	没有指定为XID的事务分支使用TMRESUME或TMJOIN时可能发生
XA_RETRY	指定的事务分支已被其他会话占用时发生使用TMNOWAIT时可能发生
XAER_OUTSIDE	此会话已在执行本地事务
XA_RBROLLBACK	事务分支设置为rollback-only

xa_end

终止事务分支的操作

```
int xa_end(
    XID *  xid,
    int    rmid,
    long   flags );
```

参数

xid

【输入】要终止操作的事务ID

rmid

【输入】要访问的RM的固有ID忽略此参数

flags

【输入】操作终止标志

flags可使用以下值

- TMFAIL：指操作已失败不能与TMSUSPEND或TMSUCCESS一起使用将事务分支的状态变更为rollback-only
- TMMIGRATE：连接其他分支后重新开始（不支持）
- TMSUCCESS：指操作成功完成执行不能与TMSUSPEND或TMFAIL一起使用
- TMSUSPEND：暂停事务分支并终止

检测

返回值	说明
XA_OK	操作正常执行
XAER_RMFAIL	使用中的会话异常结束时发生
XAER_RMERR	由于资源不足等RM无法在事务分支执行操作
XA_NOMIGRATE	不支持MIGRATE标志
XAER_PROTO	未按照XA协议的顺序执行时发生
XAER_INVAL	传递异常参数时发生
XAER_NOTA	不存在指定为XID的事务分支

xa_prepare

准备提交属于XID的事务Two-phase Commit Protocol (2PC)的第一阶段

```
int xa_prepare(
    XID *  xid,
    int    rmid,
    long   flags );
```

参数

xid

【输入】要prepare的事务ID

rmid

【输入】 要访问的RM的固有ID忽略此参数

flags

【输入】 prepare标志应设置TMNOFLAGS

检测

返回值	说明
XA_OK	操作正常执行
XAER_RMFAIL	使用中的会话异常结束时发生
XAER_RMERR	由于资源不足等RM无法在事务分支执行操作
XA_PROTO	未按照XA协议的顺序执行时发生

返回值	说明
XA_RDONLY	事务分支为read-only事务
XAER_NOTA	没有指定为XID的事务分支
XA_RBROLLBACK	事务分支设置为rollback-only

xa_commit

提交属于XID的事务Two-phase Commit Protocol (2PC)的第二个阶段

```
int xa_commit(
    XID *  xid,
    int    rmid,
    long   flags );
```

参数

xid

【输入】要commit的事务的ID

rmid

【输入】 要访问的RM固有ID忽略此参数

flags

【输入】 commit标志

flags不能使用以下值

- TMNOFLAGS : 表示不使用FLAG如果不使用任何FLAG需指定此FLAG
- TMONEPHASE : 执行1PC (One Phase Commit)

检测

返回值	说明
XA_OK	操作正常执行
XAER_RMFAIL	使用中的会话异常结束时发生
XAER_RMERR	由于资源不足等RM无法在事务分支执行操作
XA_PROTO	未按照XA协议的顺序执行时发生
XA_RDONLY	事务分支为read-only事务
XAER_NOTA	没有指定为XID的事务分支
XA_RBROLLBACK	事务分支设置为rollback-only

xa_rollback

回滚属于XID的事务

```
int xa_rollback(
    XID *  xid,
    int    rmid,
    long   flags );
```

参数

xid

【输入】要rollback的事务的ID

rmid

【输入】要访问的RM的固有ID忽略此参数

flags

【输入】 rollback标志需设置TMNOFLAGS

检测

返回值	说明
XA_OK	操作正常执行
XAER_RMFAIL	使用中的会话异常结束时发生
XAER_RMERR	由于资源不足等RM无法在事务分支执行操作
XA_HEURRB	事务分支已被heuristic rollback

返回值	说明
XA_HEURCOM	事务分支已被heuristic commit
XAER_NOTA	没有指定为XID的事务分支

xa_recover

获取heuristic commit或rollback的事务的目录

```
int xa_recover(
    XID *  xids,
    long   count,
    int    rmid,
    long   flags );
```

参数

xids

【输出】执行Heuristic commit或rollback的事务的目录

count

【输入】指xids的数组大小

rmid

【输入】 要访问的RM的固有ID忽略此参数

flags

【输入】 recover标识

flags可使用以下值

- TMSTARTSCAN : 开始扫描或重头开始扫描
- TMENDSCAN : 终止扫描

- TMNOFLAGS : TMSTARTSCAN之后使用获取以下目录如果最初开始使用TMNOFLAGS则发生XA_PROTO错误

检测

返回值	说明
>= 0	返回的XID的数量（ xids中有效的数组的大小）
XAER_RMFAIL	使用中的会话异常结束时发生
XAER_RMERR	由于资源不足等RM无法在事务分支执行操作
XA_PROTO	未按照XA协议的顺序执行时发生

xa_forget

删除heuristic commit或rollback的事务的信息

```
int xa_forget(
    XID *  xid,
    int    rmid,
    long   flags );
```

参数

xid

【输入】要删除的事务的ID

rmid

【输入】 要访问的RM的固有ID忽略此参数

flags

【输入】 删除标志需设置TMNOFLAGS

检测

返回值	说明
XA_OK	操作正常执行
XAER_RMFAIL	使用中的会话异常结束时发生
XAER_RMERR	由于资源不足等RM无法在事务分支执行操作
XAER_PROTO	未按照XA协议的顺序执行时发生

返回值	说明
XAER_INVAL	传递异常参数时发生
XAER_NOTA	没有指定为XID的事务分支

例子

以下为访问SUNDB插入/查询/更新/删除记录后执行Two-phase Commit的简单示例与下例相关的完整代码位于$SUNDB_HOME/sample/ODBC/SAMPLE_XA.c

1. SQLGetXaSwitch()

   获取为了使用XA接口的xa_switch_t结构体

2. SQLAllocHandle()

   获取ODBC环境句柄

3. xa_open()

   根据指定的connection string连接数据库

   如果已连接数据库则使用原有连接否则尝试新连接

4. SQLGetXaConnectionHandle()

   获取线程相关的XA连接句柄

5. xa_start()

   开始事务分支

6. 执行使用标准ODBC函数的事务操作

7. xa_end()

   终止事务操作

8. xa_prepare()

作为2PC的第一个阶段准备提交事务

9. xa_commit()

作为2PC的第二个阶段提交事务

10. xa_close()

断开连接释放连接句柄

也可以使用ODBC标准函数的SQLDisconnect()和SQLFreeHandle()

11. SQLFreeHandle()

释放ODBC环境句柄

```
int main( int aArgc, char** aArgv )
{
    SQLHENV       sEnv    = NULL;
    SQLHDBC       sDbc    = NULL;
    SQLINTEGER    sState  = 0;
    xa_switch_t * sXaSwitch;
    XID           sXid;

    sXaSwitch = SQLGetXaSwitch();
```

- 调用SUNDB ODBC包含的SQLAllocEnv()时

```
SUNDB_SQL_TRY( SQLAllocHandle( SQL_HANDLE_ENV,
                               NULL,
                               &sEnv ) );
    sState = 1;
```

- SQLSetEnvAttr设置管理环境的属性

```
SUNDB_SQL_TRY( SQLSetEnvAttr( sEnv,
                              SQL_ATTR_ODBC_VERSION,
                              (SQLPOINTER)SQL_OV_ODBC3,
                              0 ) );
    if( (sXaSwitch->xa_open_entry)(
               "DSN=SUNDB;UID=test;PWD=test",
               0,
               TMNOFLAGS ) != XA_OK )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
    sState = 2;

    sDbc = SQLGetXaConnectionHandle();

    sXid.formatID = 0;
    sXid.gtrid_length = 2;
    sXid.bqual_length = 1;
    memcpy( sXid.data,
           "100",
           sXid.gtrid_length + sXid.bqual_length );

    if( (sXaSwitch->xa_start_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
```

```
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
```

- 成功执行添加函数的SQL_SUCCESS时

```
SUNDB_SQL_TRY( testInsert( sDbc ) );
```

- 成功执行选择函数的SQL_SUCCESS时

```
SUNDB_SQL_TRY( testSelect( sDbc ) );
```

- 成功执行更新函数的SQL_SUCCESS时

```
SUNDB_SQL_TRY( testUpdate( sDbc ) );
```

- 成功执行删除函数的SQL_SUCCESS时

```
SUNDB_SQL_TRY( testDelete( sDbc ) );


    if( (sXaSwitch->xa_end_entry)( &sXid, 0, TMSUCCESS ) != XA_OK )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }

    if( (sXaSwitch->xa_prepare_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
```

```
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }

    if( (sXaSwitch->xa_commit_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }

    sState = 1;
    if( (sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS ) != XA_OK )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
    sDbc = NULL;
```

- SQLFreeHandleEnv释放与环境相关的资源

```
sState = 0;
    SUNDB_SQL_TRY( SQLFreeHandle( SQL_HANDLE_ENV,
                                  sEnv ) );

    sEnv = NULL;

    return EXIT_SUCCESS;

    SUNDB_FINISH;
```

```
    if( sDbc != NULL)
    {
        PrintDiagnosticRecord( SQL_HANDLE_DBC, sDbc );
    }
    if( sEnv != NULL)
    {
        PrintDiagnosticRecord( SQL_HANDLE_ENV, sEnv );
    }

    switch( sState )
    {
```

- 例2：SQLDisconnect关闭与特定连接句柄相关的连接

```
(void)(sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS );
```

- 例1：SQLFreeHandleEnv释放与环境相关的资源

```
(void)SQLFreeHandle( SQL_HANDLE_ENV, sEnv );
            sEnv = NULL;
        default:
            break;
    }
    return EXIT_FAILURE;
}
```

3.JDBC

3.1 概要

SUNDB JDBC 介绍

SUNDB提供遵守基于TCP/IP连接的标准JDBC 4.0（除部分功能外）的SUNDB JDBC驱动程序用户可以通过SUNDB JDBC驱动程序在JAVA应用程序访问SUNDB并使用各种事务功能及数据查询功能SUNDB JDBC驱动程序是基于JDK 1.6编写并创建的因此支持JDBC 4.0的功能将$SUNDB_HOME/lib/sundb6.jar文件添加至class path后可使用驱动程序

sundb后面 的数字表示JDK版本详细内容参考**版本及支持与否**

SUNDB JDBC基本遵守JDBC标准规格为了提供部分固有功能同时也支持非标准API的Method与CLASS非标准Method相关内容参考**JDBC API References**各CLASS API或 **使用添加类型**

特点

• Type-4 JDBC驱动程序

SUNDB JDBC为纯JAVA实现的JDBC Type-4类型不需要额外的library仅通过jar文件就可以使用JDBC驱动程序不仅如此比JDBC-ODBC bridge类型更快更稳定比使用native API的Type-2具有更优越的移植性(portability)

- **遵循JDBC标准**

SUNDB JDBC遵循JDBC标准因此几乎不需要修改原有的JDBC应用程序而直接使用获取各种连接的方法以及各种statementResultSet功能均不需要修改并直接使用但需要将连接URL参数名要加载的CLASS名修改为符合SUNDB的名称而且可通过额外的CLASS与method API使用非标准的功能与类型

- **支持多种Java版本**

SUNDB JDBC驱动程序支持三个文件(sundb8.jar, sundb7.jar, sundb6.jar)因此可以根据用户的JAVA执行环境选择使用每个jar文件是基于JDK 1.8JDK 1.7JDK 1.6制作创建的因此遵循JDBC 4.2JDBC 4.1JDBC 4.0规格

- **与服务器的灵活的版本兼容性**

服务器与JDBC驱动程序的兼容性参考通信协议版本服务器的通信协议版本高于JDBC驱动程序的通信协议版本时可连接

- **为connection pooling支持API**

JDBC connection对象是生成费用昂贵的资源因此JDBC标准定义了可pooling的体系其对应的接口为ConnectionPoolDataSource与PooledConnectionSUNDB JDBC通过实现此接口向第三方中间件提供可使用pooling功能的基础

- **支持XA API**

通过实现global事务标准的XA接口使用户可以执行global事务操作通过使用JDBC接口的XAResource可按照标准执行各种XA功能

- **SUNDB 固有数据类型**

SUNDB使用JDBC标准不提供的数据类型例如interval相关类型与timestamp with time zone等有time zone信息的类型驱动程序提供在数据库存取这些类型的方法

- **基于服务器的强大的游标滚动功能**

其他JDBC驱动程序为了结果集的滚动将所有的行集合缓存到驱动程序此时客户端应用程序会使用过多的内存例如用scroll insensitive打开游标并获取至最后一条记录时会把所有行缓存到驱动程序此时所有表会长驻在客户端的内存中因此可能发生out of memory错误并过渡消耗客户端内存资源

但SUNDB服务器中有游标滚动功能因此客户端可更加快捷稳定的提高性能

- **有效使用资源**

相比其他JDBC驱动程序使缓存行信息的内存达到最小化并尽量不使用JAVA对象因此在获取大量行时也能避免garbage collection操作并减少内存使用量因此可以更快更稳定地扫描表

- **准确而庞大的元数据**

完美准确的实现了JDBC标准的DatabaseMetaData接口因此便于连接多种数据库工具不仅如此还可通过各种系统视图查询数据库元数据提高了使用性

- **强大的日志功能**

为了监控出现问题的情况以及平时的JDBC API调用或网路使用情况等提供各种logging功能如果在连接URL中指定日志相关功能则可通过控制器或文件等记录相关内容日志有JDBC method call记录相关日志通信协议收发相关日志SQL执行相关日志使用global connection相关日志等四种

- **Connection Failover**

支持JDBC driver级别的connection failover在连接SUNDB数据库失败或execution的过程中连接断开时可自动重新连接并使用原有连接即连接断开时用户不需要做额外处理可通过原有的JDBC程序使用connection failover功能

- **与服务器的direct attach连接**

除了基于TCP/IP的连接外还可以使用可联动服务器等进程的direct attach方式如ODBC连接支持所有direct attachserver/client方式SUNDB JDBC Driver也同样支持两种连接方式通过Direct attach连接的JDBC程序不用与TCP/IP通信也可直接联动服务器进程并在jvm使用服务器的功能其性能也比通过TCP/IP连接的JDBC程序高出两倍以上

版本及支持与否

SUNDB JDBC版本体系

在sundb6.jar文件中执行如下即可查看SUNDB JDBC版本信息

```
shell>java -jar sundb6.jar


 SUNDB JDBC Driver 1.1 Procotol-2.5.2, JDBC4.0 compiled with JDK1.6
```

以上示例的当前SUNDB JDBC驱动程序版本为1.1通信协议版本为2.5.2其驱动程序遵守JDBC 4.0标准是在JDK 1.6创建的驱动程序版本与SUNDB产品版本无关表示驱动程序的版本每次完善功能时升级JDBC驱动程序版本的详细内容参考DatabaseMetaData的

getDriverMajorVersiongetDriverMinorVersiongetDriverVersion

通信协议版本决定是否与服务器兼容与服务器的通信协议版本相同或更低时驱动程序可联动服务器即服务器支持所有下级版本通信协议的客户端API

sundb6.jar遵守JDBC 4.0标准是在JDK 1.6创建的sundb7.jar遵循JDBC 4.1标准创建在JDK 1.7sundb8.jar遵循JDBC 4.2标准是在JDK 1.8创建的因此用户的JAVA环境为JDK 1.8以上时使用sundb8.jar即可但使用JDK 1.9以上的JAVA时可以使用sundb8.jar但无法使用JDBC 4.3的API或CLASS

使用示例

设置CLASSPATH

为了使用SUNDB JDBC驱动程序应设置CLASSPATH

```
export CLASSPATH=.:$SUNDB_HOME/lib/sundb6.jar
```

或也可以在PATH中添加符合用户的JAVA执行环境的jar文件

加载驱动程序CLASS

如下加载驱动程序CLASS

```
Class.forName("csii.sundb.jdbc.SundbDriver");
```

以上是传统的JDBC使用方法动态加载该驱动程序CLASS并登记到DriverManager后获取连接现在

更普遍使用通过数据源获取连接的方法其相关内容参考**JDBC API References**的对应CLASS

获取连接

获取连接需编写如下代码

```
Connection con = DriverManager.getConnection(
    "jdbc:sundb://127.0.0.1:22581/test", "TEST", "test");
```

为了使用SUNDB JDBC的连接URL需要以"jdbc:sundb:"开头随后为服务器的IP与端口号URL最后的"/test"为DB名称当前SUNDB不支持多重DB因此不会特意检查DB名称用户连接的URL可以通过DatabaseMetaData.getURL()再次获取

ip设置为0.0.0.0端口设置为0时以Direct Attach (D/A)模式访问或也可以用da访问协议代替ip:port即如下两个url均以D/A模式访问

```
"jdbc:sundb://0.0.0.0:0/test"
"jdbc:sundb:da/test"
```

Username与password应使用各使用DB的账号和密码

使用Statement与ResultSet

Statement与ResultSet的使用方法与其他JDBC应用程序相同

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT NAME, ADDRESS FROM EMP");
while (rs.next())
```

```
{
    System.out.println("name = " + rs.getString(1));
    System.out.println("address = " + rs.getString(2));
}
rs.close();
stmt.close();
```

3.1 功能详情

连接

使用DriverManager连接

通过传统方法使用DriverManager的获取Connection

```
Connection con = DriverManager.getConnection(url_string, user_name,
password);
```

url_string有如下三种类型

```
jdbc:sundb://[ip address]:[port_no]/[db_name]
```

```
jdbc:sundb:da/[db_name]
```

```
jdbc:sundb:locator//[ip address]:[port_no][, [ip
address]:[port_no] ]*/[db_name]
```

IP address均可使用IPv4和IPv6port_no指定设置的端口号即可db_name不用于当前访问因此可以设置为任何名称但不能省略以下为URL的一个示例

```
String url_string = "jdbc:sundb://localhost:22581/test";
```

```
String url_string = "jdbc:sundb://127.0.0.1:22581/test";
```

```
String url_string = "jdbc:sundb://[::1]:22581/test";
```

ip“0.0.0.0”和端口0用于D/A访问的特殊地址D/A模式的详细内容参考**Ditect Attach模式连接**

user_name和password指SUNDB账号

除了以上getConnection() method外为了设置各种property使用以下method

```
Properties prop = new Properties();
prop.setProperty("user", user_name);
prop.setProperty("password", password);
Connection con = DriverManager.getConnection(url_string, prop);
```

为了从glocator获取服务器的访问信息可以在url_string添加locator关键字代替连接属性此时ip address和port_no视为glocator的访问信息以下为在连接语句使用locator关键字的示例

```
String url_string =
"jdbc:sundb:locator//127.0.0.1:42581,127.0.0.1:42582/test?locator_service=
S1";
```

初次访问信息127.0.0.1:42581是glocator的访问信息后续访问信息如127.0.0.1:42582被处理为ALTERNATE_LOCATORS属性

以下为与上述url_string相同意义的属性设置

```
String url_strng = "jdbc:sundb://0.0.0.0:0/test";
Properties prop = new Properties();
prop.setProperty("locator_host", "127.0.0.1");
prop.setProperty("locator_port", "42581");
prop.setProperty("alternate_locators", "127.0.0.1:42582");
prop.setProperty("locator_service", "S1");
Connection con = DriverManager.getConnection(url_string, prop);
```

连接中可使用的属性目录可以通过SUNDBDriver的getPropertyInfo() method查看详细内容参考

连接属性

使用DriverManager时生成的Connection对象的login timeout与logger使用DriverManager中登记的值即从该Connection对象生成的所有JDBC接口使用的logger就是该logger因此每个Connection对象不能有单独的logger

通过DataSource连接

JDBC标准更加推荐使用DataSource而不是过去的DriverManagerDataSource为可访问某一个data source的单一接口可单独设置各种data source相关的设定值也可远程传送DataSource对象本身

如下可通过DataSource获取connection对象

```
import csii.sundb.jdbc.SundbDataSource;
```

```
SundbDataSource ds = new SundbDataSource();
ds.setServerName("127.0.0.1");
ds.setPortNumber(22581);
ds.setDatabaseName("test");
ds.setUser("TEST");
ds.setPassword("test");
Connection con = ds.getConnection();
```

即为了生成DataSource对象应使用SUNDBDataSource CLASS

之后的各种连接信息应使用setter method而不是DataSource标准API详细内容参考**DataSource**

使用DriverManager时需要全局设置login timeout与logger但DataSource可以单独设置login timeout与logger

```
ds.setLoginTimeout(10);
ds.setLogWriter(out);
```

中间件联动

为了与WebLogicJBoss等中间件联动需要知道实现XADataSourceConnectionPoolDataSource接口的CLASS名称SUNDB在csii.sundb.jdbc产品包中提供SundbXADataSourceSundbConnectionPoolDataSource这两个CLASS都与SundbDataSource同样提供各种setter method

```
import csii.sundb.jdbc.SundbXADataSource;
```

```
SundbXADataSource ds = new SundbXADataSource();
ds.setServerName("127.0.0.1");
ds.setPortNumber(22581);
ds.setDatabaseName("test");
ds.setUser("TEST");
ds.setPassword("test");
XAConnection con = ds.getXAConnection();
```

连接属性

名称	必选/可选	有效值	说明
alternate_servers	可选	IP:PORT[,IP:PORT]+	Failover的替代服务器目录以逗号（）区分
alternate_locators	可选	IP:PORT[,IP:PORT]+	glocator的替代目录

名称	必选/可选	有效值	说明
batch_count	可选	Any integer	一次通信收发可处理的batch job数量默认值为1000该值过小时批量操作中需要频繁通信从而降低性能过大时会话要缓存execution result从而增加服务器内存使用量
connection_retry_count	可选	Any integer	存储连接时重试的次数默认值为0不进行重试
connection_retry_delay	可选	Any integer	以秒为单位指定重试时重试前的delay默认值为3
date_format	可选	Any string	在驱动内部用于date和string之间转换的字符格式
da_buffer_size	可选	Any integer	以direct attach模式访问时指定fetchbind时传输数据的缓冲区大小默认值为100000
decoding_replacement	可选	Any string	将字节数组解码为string时替代无法解码的字节值的字符默认值为 ?

名称	必选/可选	有效值	说明
failover_granularity	可选	{"0", "1", "2"}	判断failover成功与否non-atomic(0), atomic(1)2暂不支持failover过程中对原有的prepared statement执行prepare时发生错误时0表示继续执行failover1表示failover失败默认值为 0
failover_type	可选	{"connection", "session"}	决定failover类型 • Connection：仅在连接服务器时使用failover • Session：不仅在连接服务器时与execution等服务器的所有通信均使用failover默认值为connection
format_grammar	可选	{"db", "java"}	指定date_format等属性字符串为SUNDB语法还是java的SimpleDateFormat中使用的语法默认值为 db

名称	必选/可选	有效值	说明
global_connection_log	可选	boolean	是否进行global connection logging默认值为false
global_logger	可选	{"console"}	指定日志对象 目前只支持console只有第一个指定的有效
home_dir	可选	Any string	指定cluster server的home directory默认值为null
include_synonyms	可选	boolean	决定在DatabaseMetaData.getColumns()中是否包含synonym个体默认值为false
keep_alive	可选	boolean	是否将connection的socket属性设置为keep_alive如果赋予此属性会在TCP socket内部周期性的收发ack并保持相互连接的状态即检测到网线错误后可强行断开连接默认值为false

名称	必选/可选	有效值	说明
locality_aware_transaction	可选	boolean	是否使用GLOBAL CONNECTION 设置为true时使用GLOBAL CONNECTION默认值为false
locality_group_policy	可选	{"0", "1", "2"}	使用GLOBAL CONNECTION时没有可选的组或可选两个以上的组时设置选择组的方法 • 0：任意选择 • 1：按照顺序选择LOCALITY_GROUP_PATH设置中的组无法使用LOCALITY_GROUP_PATH中的所有组时选择任意组 • 2：按照顺序选择总是按照链接驱动程序的顺序选择组

名称	必选/可选	有效值	说明
locality_group_path	可选	Group name list	使用GLOBAL CONNECTION时可选的组不是一个时指定所选组的目录各组以逗号（）区分 例：G1G2G3

名称	必选/可选	有效值	说明
locality_member_policy	可选	{"0","1","2","3","4"}	使用GLOBAL CONNECTION时决定在所选组内选择成员的方法 • 0: DML: MASTER / SELECT : MASTER • 1: DML: ANY / SELECT : ANY • 2: DML: MASTER / SELECT : ANY • 3: DML: MASTER / SELECT : SLAVE • 4: 按照LOCALITY_MEMBER_PATH设置中的成员顺序选择无法使用LOCALITY_MEMBER_PATH中的所有成员时选择所选组的MASTER

名称	必选/可选	有效值	说明
locality_member_path	可选	Member name list	使用GLOBAL CONNECTION时指定所选组使用的成员的目录各成员以逗号（）区分 例：G1N1, G2N1, G3N1, G1N2, G2N2, G3N2
locality_on_demand	可选	boolean	使用GLOBAL CONNECTION时如果有需要连接的成员则连接到该成员上默认值为false • false: 从开始就连接到所有成员上 • true: 必要时连接到该成员上
locator_connection_timeout	可选	Any integer	从glocator获取packet之前的等待时间
locator_file	可选	Any string	location file

名称	必选/可选	有效值	说明
locator_host	可选	IP address	glocator的host address
locator_port	可选	Port no	glocator的port
locator_service	可选	Any string	为获取服务器访问信息的service名称
login_timeout	可选	Any integer	设置与服务器连接时socket的timeout时间（秒）默认值为0无限等待
lzeros	可选	Any integer	把numeric表示为字符串时小数点之后的0的数量超过此值时用exponent标记法表示默认值为15
new_password	可选	Any string	可与old_password属性同时使用变更账号密码
old_password	可选	Any string	可与new_password属性同时使用变更账号密码

名称	必选/可选	有效值	说明
packet_compression_threshold	可选	Any integer	传输至服务器的通信数据的大小大于packet_compression_threshold时压缩通信数据属性值的范围为32~2113929216
password	必选	Any string	用户账号密码
prefer_ipv6	可选	boolean	HOST名称的IP地址中是否优先IPv6默认值为false
program	可选	Any string	程序说明
protocol_log	可选	boolean	是否记录协议收发日志默认值为false
query_log	可选	boolean	是否记录query日志默认值为false
role	可选	{"", "SYSDBA","ADMIN"}	指定用户角色默认值为""

名称	必选/可选	有效值	说明
session_type	可选	{"1", "2", "3"} 或 {"dedicate", "shared", "default"}	dedicated/shared/default中的一个 (选择default时在DB中选择)
statement_pool_on	可选	boolean	激活statement pool默认值为false
statement_pool_size	可选	Any integer	设置statement pool的大小
time_format	可选	Any string	用于在驱动内部Time和String之间转换操作的字符格式
tcp_nodelay	可选	boolean	在connection的socket设置TCP_NODELAY(Nagles' Algorithm)属性默认值为true
timestamp_format	可选	Any string	用于在驱动内部timestamp和string之间转换操作的字符格式
time_with_time_zone_format	可选	Any string	用于在驱动内部time with timezone和String之间转换操作的字符格式

名称	必选/可选	有效值	说明
timestamp_with_time_zone_format	可选	Any string	用于在驱动内部内部timestamp with timezone和String之间转换操作的字符格式
trace_log	可选	boolean	是否记录跟踪日志默认值为false
tzeros	可选	Any integer	把numeric表示为字符串时位数的0的数量超过此值时用exponent标记法表示默认值为15
user	必选	Any string	用户账号名
use_global_session	可选	boolean	是否使用GLOBAL SESSION默认值为false

名称	必选/可选	有效值	说明
use_targettype	可选	{"0", "1", "2"}	通过通信接收column类型时同时接收的信息 • 0：none • 1：name • 2：all

Table 3-1 连接属性

Note:

* locator_file优先于locator_host与locator_portlocator_file的详细内容参考**Location File**

* 通过locator_sevice属性可访问locator_sevice所属的服务器详细内容参考**glocator**与**gloctl**

数据操作

使用Statement的数据操作

可以使用statement对象执行各种SQL语句如下可从connection对象获取statement对象

```
Connection con = DriverManager.getConnection(...);
Statement stmt = con.createStatement();
```

可以使用这样生成的Statement对象的execute method与executeUpdate method执行各种DML或DDL语句

```
stmt.executeUpdate("create table emp ( id varchar(20), name varchar(30),
age integer )");
stmt.executeUpdate("insert into emp values ('1234560000', '金妍儿', 24)");
int updated = stmt.executeUpdate("delete from emp where age > " + age);
```

execute与executeUpdate的区别只有返回值execute返回执行的SQL语句是否返回ResultSetexecuteUpdate返回被更新的行数量execute均可用于DML或SELECT语句但executeUpdate用于SELECT语句时产生SQLException

> Note:
> 用Statement执行SQL语句叫direct-executiondirect-execution是一次性执行语句的prepare操作（parsing验证(validation)优化(optimization)的操作）与执行（execution）的方式相反prepare-execution是只执行一次prepare操作后反复执行

> execution操作的方式由于每次执行direct-execution时都要产生prepareexecution操作
>
> 因此需要反复执行相同SQL时prepare-execution的效率更高

使用PreparedStatement的数据操作

标准JDBC提供可以使用参数操作数据的PreparedStatement接口使用户更加高效的执行而且PreparedStatement对反复执行仅操作一次SQL语句的prepare因此可提高性能

```
Connection con = DriverManager.getConnection(...);
PreparedStatement pstmt = con.prepareStatement(
  "insert into emp values (?, ?, ?)");
pstmt.setString(1, "55544123000");
pstmt.setString(2, "王建国");
pstmt.setInt(3, 32);
pstmt.executeUpdate();
pstmt.setString(1, "1357924680");
pstmt.setString(2, "王振涛");
pstmt.setInt(3, 41);
pstmt.executeUpdate();
```

如2~3行prepare SQL语句后绑定数据并执行对prepare过的PreparedStatement可反复绑定与执行

如果省略绑定后执行execute()则使用之前绑定过的值

```
pstmt.setString(1, "333555777000");
```

```
pstmt.setString(2, "张三丰");
pstmt.setInt(3, 55);
pstmt.executeUpdate();
pstmt.setString(1, "245778884440");
pstmt.executeUpdate();
```

如上在第6行不绑定第2第3个参数并直接执行时使用之前绑定的值"张三丰"与55如果没有之前绑定的值则产生SQLException

如果要删除所有绑定值使用clearParameters()即可

```
pstmt.setString(1, "333555777000");
pstmt.setString(2, "张三丰");
pstmt.setInt(3, 55);
pstmt.executeUpdate();
pstmt.clearParameters();
pstmt.setString(1, "245778884440");
pstmt.executeUpdate();
```

如上在第5行清除所有参数后只在第一个参数绑定值并执行（第7行）则产生SQLException

使用PreparedStatement的最主要原因是因为可以绑定多种类型的数据Statement只能以字符串形式表示值因此不方便插入binary或timestamp等类型的数据但是PreparedStatement可以绑定所有类型因此更加方便

```
pstmt.setTimestamp(1,
  new Timestamp(Calendar.getInstance().getTimeInMillis()));
```

```
pstmt.setCharacterStream(2, new StringReader(BIG_STRING));
pstmt.setObject(3, someObj, Types.LONGVARCHAR);
pstmt.setObject(4, otherObj);
```

上述示例中第1~第2行绑定java.sql.Timestamp对象绑定类型（向服务器传送数据时此数据的SUNDB类型）为TIMESTAMP由各种setter method决定的绑定类型的详细内容参考**PreparedStatement**的对应API

第3行绑定Reader对象内部以LONG VARCHAR类型绑定

第4行绑定JAVA Object类型的对象指定类型为LONG VARCHAR映射于Types类型的SUNDB类型的详细内容参考 **SQL类型 -> SUNDB类型**

第5行直接绑定JAVA Object类型根据CLASS类型绑定为对应的SUNDB数据类型CLASS类型与SUNDB数据类型之间的映射的详细内容参考 **JAVA对象 -> SUNDB类型**

使用Statement的批量操作

可以使用statement的addBatch()与executeBatch()以batch job形式执行互不相同的SQL语句

```
stmt.addBatch("insert into emp values ("12345", "Jake", 22)");
stmt.addBatch("insert into salary values ("12345", 5000)");
stmt.addBatch("update members set total_count=total_count+1 where
age=22");
stmt.executeBatch();
```

如上可以通过一个executeBatch() method执行互不相同的SQL语句但并非通过调用此method将三个语句在服务器执行的结果打包后通过一次的协议传送至JDBC驱动程序而是在内部执行三次

通信协议传输和在服务器的执行结果传输即性能上没有很大优势

执行executeBatch后登记的job均会抹掉

执行过程中即使产生错误也会完成执行所有登记的job是否发生错误可以参考返回的int[]类型的值查看即int[]拥有EXECUTE_FAILED值时意味着其job执行失败

使用PreparedStatement的批量操作

可以使用PreparedStatement的addBatch()与executeBatch() method执行更有效的批量操作

```
PreparedStatement pstmt = con.prepareStatement("insert into emp values
(?, ?, ?)");
pstmt.setString(1, "12345000");
pstmt.setString(2, "John");
pstmt.setInt(3, 33);
pstmt.addBatch();

pstmt.setString(1, "222333555");
pstmt.setString(2, "Henry");
pstmt.setInt(3, 24);
pstmt.addBatch();
int[] inserted = pstmt.executeBatch();
```

以上例子为插入两条行的过程通过addBatch()绑定组成两条行的值之后调用executeBatch()时向服务器传送绑定值与execute的通信协议并执行即只有在executeBatch()时产生通信因此比Statement的批量操作在性能上更有优势

执行executeBatch()后会删除所有绑定值因此再次执行executeBatch()时需要重新绑定新的值执行后为了清除登记的job不需要指定调用clearBatch()

执行过程中即使发生错误也会完成执行所有登记的job可以参考返回的int[]类型的值查看是否产生错误即int[]拥有EXECUTE_FAILED值时意味着其job执行失败

SUNDB JDBC除了executeBatch()外还支持额外的称为executeBatchAtomic()的methodexecuteBatch()的执行次数与服务器上登记的job的数量相同但executeBatchAtomic()一次性执行登记的job（绑定值）因此响应时间更快但从atomic名字也能看出只要有一个失败则整个操作均失败因此返回类型不是int[]型而是int

```
...
int inserted = ((SundbPreparedStatement)pstmt).executeBatchAtomic();
```

查询数据

获取column值

可根据通过statement或PreparedStatement的executeQuery()获取的ResultSet对象查询数据

```
...
ResultSet rs = pstmt.executeQuery();
while(rs.next())
{
    String id = rs.getString(1);
    String name = rs.getString(2);
```

```
    int age = rs.getInt(3);
    ...
}
```

查询表的数据后可通过ResultSet的各种getter method获取对应column的内容表的数据以SUNDB数据类型的原型传送到JDBC驱动程序根据用户调用的getter method种类转换为恰当的Java数据类型并传输给用户SUNDB数据类型与getter method之间的类型转换映射参考 **对SUNDB类型的getter method支持与否-1**

SUNDB数据类型不能转换为对应getter的类型时发生SQLException

Closing ResultSet

可以通过close()解除完成使用的ResultSet即使用户未指定调用close()在以下情况下获取与调用ResultSet的close()相同的结果

- 上级statement被close时
- 上级connection被close时
- 再次调用statement的executeQuery()时
- Fetch过程中发生错误时

上述第三个情况中从一个statement生成的ResultSet不能同时维持两个只有通过最后的executeQuery()生成的ResultSet对象才有效

fetch大小

JDBC可以指定通过Statement的setFetchSize(int rows)每次从服务器获取的row条数SUNDB的ResultSet的此属性默认值为00自动指定从服务器获取的row条数forward only时是一个通信包可

容纳的最大row条数scrollable时固定为100个此值太大则JDBC ResultSet占的内存量大太小则通信交互频繁

此值主要用于scrollable ResultSet因为即使是scroll sensitive在ResultSet的行缓存中移动时不sensitive因此如果此值过于大则无法获取行的最新变更信息

field大小限制

JDBC可通过Statement的setMaxFieldSize(int size)限制一个column的最大长度此设置可以限制CHARVARCHARLONG VARCHARBINARYVARBINARYLONG VARBINARY类型的最大长度将截断超出此长度的数据默认值为0此时不限制最大长度

对INTEGERDATE等其他类型忽略此属性

ResultSet滚动

获取Scrollable ResultSet

Scrollable ResultSet有scroll sensitive与scroll insensitive两种scroll sensitive是在滚动ResultSet时可查看被自身事务或其他事务变更的值的游标类型scroll insensitive为无法查看的游标SUNDB服务器均支持这两种游标类型也可以通过JDBC使用此功能

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                     ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("select id, name, age from emp");
```

如第一行生成statement时如果将ResultSet类型设置为TYPE_SCROLL_SENSITIVE则从此

statement对象生成的ResultSet均成为scroll sensitive游标但目前SUNDB不支持updatable ResultSet

将类型设置为TYPE_SCROLL_INSENSITIVE即可获取scroll insensitive的 ResultSet默认值为TYPE_FORWARD_ONLY

但即使是scroll sensitive ResultSet在ResultSet的行缓存中移动时无法查看最新值也无法查看变更与否只有从服务器再次获取行后才能查看

滚动

JDBC API为游标滚动提供以下method

```
public boolean next() throws SQLException;
public boolean previous() throws SQLException;
public boolean first() throws SQLException;
public boolean last() throws SQLException;
public boolean absolute(int rows) throws SQLException;
public boolean relative(int rows) throws SQLException;
public void beforeFirst() throws SQLException;
public void afterLast() throws SQLException;
```

next()为forward only或scrollable ResultSet均可使用的method将游标移动至下一行可以读取移动的位置的行数据时返回true无法读取时返回false其他method只能在scrollable ResultSet中使用

previous()移动至前一条行first()移动至第一行last()移动至最后一行absolute()移动至绝对位置relative()从当前位置移动至相对位置beforeFirst()移动至第一行之前afterLast()移动到最后一行之后

```
rs.afterLast();
while (rs.previous())
{
    String id = rs.getString(1);
    String name = rs.getString(2);
    int age = rs.getInt(3);
    ...
}
```

以上代码是从最后一行开始倒序查询到第一行

通过以下method可以查看当前游标的位置

```
public int getRow() throws SQLException;
```

滚动的原理

ResultSet滚动使用起来简单但不了解内部运行原理可能造成性能大幅下降因此需谨慎使用游标在JDBC驱动程序的ResultSet的result cache执行但服务器对超出cache的部分执行滚动由此会因此发生通信交互导致性能下降

Scrollable ResultSet的fetch size的内部（internal）默认值为100（外部默认值为0其值可通过setFetchSize() method变更）即每次从JDBC驱动程序获取的行条数为100条ResultSet的行缓存由100条行组成要移动的行位于缓存时更改ResultSet的游标位置即可但如果不在缓存内则需要从服务器上重新获取要移动的位置的行此时需要知道获取包含要移动的行的行集合的方式

例如ResultSet有1号到100号的row当前游标位于第100号时如果调用next()则由于缓存中没有下

一行因此需要从服务器上获取101号到200号row相反ResultSet缓存中有101号到200号的行数据当前游标位于第101号行时调用previous()则如果从服务器上获取100号到199号的数据其为不恰当因为这样实现则每次调用previous()时均要从服务器上获取SUNDB为了防止这种降低效率的情况在需要调用previous()从服务器上获取时从服务器获取以对应行为最后一行的行集合即需要通过previous()获取第100号行时获取第1号到第100条的行这种方式的获取有利于previous()

此规则概括为如下

- 要移动的位置为当前位置的后面时从其位置开始获取100个（设置的fetch size）行数据
- 要移动的位置为当前位置的前面时从其位置开始获取前100个行数据
- 通过first()移动时获取前100个
- 通过last()移动时获取最后100个

即向后移动时使获取有利于next()向前移动时使获取有利于previous()其在absolute()relative()移动时也相同

使用添加类型

Interval类型

SUNDB对于interval支持以下13种类型

- interval year
- interval month
- interval year to month
- interval day

- interval hour
- interval minute
- interval second
- interval minute to second
- interval hour to minute
- interval hour to second
- interval day to hour
- interval day to minute
- interval day to second

如果使用JDBC生成csii.sundb.jdbc.SundbInterval对象则可以通过PreparedStatement的setObject()在表插入interval数据

可使用SundbInterval的static method的createIntervalXXX生成SUNDBInterval对象

```
import csii.sundb.jdbc.SundbInterval;


...
SundbInterval interval = SundbInterval.createIntervalDayToSecond(2, 6, "2
08:23:54.560843");
PreparedStatement pstmt = con.prepareStatement("insert into interval_table
values (?,?)");
pstmt.setString(1, someId);
pstmt.setObject(2, interval);
pstmt.executeUpdate();
```

createIntervalXXX method的详细内容参考**SUNDBInterval**就像如此可生成SundbInterval后通过setObject绑定或为了指定类型可如下执行

```
import csii.sundb.jdbc.SundbTypes;
...
pstmt.setObject(2, interval, SundbTypes.INTERVAL_DAY_TO_SECOND);
```

没有SundbInterval对象的情况下也可以通过String字符串插入数据

```
pstmt.setObject(2, "2 08:23:54.560843",
SundbTypes.INTERVAL_DAY_TO_SECOND);
```

此时可以在JDBC驱动程序中生成SundbInterval对象后绑定主机变量或可通过setString() method以字符串绑定

```
pstmt.setString(2, "2 08:23:54.560843");
```

这样就会把字符串直接传送至服务器在服务器将字符串转换为Interval day to second类型并插入

为了从数据库查询Interval类型可使用ResultSet的getObject()或getString()

```
import csii.sundb.jdbc.SundbInterval;


ResultSet rs = stmt.executeQuery("select interval_value from some_table");
while (rs.next())
{
    SundbInterval interval = (SundbInterval)rs.getObject(1);
```

```
    System.out.println("type = " + interval.getTypeName() +
                       ", value = " + interval.toString());
}
```

可以通过ResultSet的getObject()获取SUNDBInterval类型的数据SUNDBInterval CLASS提供可以获取各种时间数据的getYear()getHour()等gettergetter API的详细内容参考**SUNDBInterval**

Time with time zone与Timestamp with time zone类型

对于时间类型SUNDB不仅提供DateTimeTimestamp等标准SQL类型也提供Time with time zone与Timestamp with time zone类型

```
CREATE TABLE SAMPLE_TABLE ( C1 TIME WITH TIME ZONE,
                            C2 TIMESTAMP WITH TIME ZONE );
```

SUNDB JDBC为了插入Time with time zone与Timestamp with time zone类型的数据在SUNDBPreparedStatement CLASS提供setTimeTimeZone(int colIndex, Time time, Calendar timezone) method和setTimestampTimeZone(int colIndex, Timestamp time, Calendar timezone) method详细内容参考**setTimeTimeZone**与**setTimestampTimeZone**

当然也可以使用原有的setTime()与setTimestamp() method插入但此column的time zone值无条件被设置为JAVA执行环境的local time zone因此为了插入多种time zone信息应使用setTimeTimeZone()或setTimestampTimeZone

setTimeTimeZone与setTimestampTimeZone设置的TimeTimestamp数据可以与指定的time zone值同时插入

```
import csii.sundb.jdbc.SundbPreparedStatement;
```

```
PreparedStatement pstmt = con.prepareStatement(
    "INSERT INTO SAMPLE_TABLE VALUES (?,?,?,?)");
Calendar now = Calendar.getInstance();
Calendar usNow = Calendar.getInstance(TimeZone.getTimeZone("GMT-8"));

Time t = new Time(now.getTimeInMillis());
Timestamp ts = new Timestamp(now.getTimeInMillis());

pstmt.setTime(1, t);
pstmt.setTimestamp(2, ts);
pstmt.executeUpdate();

((SundbPreparedStatement)pstmt).setTimeTimeZone(1, t, now);
((SundbPreparedStatement)pstmt).setTimestampTimeZone(2, ts, now);
pstmt.executeUpdate();

((SundbPreparedStatement)pstmt).setTimeTimeZone(1, t, usNow);
((SundbPreparedStatement)pstmt).setTimestampTimeZone(2, ts, usNow);
pstmt.executeUpdate();
```

第1112行以time类型与timestamp类型绑定对应columnt与ts值各自传送到服务器在服务器中默认time zone值将插入到DB column

相反第1516行是以time with time zone类型与timestamp with time zone类型绑定对应columntime zone信息也同时传送到服务器即如果服务器与客户端的time zone设置相同则第

11~13行插入的行数据与第15~17行插入的行数据相同

第1920行插入GMT-8时区的time zone信息

查询time with time zone与timestamp with time zone数据的方法有两种从服务器上以char形式获取的方法与用JDBC获取对应类型的数据后以Time或Timestamp对象获取的方法

```
ResultSet rs = stmt.executeQuery("select c1, cast(c1 as char(33)), c2,
cast(c2 as char(33)) from sample_table");
while (rs.next())
{
    System.out.println("c1 = " + rs.getTime(1).toString());
    System.out.println("c1 as char = " + rs.getString(2));

    System.out.println("c2 = " + rs.getTimestamp(3).toString());
    System.out.println("c2 as char = " + rs.getString(4));
}
```

执行以上代码时其执行结果为如下

```
c1 = 12:35:26
c1 as char = 12:35:26.052000 +09:00
c2 = 2014-03-27 12:35:26.052
c2 as char = 2014-03-27 12:35:26.052000 +09:00
c1 = 12:35:26
c1 as char = 12:35:26.052000 +09:00
c2 = 2014-03-27 12:35:26.052
```

```
c2 as char = 2014-03-27 12:35:26.052000 +09:00

c1 = 12:35:26

c1 as char = 19:35:26.052000 -08:00

c2 = 2014-03-27 12:35:26.052

c2 as char = 2014-03-26 19:35:26.052000 -08:00
```

REF CURSOR类型

也可以在JDBC中使用PSM中用作PARAMETER的REF CURSOR

```
CREATE OR REPLACE PROCEDURE proc_cursor1( p1 OUT SYS_REFCURSOR )
AS
  var1 INTEGER;
BEGIN
  OPEN p1 FOR SELECT r_c1 FROM r;
  FETCH p1 INTO var1;
  DBMS_OUTPUT.PUT_LINE(var1);
END;
/
```

```
CREATE OR REPLACE PROCEDURE proc_cursor2( p1 IN SYS_REFCURSOR )
AS
  var1 INTEGER;
BEGIN
```

```
  LOOP
    FETCH p1 INTO var1;
    EXIT WHEN p1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(var1);
  END LOOP;
  CLOSE p1;
END;
/
```

如果想在JDBC中将REF CURSOR用作OUTPUT PARAMETER需要将sqlType类型设置为Types.REF_CURSOR (JDK 1.8以上) 或SundbTypes.REF_CURSOR (JDK 1.8以下)

```
import import java.sql.Types;


CallableStatement cstmt1 = con.prepareCall("{CALL proc_cursor1(?)}");
cstmt1.registerOutParameter(1, Types.REF_CURSOR);
cstmt1.execute();
```

```
import csii.sundb.jdbc.SundbTypes;


CallableStatement cstmt1 = con.prepareCall("{CALL proc_cursor1(?)}");
cstmt1.registerOutParameter(1, SundbTypes.REF_CURSOR);
cstmt1.execute();
```

执行CallableStatement后可以通过CallableStatement.getObject()接收ResultSet对象

```
ResultSet rs = (ResultSet)cstmt1.getObject(1);
if (!rs)
{
    if (rs.next())
    {
        System.out.println(rs.getInt(1));
    }
}
```

若想将接收为OUTPUT PARAMETER的REF CURSOR用作新statement的INPUT PARAMETER须将通过CallableStatement.getObject()接收的ResultSet对象设置为PreparedStatement.setObject()

```
CallableStatement cstmt2 = con.prepareCall("{CALL proc_cursor2(?)}");
cstmt2.setObject(1, rs);
cstmt2.execute();
```

> Note:
>
> 执行ResultSet.next()时按照通过Statement.setFetchSize()设置的值进行fetch操作将接收为OUTPUT PARAMETER的REF CURSOR用作新statement的INPUT PARAMETER时须设置使用OUTPUT PARAMETER的CallableStatement的fetch size

日志

日志的种类

使用JDBC开发项目时如果JDBC驱动程序可记录各种日志则会有很多帮助

SUNDB不仅在开发时在运营时也提供可记录有效信息的功能

日志的种类有trace logprotocol logquery log三种

每次调用JDBC API时Trace log均记录信息因此可以知道调用什么样的JDBC APIProtocol log显示JDBC驱动程序和SUNDB服务器之间的通信交互情况Query log记录执行PreparedStatement或Statement execute时执行的SQL语句

使用DriverManager的日志

首先可以通过DriverManager记录日志可在DriverManager使用setLogWriter() method决定日志的介质在 connection url中指定记录日志的类型

```
DriverManager.setLogWriter(new PrintWriter(System.out));
String url =
"jdbc:sundb://localhost:22581/test?trace_log=on&query_log=on";
Connection con = DriverManager.getConnection(url, "TEST", "test");
```

第1行定义日志介质所有日志输出到console第2行定义connection url可在?字符后面定义属性意味着使用 trace_logquery_log

如下这些参数可以在url指定也可以用Properties对象定义

```
Properties prop = new Properties();
prop.put("trace_log", "on");
prop.put("query_log", "on");
prop.put("protocol_log", "on");
prop.put("user", "TEST");
prop.put("password", test");
Connection con = DriverManager.getConnection(url, prop);
```

SUNDB中可使用的connection property定义在**连接属性**

偶尔会出现难以调用DriverManager的setLogWriter()的情况例如使用中间件时无法添加这样的代码为此SUNDB JDBC提供global_logger全局参数其他普通的参数仅适用于一个connection但此属性应用于全局即设置此参数时不需要调用DriverManager.setLogWriter()

```
String url = "jdbc:sundb://localhost:22581/test?" +
             "global_logger=console&trace_log=on&query_log=on";
Connection con = DriverManager.getConnection(url, "TEST", "test");
```

global_logger属性仅在第一次适用一次如果DriverManager中已设置log writer则忽略此属性此属性值仅支持当前console其他值的情况下不进行任何操作

使用DataSource的日志

与DriverManager不同DataSource可以按照各个对象单独设置log writer在一个DataSource对象中设置log writer后从该DataSource生成的所有connection均以该log writer记录日志DataSource中设置日志的方法如下

```
import csii.sundb.jdbc.SundbDataSource;

...

SundbDataSource ds = new csii.sundb.jdbc.SundbDataSource();

ds.setServerName("localhost");

ds.setDatabaseName("test");

ds.setUser("TEST");

ds.setPassword("test");

ds.setPortNumber(22581);

ds.setLogTarget("console");

ds.setTraceLog("on");

ds.setQueryLog("on");

ds.setProtocolLog("on");

Connection con = ds.getConnection();
```

如上可使用setLogTarget()setTraceLog()setQueryLog()setProtocolLog() method设置日志功能可以替代setLogTarget()调用DataSource的setLogWriter()但使用中间件时无法直接调用setLogWriter()等method因此需要用property控制此时如果定义logTargettraceLogqueryLogprotocolLog属性则中间件会自动调用该method

查看Plan Text

简单用法

类似使用SUNDB ODBC获取执行的Statement的plan text也可以使用SUNDB JDBC获取plan textSUNDBStatement CLASS中可以使用非标准API method设置为生成plan text也可以获取生成

的plan text

```
Connection con = ...

SundbStatement stmt = (SundbStatement)con.createStatement();

stmt.setExplainPlanOption(SundbStatement.EXPLAIN_PLAN_OPTION_ON);

ResultSet rs = stmt.executeQuery("select * from t1");

System.out.println(stmt.getExplainPlan());
```

例如查询简单的表时如下输出plan text

```
< Execution Plan >

================================================================================

|  IDX  |  NODE DESCRIPTION                                |       ROWS |

--------------------------------------------------------------------------------

|    0  |  SELECT STATEMENT                                |            |

|    1  |    TABLE ACCESS ("T1")                           |          0 |

================================================================================

     1  -  READ COLUMNS : A
```

选项的种类

SUNDB对生成plan text提供四种属性

- EXPLAIN_PLAN_OPTION_OFF : Statement的plan text相关属性的默认值不生成plan text
- EXPLAIN_PLAN_OPTION_ON：执行或fetch时生成plan text
- EXPLAIN_PLAN_OPTION_ON_VERBOSE：执行或fetch时生成执行时间等更加详细的plan text

- EXPLAIN_PLAN_OPTION_ONLY：执行或fetch时像EXPLAIN_PLAN_OPTION_ON一样生成plan text但实际不执行

以上属性可以使用SUNDBStatement.setExplainPlanOption(int) method设置只要设置一次则一直维持此属性属性的常数值定义在SundbStatement

> Note:
>
> 执行非SELECT语句的DML或其他语句时生成plan text但SELECT语句被fetch之后在服务器内部游标移到最后位置时生成plan text因此SELECT语句的情况需要用ResultSet扫描所有row后获取plan text但表的row少时即使没有fetch到最后一条记录服务器的游标也可以扫描到最后此时可以获取plan text

Connection Failover

SUNDB支持客户端级别(JDBC, ODBC)的connection failover通过JDBC或ODBC以client/server模式连接SUNDB的应用程序连接失败或连接断开时可自动连接至替代服务器由此可以使用多台服务器编写高可用的应用程序尤其在与Web服务器联动的环境下通过连接属性（connection property）登记主服务器url与替代服务器url后不用担心从connection pool获取的Connection对象被断开并可稳定使用failover全部在驱动内部自动进行用户不用考虑重连的逻辑

为了使用failover功能需赋予alternate_servers属性可以用此属性值赋予替代服务器的IP与port也可以使用逗号()指定多台替代服务器

```
Class.forName("csii.sundb.jdbc.SundbDriver");
```

```
String url = "jdbc:sundb://192.168.0.101:22581/test";
Property prop = new Properties();
prop.setProperty("user", "TEST");
prop.setProperty("password", "test");
prop.setProperty("alternate_servers", "192.168.0.201:22581");
Connection con = DriverManager.getConnection(url, prop); ❶
Statement stmt = con.createStatement();
stmt.executeUpdate("insert into time_tab values (sysdate)"); ❷
...
```

例如在❶连接主服务器(192.168.0.101)失败时自动连接替代服务器(192.168.0.201)后返回connection对象用户不会收到其他报错信息如果在❷executeUpdate()过程中连接断开或之前已断开而无法与服务器通信时在驱动程序内部自动连接替代服务器后采取措施使其可继续使用原有connection对象后产生SQLException用户处理SQLException后可继续使用原有的Connection对象

通过connection对象的getWarnings()可查看是否failover

SUNDB JDBC 的failover特点中的一个为可继续使用从Connection对象生成的Statement对象或PreparedStatement对象（ResultSet无法继续使用）

尤其是当PreparedStatement发生failover时内部在替代服务器重新执行prepare因此用户可继续使用此对象

```
Connection con = DriverManager.getConnection(url, prop);
PreparedStatement pstmt =
    con.prepareStatement("insert into t1 values (?,?)");
```

```
...
try
{
    pstmt.setInt(1, 100);
    pstmt.setString(2, "John");
    pstmt.executeUpdate();
}
catch (SQLException sException)
{
    if (sException.getErrorCode() == 21012)
    {
```

- 由于Communication link failure报错而产生failover
- 无法继续使用pstmt对象

```
pstmt.executeUpdate();
    }
    else
    {
        ...
    }
}
```

SUNDB JDBC驱动关于failover提供几种属性通过此属性可决定failover类型连接顺序及failover时使用的简单策略等

failover_type

通过此属性可选择failover类型上述示例代码❶中发生的failover叫做connection failover❷中发生的failover叫做session failover以此属性值设置connection时仅执行connection failover设置session时执行connection failover与session failover默认值为connection

failover_granularity

发生failover后从当前Connection对象生成的PreparedStatement对象将连接到替代服务器后重新执行prepare操作此时在替代服务器上prepare失败时（根据服务器环境不同可能报错）通过此属性决定是否将failover本身处理为失败还是忽略prepare失败当前此参数值可以设置为0和10表示忽略prepare错误继续执行failover1表示failover失败默认值为0设置为0后如果prepare失败则无法继续使用PreparedStatement对象用户需要直接生成PreparedStatement对象

Ditect Attach模式连接

从SUNDB JDBC 1.1开始除原有的基于TCP/IP的Client/ Server模式（C/S模式）外提供Direct Attach模式（D/A模式）连接此连接模式与ODBC D/A模式相同直接与服务器进程联动并在一个进程内（jvm等进程中）联动服务器模块后运行因此远程主机无法以D/A模式连接

D/A模式是为了提高SUNDB的快速处理性能而研究出的特殊连接方式使用基于TCP/IP的连接时SUNDB的快速事务处理性能需要昂贵的网络费用因此要求快速处理的情况需要其他对策虽然存在应在与SUNDB服务器相同的主机上执行的限制但需要进行快速处理时以D/A模式连接是很好的对策

通过D/A模式连接的JDBC应用程序在jvm内初次生成DB connection时加载SUNDB jni库并可以直

接调用服务器的功能在没有网络费用的情况下可以从jvm通过native interface直接调用服务器模块因此处理速度比原有的CA模式更快

连接方法

为了以D/A模式访问要在connection url使用0.0.0.0:0代替原ip:port可继续使用原有的url方式通过0.0.0.0:0的特殊ip:port地址可以使原应用程序的修改达到最小化或者可以在url使用da的特殊协议以D/A模式连接

```
Connection con =
 DriverManager.getConnection("jdbc:sundb://0.0.0.0:0/test","TEST","test");
```

或者

```
Connection con =
 DriverManager.getConnection("jdbc:sundb:da/test", "TEST", "test");
```

D/A连接的特征

以D/A模式访问可以在jvm中直接调用SUNDB server模块JDBC程序与SUNDB server模块之间通过Java Native Interface (JIN)进行调用考虑到调用JNI昂贵的费用通过最少的调用费用使用服务器模块因此其性能比原有的基于TCP/IP的JDBC快两倍以上

使用原有的基于TCP/IP的连接时仅使用sundb6.jar文件但以D/A模式连接时需要动态加载使用$SUNDB_HOME/lib中的libsundbjni.so和libsundbas.so文件因此如果没有两个库文件连接时会报错但启动java程序时不需要单独指定库文件的位置因为从与sundb6.jar文件相同的目录查找这两个库文件并加载

Global Connection

集群环境中提供应用程序可选择符合查询处理的节点并执行的global connection功能

设置

使用global connection需要与locality_aware_transaction属性值同时设置locator_file或locator_hostlocator_port使用global session功能需要将use_global_session属性值设置为1

```
Properties prop = new Properties();
prop.put("locality_aware_transaction", "1");
prop.put("locator_file", "/home/sundb/.location.ini");
prop.put("user", "TEST");
prop.put("password", test");
Connection con = DriverManager.getConnection(url, prop);
```

```
String url = "jdbc:sundb://192.168.0.1:22581/test?" +

"locality_aware_transaction=1&locator_host=192.168.0.2&locator_port=42581"
;
Connection con = DriverManager.getConnection(url, "TEST", "test");
```

处理过程

1. 访问数据库

连接用户输入的服务器并获取集群系统的信息后通过locator file或glocator部署集群系统信息访问集群胸痛的所有节点

2. 执行statement

未部署集群系统信息时通过locator file或glocator部署集群系统信息并访问集群系统的所有节点

在集群增加节点后访问新节点时在该节点同样生成其他节点的所有statement后并准备好执行SQL

statement class的情况没有sharding key因此根据locality_group_policy或locality_member_policy属性选择节点并执行statement

PreparedStatement或CallableStatement class的情况未部署sharding key信息时从任意服务器部署该SQL的sharding key信息部署了sharding key信息时使用sharding key信息或locality_group_policylocality_member_policy选择合适的节点并执行查询

所选节点发生故障时除该节点外重新选择合适的节点并执行查询

执行statement后sharding信息发生变化时删除部署的sharding key信息

执行statement后增加/删除集群节点等集群系统信息发生变更时删除部署的集群系统信息

3. ResultSet数据接收

从执行SQL的节点获取数据

4. 关闭ResultSet

在执行SQL饿节点关闭游标

5. 关闭Statement

在连接的所有节点释放statement

6. 关闭数据库

释放与所有节点的连接

为高可用性的异常处理

使用global connection时运行中如果在所选节点发生故障则根据事务发生及SELECT进行与否如下操作

- 没有事务时

没事事务时如果所选节点发生故障则在JDBC内部选择其他节点并执行对应查询所选节点发送故障但在其他节点正常执行因此不向用户报错

- 有事务或SELECT进行时

事务发生或ResultSet.next()进行时如果所选节点发生故障则JDBC无法再进行当前操作因此报21047(Retry the transactional operations again) 错误发生21047错误时用户要重新执行该事务或重新执行SELECT

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO T1 VALUES
(?)");


boolean retry;
```

```
do
{
    retry = false;

    try
    {
        pstmt.setInt(1, 1);
        pstmt.executeUpdate();
    }
    catch (SQLException e)
    {
        if (e.getErrorCode() == 21047)
        {
            retry = true;
        }
        else
        {
            throw e;
        }
    }
} while (retry == true);
```

```
PreparedStatement pstmt = con.prepareStatement("SELECT * FROM T1 WHERE C1
```

```
= ?");


boolean retry;


do
{
    retry = false;


    try
    {
        pstmt.setInt(1, 1);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next())
        {
            ...
        }
        rs.close();
    }
    catch (SQLException e)
    {
        if (e.getErrorCode() == 21047)
        {
            retry = true;
        }
        else
```

```
        {
            throw e;
        }
    }
} while (retry == true);
```

- COMMIT或ROLLBACK时

COMMIT事务时如果所选节点发生故障则JDBC通过其他节点查看事务是否在所选节点发生故障之前COMMIT如果所选节点发生故障但事务正常COMMIT时不报错在事务未COMMIT的情况下所选节点发生故障时报21047(Retry the transactional operations again)错发生21047错误时用户需要重新执行该事务

ROLLBACK事务时如果所选节点发生故障则JDBC不报错因为由于节点故障该事务已经被ROLLBACK

```
con.setAutoCommit(false);

PreparedStatement pstmt = con.prepareStatement("INSERT INTO T1 VALUES
(?)");

boolean retry;

do
{
    retry = false;
```

```
    try
    {
        pstmt.setInt(1, 1);
        pstmt.executeUpdate();
        con.commit();
    }
    catch (SQLException e)
    {
        if (e.getErrorCode() == 21047)
        {
            retry = true;
        }
        else
        {
            throw e;
        }
    }
} while (retry == true);
```

约束事项

- 只有PreparedStatement和CallableStatement class查找适合SQL的节点

由于Statement class中没有查找适合节点的信息因此根据locality_group_policy和locality_member_policy属性选择节点

- COMMIT或ROLLBACK事务时使用Connection.commit(). Connection.rollback()

使用GLOBAL CONNECTION时通过SQL语句执行COMMIT或ROLLBACK则无法检测到事务状态变化COMMIT或ROLLBACK事务时必须使用Connection.commit(). Connection.rollback()

- Global会话功能在SQL语句中不支持Data Definition Language (DDL)

Statement Pooling

SUNDB支持statement pooling功能Statement pooling是对重复语句或重复调用的method等重复使用的statement进行pooling提高性能的功能在JDBC 3.0定义statement pooling接口

应用程序使用statement pool对与特定connection相关的statement进行pooling各个connection对象拥有自己的poolSundbConnection中包含激活statement pool的method使用statement pool时statement对象在调用close method时pooling

说明

激活statement pool并调用statement对象的colse method时SUNDB JDBC驱动自动pooling statement, PreparedStatement, CallableStatementPreparedStatement, CallableStatement对象将SQL字符串用作key值进行poolingJDBC驱动在生成PreparedStatement或CallableStatement时在pool自动对比/检索其

对比标准为如下

- SQL字符串应相同
- Statement类型应相同

- Result set的属性应相同

Statement对象的SQL字符串会发生变化因此为了pooling不使用SQL字符串而是在JDBC驱动自动处理

在pool检索的过程中如果发现一致的statement则返回其如果未发现则生成新的statement两种情况均调用对象的close method时statement和游标以及状态被pooling但statement是closed的状态如果要重新使用closed状态的statement则SQL字符串和statement类型result set属性均应生成相同的statement

检索到pooling的PreparedStatement和CallableStatement时状态和数据信息自动初始化后重新设置为默认值Statement pool满时通过LRU算法在pool被清除存储在statement pool的statement需要调用connection对象的close method才能整理

使用

激活Statement Pool

使用connection对象的statement pool要设置STATEMENT_POOL_ON和STATEMENT_POOL_SIZE参数即使激活STATEMENT_POOL_ON参数STATEMENT_POOL_SIZE的默认值为0因此必须使用大于0的数字

可以在properties对象添加STATEMENT_POOL_ON和STATEMENT_POOL_SIZE参数激活statement pool功能另外也可以通过SundbDataSource API和SundbConnection API激活statement pool

通过SundbDataSource激活

使用SundbDataSource class所有connection对象会拥有具备相同STATEMENT_POOL_SIZE的

statement pool

- 调用setStatementPoolOn(true) method
- 调用setStatementPoolsize method

```
SundbDataSource sDataSource = new SundbDataSource();


sDataSource.setStatementPoolOn( true );
sDataSource.setStatementPoolSize( 10 );
```

可如下查看参数设定值

```
System.out.println("Statement Pool on:" +
sDataSource.getStatementPoolOn());
System.out.println("Statement Pool size:" +
sDataSource.getStatementPoolSize());
```

通过SundbConnection激活

使用SundbConnection class可与其他对象分开单独激活statement pool

- 调用setStatementPoolOn(true) method
- 调用setStatementpoolSize

```
SundbConnection sCon = sDataSource.getConnection();


sCon.setStatementPoolOn( true );
```

```
sCon.setStatementPoolSize( 10 );
```

可如下查看参数设定值

```
System.out.println("Statement Pool on:" + sCon.getStatementPoolOn());
System.out.println("Statement Pool size:" + sCon.getStatementPoolSize());
```

禁用Statement Pool

可禁用激活的statement pool在激活的状态下变更为禁用状态时存储在statement pool的statement会被清楚并close

使用setStatementPoolOn method禁用

```
sCon.setStatementPoolOn(false);
```

使用setStatementPoolSize method禁用

```
sCon.setStatementPoolSize(0);
```

生成statement

在激活statement pool的状态下生成statement, PreparedStatement, CallableStatement的方式与普通的生成方式相同

以下为生成新statement对象的代码

```
PreparedStatement sPstmt = sCon.prepareStatement( "INSERT INTO EMP VALUES
( ?, ? )" );
```

禁用特定statement

激活statement pool时SUNDB JDBC驱动自动pooling所有statement使用setPoolable method时可在pooling排除特定statement

以下为通过 isPoolable method和 setPoolable method查看是否pooling的示例

```
PreparedStatement sPstmt = sCon.prepareStatement( "SELECT 1 FROM DUAL" );
System.out.println( "Is poolable: " + sPstmt.isPoolable() );
sPstmt.setPoolable( false );
System.out.println( "Is poolable: " + sPstmt.isPoolable() );
```

3.2 JDBC API References

Array

未实现class

free

```
void free() throws SQLException
```

getArray

```
Object getArray() throws SQLException
```

```
Object getArray(Map<String,Class<?>> map) throws SQLException
```

```
Object getArray(long index, int count) throws SQLException
```

```
Object getArray(long index, int count, Map<String,Class<?>> map) throws
SQLException
```

getBaseType

```
int getBaseType() throws SQLException
```

getBaseTypeName

```
String getBaseTypeName() throws SQLException
```

getResultSet

```
ResultSet getResultSet() throws SQLException
```

```
ResultSet getResultSet(Map<String,Class<?>> map) throws SQLException
```

```
ResultSet getResultSet(long index, int count) throws SQLException
```

```
ResultSet getResultSet(long index, int count, Map<String,Class<?>> map)
throws SQLException
```

Blob

free

```
void free() throws SQLException
```

- 操作：释放此对象占用的资源
- 例外：无

getBinaryStream

```
InputStream getBinaryStream() throws SQLException
```

- 操作：Blob对象的值作为InputStream类型返回
- 例外：Blob已经free的情况下将发生SQLException

```
InputStream getBinaryStream(long pos, long length) throws SQLException
```

- 操作：从指定为pos的字节开始将长度为length的值作为包含blob对象值的InputStream类型返回
- 例外：Blob已经free的情况下pos比1小或者比blob对象字节长度更大或者pos + length比blob对象字节的长度更大时将发生SQLException

getBytes

```
byte[] getBytes(long pos, int length) throws SQLException
```

- 操作：从指定为pos的字节开始将长度为length的值作为包含blob对象值的byte数组返回
- 例外：Blob已经free的情况下pos比1小或者length的长度比0小时将发生SQLException

length

```
long length() throws SQLException
```

- 操作：返回Blob对象的字节长度
- 例外：Blob已经free的情况下将发生SQLException

position

```
long position(byte[] pattern, long start) throws SQLException
```

- 操作：将Blob对象值里指定的byte数组pattern开始的字节位置进行返回Pattern检索从start位置开始Pattern如不能在blob对象内检索到那么返回-1.
- 例外：Blob已经free的情况下start比1小时将发生SQLException

```
long position(Blob pattern, long start) throws SQLException
```

- 操作：将Blob对象值里指定的Blob对象pattern开始的字节位置进行返回Pattern检索从start位置开始Pattern如不能在blob对象内检索到那么返回-1.
- 例外：Blob已经free的情况下start比1小时将发生SQLException

setBinaryStream

```
OutputStream setBinaryStream(long pos) throws SQLException
```

- 操作：暂未实现
- 例外：将一直发生SQLFeatureNotSupportedException

setBytes

```
int setBytes(long pos, byte[] bytes) throws SQLException
```

- 操作：Blob对象值中给定的pos位置开始记录byte数组bytes并返回记录的bytes的长度Pos位置已经存在值的话字节序列将覆盖在使用byte数组期间如果达到blob值的末端那么则会增加blob值的长度使得可以容纳额外的字节
- 例外：Blob已经free的情况下pos比1小时将发生SQLException

```
int setBytes(long pos, byte[] bytes, int offset, int len) throws
SQLException
```

- 操作：Blob对象值的pos位置开始记录长度为len的从offset位置开始的byte数组bytes并返回记录的bytes的长度Pos位置已经存在值的话byte数组将覆盖在使用byte数组期间如果达到blob值的末端那么则会增加blob值的长度使得可以容纳额外的字节
- 例外：Blob已经free的情况下pos比1小或者offset比0小或者offset + len比bytes的长度大时将发生SQLException

truncate

```
void truncate(long len) throws SQLException
```

- 操作：Blob对象的长度按照len大小进行裁切
- 例外：Blob已经free的情况下len比0小时将发生SQLException

CallableStatement

getArray

```
Array getArray(int parameterIndex) throws SQLException
```

- 操作：不支持array类型
- 例外：总是返回SQLFeatureNotSupportedException

```
Array getArray(String parameterName) throws SQLException
```

- 操作：不支持array类型
- 例外：总是返回SQLFeatureNotSupportedException

getBigDecimal

```
BigDecimal getBigDecimal(int parameterIndex) throws SQLException
```

- 操作：以BigDecimal类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
BigDecimal getBigDecimal(int parameterIndex, int scale) throws
SQLException
```

- 操作：未实现（deprecated的method）

- 例外：总是产生SQLFeatureNotSupportedException

```
BigDecimal getBigDecimal(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getBlob

```
Blob getBlob(int parameterIndex) throws SQLException
```

- 操作：第parameterIndex个column的数据获取为blob类型
- 例外：当parameterIndex无效或者不能访问database或CallableStatement已经关闭时将发生SQLException

```
Blob getBlob(String parameterName) throws SQLException
```

- 操作：暂未实现
- 例外：总是返回SQLFeatureNotSupportedException

getBoolean

```
boolean getBoolean(int parameterIndex) throws SQLException
```

- 操作：以boolean类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生

SQLException

```
boolean getBoolean(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getByte

```
byte getByte(int parameterIndex) throws SQLException
```

- 操作：以byte类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
byte getByte(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getBytes

```
byte[] getBytes(int parameterIndex) throws SQLException
```

- 操作：以byte[]类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**

- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
byte[] getBytes(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getCharacterStream

```
Reader getCharacterStream(int parameterIndex) throws SQLException
```

- 操作：以reader类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
Reader getCharacterStream(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getClob

```
Clob getClob(int parameterIndex) throws SQLException
```

- 操作：第parameterIndex个column的数据获取为clob类型

- 例外：parameterIndex无效或者不能连接database或者CallableStatement已关闭时将发生SQLException

```
Clob getClob(String parameterName) throws SQLException
```

- 操作：暂未实现
- 例外：总是返回SQLFeatureNotSupportedException

getDate

```
Date getDate(int parameterIndex) throws SQLException
```

- 操作：以data类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**创建Date对象时使用local timezone和locale
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
Date getDate(int parameterIndex, Calendar cal) throws SQLException
```

- 操作：以data类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**创建Date对象时使用cal的timezone和locale
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
Date getDate(String parameterName) throws SQLException
```

- 操作：未实现

- 例外：总是返回SQLFeatureNotSupportedException

```
Date getDate(String parameterName, Calendar cal) throws SQLException
```

- 操作：未实现
- 例外：总是返回SQLFeatureNotSupportedException

getDouble

```
double getDouble(int parameterIndex) throws SQLException
```

- 操作：以double类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
double getDouble(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是返回SQLFeatureNotSupportedException

getFloat

```
float getFloat(int parameterIndex) throws SQLException
```

- 操作：以float类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**

- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
float getFloat(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是返回SQLFeatureNotSupportedException

getInt

```
int getInt(int parameterIndex) throws SQLException
```

- 操作：以int类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
int getInt(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是返回SQLFeatureNotSupportedException

getLong

```
long getLong(int parameterIndex) throws SQLException
```

- 操作：以long类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对**

SUNDB类型的getter method支持与否-1

- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
long getLong(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是返回SQLFeatureNotSupportedException

getNCharacterStream

```
Reader getNCharacterStream(int parameterIndex) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 例外：总是产生SQLFeatureNotSupportedException

```
Reader getNCharacterStream(String parameterName) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 例外：总是产生SQLFeatureNotSupportedException

getNClob

```
NClob getNClob(int parameterIndex) throws SQLException
```

- 操作：目前不支持NClob系列类型
- 例外：总是产生SQLFeatureNotSupportedException

```
NClob getNClob(String parameterName) throws SQLException
```

- 操作：目前不支持NClob系列类型
- 例外：总是产生SQLFeatureNotSupportedException

getNString

```
String getNString(int parameterIndex) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 例外：总是产生SQLFeatureNotSupportedException

```
String getNString(String parameterName) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 例外：总是产生SQLFeatureNotSupportedException

getObject

```
Object getObject(int parameterIndex) throws SQLException
```

- 操作：以Java对象类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
Object getObject(int parameterIndex, Map<String,Class<?>> map) throws
```

```
SQLException
```

- 操作：不支持
- 例外：总是产生SQLFeatureNotSupportedException

```
Object getObject(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
Object getObject(String parameterName, Map<String,Class<?>> map) throws
SQLException
```

- 操作：不支持
- 例外：总是产生SQLFeatureNotSupportedException

getRef

```
Ref getRef(int parameterIndex) throws SQLException
```

- 操作：不支持Ref类型
- 例外：总是产生SQLFeatureNotSupportedException

```
Ref getRef(String parameterName) throws SQLException
```

- 操作：不支持Ref类型
- 例外：总是产生SQLFeatureNotSupportedException

getRowId

```
RowId getRowId(int parameterIndex) throws SQLException
```

- 操作：以rowId类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
RowId getRowId(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getShort

```
short getShort(int parameterIndex) throws SQLException
```

- 操作：以shot类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
short getShort(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getSQLXML

```
SQLXML getSQLXML(int parameterIndex) throws SQLException
```

- 操作：不支持SQLXML类型
- 例外：总是产生SQLFeatureNotSupportedException

```
SQLXML getSQLXML(String parameterName) throws SQLException
```

- 操作：不支持SQLXML类型
- 例外：总是产生SQLFeatureNotSupportedException

getString

```
String getString(int parameterIndex) throws SQLException
```

- 操作：以String类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考 **对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
String getString(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getTime

```
Time getTime(int parameterIndex) throws SQLException
```

- 操作：以time类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
Time getTime(int parameterIndex, Calendar cal) throws SQLException
```

- 操作：以time类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**生成Time对象时使用cal的timezone
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
Time getTime(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
Time getTime(String parameterName, Calendar cal) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getTimestamp

```
SQLException
```

- 操作：以timestamp类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**创建Timestamp对象时使用local timezone
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
Timestamp getTimestamp(int parameterIndex, Calendar cal) throws
SQLException
```

- 操作：以timestamp类型获取第parameterIndex个column的数据各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**创建Timestamp对象时使用cal的timezone
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException

```
Timestamp getTimestamp(String parameterName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
Timestamp getTimestamp(String parameterName, Calendar cal) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

getURL

```
URL getURL(int parameterIndex) throws SQLException
```

- 操作：不支持URL类型
- 例外：总是产生SQLFeatureNotSupportedException

```
URL getURL(String parameterName) throws SQLException
```

- 操作：不支持URL类型
- 例外：总是产生SQLFeatureNotSupportedException

registerOutParameter

```
void registerOutParameter(int parameterIndex, int sqlType) throws
SQLException
```

- 操作：以sqlType登记parameterIndex位置的OUT参数所有Out参数应在执行stored procedure之前注册指定为sqlType的OUT参数的JDBC类型为了读取参数值决定用于get method的JAVA类型各SUNDB类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 例外：parameterIndex无效或无法访问database或CallableStatement已关闭时产生SQLException对SUNDB不支持的类型产生SQLFeatureNotSupprtedException

```
void registerOutParameter(int parameterIndex, int sqlType, int scale)
throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void registerOutParameter(int parameterIndex, int sqlType, String
typeName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void registerOutParameter(String parameterName, int sqlType) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void registerOutParameter(String parameterName, int sqlType, int scale)
throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void registerOutParameter(String parameterName, int sqlType, String
typeName) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setAsciiStream

```
void setAsciiStream(String parameterName, InputStream x) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setAsciiStream(String parameterName, InputStream x, int length)
throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setAsciiStream(String parameterName, InputStream x, long length)
throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setBigDecimal

```
void setBigDecimal(String parameterName, BigDecimal x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setBinaryStream

```
void setBinaryStream(String parameterName, InputStream x) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setBinaryStream(String parameterName, InputStream x, int length)
throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setBinaryStream(String parameterName, InputStream x, long length)
throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setBlob

```
void setBlob(String parameterName, Blob x) throws SQLException
```

- 操作：不支持Blob类型
- 例外：总是产生SQLFeatureNotSupportedException

```
void setBlob(String parameterName, InputStream inputStream) throws
SQLException
```

- 操作：不支持Blob类型
- 例外：总是产生SQLFeatureNotSupportedException

```
void setBlob(String parameterName, InputStream inputStream, long length)
throws SQLException
```

- 操作：不支持Blob类型
- 例外：总是产生SQLFeatureNotSupportedException

setBoolean

```
void setBoolean(String parameterName, boolean x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setByte

```
void setByte(String parameterName, byte x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setBytes

```
void setBytes(String parameterName, byte[] x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setCharacterStream

```
void setCharacterStream(String parameterName, Reader reader) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setCharacterStream(String parameterName, Reader reader, int length)
throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setCharacterStream(String parameterName, Reader reader, long length)
throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setClob

```
void setClob(String parameterName, Clob x) throws SQLException
```

- 操作：不支持Clob类型
- 例外：总是产生SQLFeatureNotSupportedException

```
void setClob(String parameterName, Reader reader) throws SQLException
```

- 操作：不支持Clob类型
- 例外：总是产生SQLFeatureNotSupportedException

```
void setClob(String parameterName, Reader reader, long length) throws
SQLException
```

- 操作：不支持Clob类型
- 例外：总是产生SQLFeatureNotSupportedException

setDate

```
void setDate(String parameterName, Date x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setDate(String parameterName, Date x, Calendar cal) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setDouble

```
void setDouble(String parameterName, double x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setFloat

```
void setFloat(String parameterName, float x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setInt

```
void setInt(String parameterName, int x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setLong

```
void setLong(String parameterName, long x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setNCharacterStream

```
void setNCharacterStream(String parameterName, Reader value) throws
SQLException
```

- 操作：不支持NChar
- 例外：总是产生SQLFeatureNotSupportedException

```
void setNCharacterStream(String parameterName, Reader value, long length)
throws SQLException
```

- 操作：不支持NChar
- 例外：总是产生SQLFeatureNotSupportedException

setNClob

```
void setNClob(String parameterName, NClob value) throws SQLException
```

- 操作：不支持NClob
- 例外：总是产生SQLFeatureNotSupportedException

```
void setNClob(String parameterName, Reader reader) throws SQLException
```

- 操作：不支持NClob

- 例外：总是产生SQLFeatureNotSupportedException

```
void setNClob(String parameterName, Reader reader, long length) throws
SQLException
```

- 操作：不支持NClob
- 例外：总是产生SQLFeatureNotSupportedException

setNString

```
void setNString(String parameterName, String value) throws SQLException
```

- 操作：不支持NChar
- 例外：总是产生SQLFeatureNotSupportedException

setNull

```
void setNull(String parameterName, int sqlType) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setNull(String parameterName, int sqlType, String typeName) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setObject

```
void setObject(String parameterName, Object x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setObject(String parameterName, Object x, int targetSqlType) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setObject(String parameterName, Object x, int targetSqlType, int
scale) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setRowId

```
void setRowId(String parameterName, RowId x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setShort

```
void setShort(String parameterName, short x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setSQLXML

```
void setSQLXML(String parameterName, SQLXML xmlObject) throws SQLException
```

- 操作：不支持SQLXML类型
- 例外：总是产生SQLFeatureNotSupportedException

setString

```
void setString(String parameterName, String x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setTime

```
void setTime(String parameterName, Time x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setTime(String parameterName, Time x, Calendar cal) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setTimestamp

```
void setTimestamp(String parameterName, Timestamp x) throws SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

```
void setTimestamp(String parameterName, Timestamp x, Calendar cal) throws
SQLException
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

setURL

```
void setURL(String parameterName, URL val) throws SQLException
```

- 操作：不支持URL类型
- 例外：总是产生SQLFeatureNotSupportedException

wasNull

```
boolean wasNull()
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

isWrapperFor

```
boolean isWrapperFor(Class<?> iface)
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

unwrap

```
boolean isWrapperFor(Class<?> iface)
```

- 操作：未实现
- 例外：总是产生SQLFeatureNotSupportedException

Clob

free

```
void free() throws SQLException
```

- 操作：解除该对象占用的资源
- 例外：无

getAsciiStream

```
InputStream getAsciiStream() throws SQLException
```

- 操作：Clob对象的值作为InputStream类型返回
- 例外：Clob已经free的情况下将发生SQLException

getCharacterStream

```
Reader getCharacterStream() throws SQLException
```

- 操作：Clob对象的值作为reader类型返回
- 例外：Clob已经free的情况下将发生SQLException

```
Reader getCharacterStream(long pos, long length) throws SQLException
```

- 操作：从指定为pos的字节开始将长度为length的值作为包含clob对象值的reader类型返回
- 例外：Clob已经free的情况下pos比1小或者length比1小或者pos + len比 clob对象字节的长

度更大时将发生SQLException

getSubString

```
String getSubString(long pos, int length) throws SQLException
```

- 操作：从指定为pos的字节开始将长度为length的值作为包含clob对象值的string返回
- 例外：Clob已经free的情况下pos比1小或者length的长度比0小时将发生SQLException

length

```
long length() throws SQLException
```

- 操作：返回Clob对象的字节长度
- 例外：Clob已经free的情况下将发生SQLException

position

```
long position(Clob searchstr, long start) throws SQLException
```

- 操作：将clob对象值里指定的clob对象searchstr开始的位置进行返回Searchstr检索从start位置开始Searchstr如不能在clob对象内检索到那么返回-1.
- 例外：Clob已经free的情况下start比1小时将发生SQLException

```
long position(String searchstr, long start) throws SQLException
```

- 操作：将clob对象值里指定的string对象searchstr开始的位置进行返回Searchstr检索从start

位置开始Searchstr如不能在clob对象内检索到那么返回-1.

- 例外：Clob已经free的情况下start比1小时将发生SQLException

setAsciiStream

```
OutputStream setAsciiStream(long pos) throws SQLException
```

- 操作：暂未实现
- 例外：将一直发生SQLFeatureNotSupportedException

setCharacterStream

```
Writer setCharacterStream(long pos) throws SQLException
```

- 操作：暂未实现
- 例外：将一直发生SQLFeatureNotSupportedException

setString

```
int setString(long pos, String str) throws SQLException
```

- 操作：Clob对象值中给定的pos位置开始记录str并返回记录的长度Pos位置已经存在值的话将被覆盖在使用String类型期间如果达到clob值的末端那么则会增加clob值的长度使得可以容纳额外的字节
- 例外：Clob已经free的情况下pos比1小时将发生SQLException

```
int setString(long pos, String str, int offset, int len) throws
```

```
SQLException
```

- 操作：Clob对象值的pos位置开始记录长度为len的从offset位置开始的给定的string对象str并返回记录的长度Pos位置已经存在值的话将被覆盖在使用string类型期间如果达到clob值的末端那么则会增加clob值的长度使得可以容纳额外的字节
- 例外：Clob已经free的情况下pos比1小或者offset + len比str的长度大时将发生SQLException

truncate

```
void truncate(long len) throws SQLException
```

- 操作：Clob对象的长度按照len大小进行裁切
- 例外：Clob已经free的情况下len比0小或者clob的大小比len小时将发生SQLException

CommonDataSource

getLoginTimeout

```
int getLoginTimeout() throws SQLException
```

- 操作：返回设置的login timeout值服务器与socket连接时login timeout用作timeout值未设置则返回00意味着无限等待
- 异常：不发生

getLogWriter

```
PrintWriter getLogWriter() throws SQLException
```

- 操作：返回此DataSource设置的log writer未设置则返回nulllog writer为记录各种trace log的PrintWriter关于日志的详细内容参考日志
- 异常：不发生

setLoginTimeout

```
void setLoginTimeout(int seconds) throws SQLException
```

- 操作：设置Login timeout值服务器与socket连接时Login timeout用作timeout值0意味着无限等待
- 异常：不发生

setLogWriter

```
void setLogWriter(PrintWriter out) throws SQLException
```

- 操作：在此DataSource中设置log writer如不设置则默认值为nulllog writer为记录各种trace log的PrintWriter设置此值时根据选项记录trace logquery logprotocol log从此数据源生成的连接对象以及随之生成的所有StatementResultSet等对象执行日志trace logquery logprotocol log选项可以在connection url或property中定义详细内容参考日志
- 异常：不发生

setDataSourceName

```
void setDataSourceName(String aDataSourceName)
```

- 操作：设置data source名称不是连接所需的信息只是为了区别对象而可附加的信息
- 异常：不发生

setServerName

```
void setServerName(String aServerName)
```

- 操作：设置服务器名称即连接URL是连接所需的必要信息
- 异常：不发生

setDatabaseName

```
void setDatabaseName(String aDBName)
```

- 操作：设置数据库名称是连接所需的必要信息
- 异常：不发生

setNetworkProtocol

```
void setNetworkProtocol(String aProtocol)
```

- 操作：网络通信协议信息非连接所需的信息
- 异常：不发生

setUser

```
void setUser(String aUser)
```

- 操作：设置连接账号名称是连接所需的必要信息
- 异常：不发生

setPassword

```
void setPassword(String aPassword)
```

- 操作：设置连接账号密码是连接所需的必要信息
- 异常：不发生

setPortNumber

```
void setPortNumber(int aPort)
```

- 操作：设置连接服务器时使用的端口号是连接所需的必要信息
- 异常：不发生

```
void setPortNumber(String aPort)
```

- 操作：设置连接服务器时使用的端口号是连接所需的必要信息
- 异常：不发生

setRoleName

```
void setRoleName(String aRoleName)
```

- 操作：定义连接服务器时的role非连接所需的必要信息但是连接相关信息设置为""或"admin"或"sysdba" 中的一个
- 异常：不发生

setDescription

```
void setDescription(String aDescription)
```

- 操作：设置此数据源的说明不用于连接
- 异常：不发生

setConnectionProperties

```
void setConnectionProperties(Properties aProps)
```

- 操作：定义可用于各种连接的属性
- 异常：不发生

setURL

```
void setURL(String aURL) throws SQLException
```

- 操作：以URL形式指定serverNameportNumberdatabaseName与通过DriverManager连接时使用同样的URL
- 异常：使用错误的形式时发生SQLException

```
void setUrl(String aUrl) throws SQLException
```

与setURL(String aURL)相同

setLogTarget

```
void setLogTarget(String aTarget)
```

- 操作：无法调用setLogWriter时可通过此method设置log writer目前仅在aTarget为"console"时支持忽略其他值当设置为"console"时从此数据源生成的所有连接对象以下的日志输出到console
- 异常：不发生

setTraceLog

```
void setTraceLog(String aMode)
```

- 操作：设置trace logaMode为on时打开trace logging
- 异常：不发生

setQueryLog

```
void setQueryLog(String aMode)
```

- 操作：设置query logaMode为on时打开query logging
- 异常：不发生

setProtocolLog

```
void setProtocolLog(String aMode)
```

- 操作：设置protocol logaMode为on时打开protocol logging
- 异常：不发生

Connection

abort

```
void abort(Executor executor) throws SQLException
```

- 操作：结束connection对象Connection对象在物理连接上被关闭变为closed状态Connection对象使用的资源将被返还当前访问连接的所有线程将正常中止或发生SQLException
- 异常：当connection关闭或executor为null时将发生SQLExcpetion当存在Security Manager并且checkPermission方法拒绝嗲用abort时将发生SecurityException
- Since: 1.7

clearWarnings

```
void clearWarnings() throws SQLException
```

- 操作：删除当前连接对象的warning对象
- 异常：不发生

close

```
void close() throws SQLException
```

- 操作：为了不再使用当前连接断开与SUNDB的连接关闭此对象创建的所有Statement对象如果已关闭则不做任何操作
- 异常：从服务器发生错误或无响应时可能发生异常

commit

```
void commit() throws SQLException
```

- 操作：Non-auto commit模式时对当前连接执行commit
- 异常：已关闭或auto commit模式时发生SQLException

createArrayOf

```
Array createArrayOf(String typeName, Object[] elements) throws
SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

createBlob

```
Blob createBlob() throws SQLException
```

- 操作：创建一个实现 Blob 接口的对象 最初返回的对象中没有数据 可以使用 Blob 接口的 setBytes 方法向 Blob 对象添加数据
- 异常：如已close将发生发生SQLException

createClob

```
Clob createClob() throws SQLException
```

- 操作：创建一个实现 Clob 接口的对象 最初返回的对象中没有数据 可以使用 Clob 接口的 setString 方法向 Clob 对象添加数据
- 异常：如已close将发生发生SQLException

createNClob

```
NClob createNClob() throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

createSQLXML

```
SQLXML createSQLXML() throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

createStatement

```
Statement createStatement() throws SQLException
```

- 操作：生成Statement对象此Statement生成的ResultSet类型为ResultSet.TYPE_FORWARD_ONLYconcurrency为ResultSet.CONCUR_READ_ONLY, holdability为ResultSet.HOLD_CURSORS_OVER_COMMIT
- 异常：已关闭时发生SQLException

```
Statement createStatement(int resultSetType, int resultSetConcurrency)
throws SQLException
```

- 操作：创建生成拥有用户指定的resultset type与resultset concurrency的ResultSet的Statement对象此Statement对象生成的ResultSet的holdability与Connection的holdability类型相同Connection的默认holdability值为ResultSet.HOLD_CURSORS_OVER_COMMIT
- 异常：已关闭时发生SQLException

```
Statement createStatement(int resultSetType, int resultSetConcurrency, int
resultSetHoldability) throws SQLException
```

- 操作：创建生成拥有用户指定的resultset typeresultset concurrencyholdability的ResultSet的Statement对象
- 异常：已关闭时发生SQLException

createStruct

```
Struct createStruct(String typeName, Object[] attributes) throws
SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

getAutoCommit

```
boolean getAutoCommit() throws SQLException
```

- 操作：返回当前auto commit模式如果未调用过setAutoCommit()则返回true
- 异常：不发生

getCatalog

```
String getCatalog() throws SQLException
```

- 操作：获取数据库的当前目录名称
- 异常：从服务器发生错误或无响应时发生SQLException

getClientInfo

```
Properties getClientInfo() throws SQLException
```

- 操作：返回用户设置的客户端信息没有设置的信息时返回NULL
- 异常：不发生

```
String getClientInfo(String name) throws SQLException
```

- 操作：返回用户设置的特定客户端信息没有相关信息时返回NULL
- 异常：不发生

getHoldability

```
int getHoldability() throws SQLException
```

- 操作：返回从对象生成的Statement的默认holdability值默认值为

ResultSet.HOLD_CURSORS_OVER_COMMIT

- 异常：不发生

getMetaData

```
DatabaseMetaData getMetaData() throws SQLException
```

- 操作：获取可从此对象查询元信息的DatabaseMetaData对象总是返回相同的对象
- 异常：已关闭时发生SQLException

getNetworkTimeout

```
int getNetworkTimeout() throws SQLException
```

- 操作：返回当前network timeou(毫秒)0表示没有限制
- 异常：已经close时将发生SQLException
- Since: 1.7

getTransactionIsolation

```
int getTransactionIsolation() throws SQLException
```

- 操作：获取当前会话(connection)中设置的transaction isolation level服务器上设置的默认值为Connection.TRANSACTION_READ_COMMITTED
- 异常：从服务器收到错误时发生SQLException

getTypeMap

```
Map<String,Class<?>> getTypeMap() throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

getWarnings

```
SQLWarning getWarnings() throws SQLException
```

- 返回：返回目前为止Connection对象从服务器收到的warning没有或是clearWarnings()之后则返回NULL
- 异常：不发生

isClosed

```
boolean isClosed() throws SQLException
```

- 操作：询问是否成功调用过close()如果成功执行close()则返回true否则返回false
- 异常：不发生

> Note:
>
> 此method无法查看是否实际上与服务器的断开connection即使实际上connection已断开但如果用户没有调用过close()则返回true

isReadOnly

```
boolean isReadOnly() throws SQLException
```

- 操作：此会话(connection)为read-only模式时返回true否则返回false默认值为false产生与服务器的通讯
- 异常：从服务器收到错误或无响应时发生SQLException

isValid

```
boolean isValid(int timeout) throws SQLException
```

- 操作：isClosed()为返回true或将heart beat query传送给服务器后未收到成功执行的响应时返回false否则返回true
- 异常：不发生

nativeSQL

```
String nativeSQL(String sql) throws SQLException
```

- 操作：对指定的用户SQL返回服务器识别的native SQLSUNDB服务器直接识别用户的SQL语句因此总是返回用户定义的值
- 异常：不发生

prepareCall

```
CallableStatement prepareCall(String sql) throws SQLException
```

- 操作：将用于调出stored procedures的CallableStatement个体返回给用户从此CallableStatement生成的ResultSet类型为ResultSet.TYPE_FORWARD_ONLYconcurrency是ResultSet.CONCUR_READ_ONLY, holdability是 ResultSet.HOLD_CURSORS_OVER_COMMIT
- 异常：已经close或者SQL语句错误时将发生SQLException

```
CallableStatement prepareCall(String sql, int resultSetType, int
resultSetConcurrency) throws SQLException
```

- 操作：生成包含用户指定的resultset type和resultset concurrency的ResultSet的CallableStatement个体由此CallableStatement生成的ResultSet的holdability与connection的holdability类型相同Connection的默认holdability值为ResultSet.HOLD_CURSORS_OVER_COMMIT
- 异常：已经close或者SQL语句错误时将发生SQLException

```
CallableStatement prepareCall(String sql, int resultSetType, int
resultSetConcurrency, int resultSetHoldability) throws SQLException
```

- 操作：生成包含用户指定的resultset type和resultset concurrencyholdability的ResultSet的CallableStatement个体
- 异常：已经close或者SQL语句错误时将发生SQLException

prepareStatement

```
PreparedStatement prepareStatement(String sql) throws SQLException
```

- 操作：把sql语句传输至服务器并执行prepare (parsing, validation, optimization)像用户返回可控制其prepared statement的PreparedStatement对象从该PreparedStatement生成的

ResultSet的type为ResultSet.TYPE_FORWARD_ONLYconcurrency为ResultSet.CONCUR_READ_ONLYholdability为ResultSet.HOLD_CURSORS_OVER_COMMIT

- 异常：已关闭或sql语句有误时发生SQLException

```
PreparedStatement prepareStatement(String sql, int autoGeneratedKeys)
throws SQLException
```

- 操作：autoGeneratedKeys为Statement.NO_GENERATED_KEYS时与prepareStatement(String sql) method的操作相同autoGeneratedKeys为Statement.RETURN_GENERATED_KEYS并且sql是INSERT语句时将返回查询自动生成key的PreparedStatement对象
- 异常：已经关闭或sql语句有误时而且autoGeneratedKeys值既不是Statement.NO_GENERATED_KEYS也不是Statement.RETURN_GENERATED_KEYS时发生SQLException

> Caution:
>
> 表中如果没有自动生成key则ResultSet将变成closed状态

```
PreparedStatement prepareStatement(String sql, int[] columnIndexes) throws
SQLException
```

- 操作：columnIndexes为null时与prepareStatement(String sql) method的操作相同sql是INSERT语句并且columnIndexes不是null的情况下将返回使用已有的索引排列查询自动生成key的PreparedStatement对象
- 异常：已关闭或sql语句有误时发生SQLExceptioncolumnIndexes中存在无效的索引时发生SQLException

```
PreparedStatement prepareStatement(String sql, int resultSetType, int
resultSetConcurrency) throws SQLException
```

- 操作：把SQL语句传输至服务器并执行prepare (parsing, validation, optimization)向用户返回可控制其prepared statement的PreparedStatement对象创建生成拥有用户指定的resultset type和resultset concurrency的ResultSet的PreparedStatement对象从该PreparedStatement对象生成的ResultSet的holdability与Connection的holdability的类型相同Connection的默认holdability值为ResultSet.HOLD_CURSORS_OVER_COMMIT
- 异常：已关闭或sql语句有误时发生SQLException

```
PreparedStatement prepareStatement(String sql, int resultSetType, int
resultSetConcurrency, int resultSetHoldability) throws SQLException
```

- 操作：把sql语句传输至服务器并执行prepare (parsing, validation, optimization)向用户返回可控制其prepared statement的PreparedStatement对象创建生成拥有用户指定的resultset type和resultset concurrencyholdability的ResultSet的PreparedStatement对象
- 异常：已关闭或sql语句有误时发生SQLException

```
PreparedStatement prepareStatement(String sql, String[] columnNames)
throws SQLException
```

- 操作：columnNames为null时与prepareStatement(String sql) method的操作相同sql是INSERT语句并且columnNames不是null的情况下将返回使用已有的列名查询自动生成key的PreparedStatement对象
- 异常：已关闭或sql语句有误时发生SQLException columnNames中存在无效的列名时发生SQLException

releaseSavepoint

```
void releaseSavepoint(Savepoint savepoint) throws SQLException
```

- 操作：在服务器中清除该savepoint也清楚该savepoint之后的所有savepoint
- 异常：已关闭或该savepoint对象已被清除或非SUNDB savepoint对象时发生SQLException

rollback

```
void rollback() throws SQLException
```

- 操作：Non auto commit模式时回滚当前事务
- 异常：已关闭或auto commit模式时或从服务器收到错误时发生SQLException

```
void rollback(Savepoint savepoint) throws SQLException
```

- 操作：Non auto commit模式时对当前事务部分回滚（partial rollback）到当前savepoint
- 异常：已关闭或auto commit模式时或从服务器收到错误时或当前savepoint无效时发生SQLException

setAutoCommit

```
void setAutoCommit(boolean autoCommit) throws SQLException
```

- 操作：变更当前connection的auto commit模式有执行的事务并从non auto commit模式变更为auto commit模式时执行commit
- 异常：与commit过程中可发生的异常相同

setCatalog

```
void setCatalog(String catalog) throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

setClientInfo

```
void setClientInfo(Properties properties) throws SQLException
```

- 操作：设置用户指定的客户端信息之前的客户端信息将消失对服务器没有任何影响
- 异常：不发生

```
void setClientInfo(String name, String value) throws SQLException
```

- 操作：追加用户指定的客户端信息维持原有的客户端信息
- 异常：不发生

setHoldability

```
void setHoldability(int holdability) throws SQLException
```

- 操作：设置该对象生成的Statement生成的ResultSet的holdability不调用此method时默认值为ResultSet.HOLD_CURSORS_OVER_COMMIT
- 异常：不发生

setNetworkTimeout

```
void setNetworkTimeout(Executor executor, int milliseconds) throws
SQLException
```

- 操作：设置Connection或者从Connection生成的对象向数据库发出请求后等待响应的最长时间如果请求没有得到回复那么将与SQLException一起返回Connection对象将变为closed
- 异常：如果已经是close状态时executor为null时或者毫秒小于0时将发生SQLException当存在Security manager并且该checkPermission方法拒绝setNetworkTimeout调用时将发生SecurityException
- Since: 1.7

setReadOnly

```
void setReadOnly(boolean readOnly) throws SQLException
```

- 操作：设置当前会话 (connection)的read-only属性默认值为false
- 异常：已关闭或从服务器收到错误时发生SQLException

setSavepoint

```
Savepoint setSavepoint() throws SQLException
```

- 操作：Non auto commit模式时对当前事务设置savepoint内部决定savepoint名称
- 异常：已关闭或auto commit模式时或从服务器收到错误时发生SQLException

```
Savepoint setSavepoint(String name) throws SQLException
```

- 操作：Non auto commit模式时对当前事务设置用户指定的名称的savepoint
- 异常：已关闭或auto commit模式时或从服务器收到错误时发生SQLException

setTransactionIsolation

```
void setTransactionIsolation(int level) throws SQLException
```

- 操作：变更当前会话(Connection)的isolation级别支持的值为 Connection.TRANSACTION_READ_COMMITEDConnection.TRANSACTION_SERIALIZABLE Connection.READ_UNCOMMITTED变更为 Connection.TRANSACTION_READ_COMMITEDConnection.TRANSACTION_REPEATABLE_READ变更为Connection.TRANSACTION_SERIALIZABLE
- 异常：已关闭或level为错误的值时发生SQLException

setTypeMap

```
void setTypeMap(Map<String,Class<?>> map) throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

isWrapperFor

```
boolean isWrapperFor(Class<?> iface) throws SQLException
```

- 操作：询问此对象是否为实现iface接口的CLASS如果是则返回true否则返回falseSUNDB Connection对象不是其他CLASS的wrapper因此不判断wrapper的存在与否仅询问是否实现

了对应因子CLASS类型

- 异常：不发生

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- 操作：SUNDB Connection不是其他CLASS的wrapper因此即使unwrap也会返回自身转换为对应类型后返回iface为isWrapperFor() method的因子时如果返回false则此method发生异常
- 异常：iface不是此对象类型时（赋予此对象未执行implement的类型时）发生SQLException

ConnectionPoolDataSource

getPooledConnection

```
PooledConnection getPooledConnection() throws SQLException
```

- 操作：生成PooledConnection对象需要提前以setter method设置user namepasswordconnection URL等各种连接信息
- 异常：连接失败时发生SQLException

```
PooledConnection getPooledConnection(String user, String password) throws SQLException
```

- 操作：生成PooledConnection对象user name和password取决于argument其他连接信息应提前通过setter method设置
- 异常：连接失败时发生SQLException

DatabaseMetaData

allProceduresAreCallable

```
boolean allProceduresAreCallable() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

allTablesAreSelectable

```
boolean allTablesAreSelectable() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

autoCommitFailureClosesAllResultSets

```
boolean autoCommitFailureClosesAllResultSets() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

dataDefinitionCausesTransactionCommit

```
boolean dataDefinitionCausesTransactionCommit() throws SQLException
```

- 操作：执行DDL语句时不会自动提交因此总是返回false
- 异常：不发生

dataDefinitionIgnoredInTransactions

```
boolean dataDefinitionIgnoredInTransactions() throws SQLException
```

- 操作：DDL语句也包含于事务因此总是返回false
- 异常：不发生

deletesAreDetected

```
boolean deletesAreDetected(int type) throws SQLException
```

- 操作：type为ResultSet.TYPE_SCROLL_SENSITIVE时返回true否则返回false
- 异常：不发生

doesMaxRowSizeIncludeBlobs

```
boolean doesMaxRowSizeIncludeBlobs() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

getAttributes

```
ResultSet getAttributes(String catalog, String schemaPattern, String
```

```
typeNamePattern, String attributeNamePattern) throws SQLException
```

- 操作：返回empty ResultSet
- 异常：从服务器收到错误时发生SQLException

getBestRowIdentifier

```
ResultSet getBestRowIdentifier(String catalog, String schema, String
table, int scope, boolean nullable) throws SQLException
```

- 操作：对所有的表支持rowid类型因此返回由rowid信息构成的有1个行的ResultSet没有对应表时返回empty ResultSet
- 异常：表为null或从服务器收到错误时发生SQLException

getCatalogs

```
ResultSet getCatalogs() throws SQLException
```

- 操作：返回catalog名称为一个column的ResultSet
- 异常：从服务器收到错误时发生SQLException

getCatalogSeparator

```
String getCatalogSeparator() throws SQLException
```

- 操作：返回catalog separator字符
- 异常：从服务器收到错误时发生SQLException

getCatalogTerm

```
String getCatalogTerm() throws SQLException
```

- 操作：返回catalog term字符
- 异常：从服务器收到错误时发生SQLException

getClientInfoProperties

```
ResultSet getClientInfoProperties() throws SQLException
```

- 操作：返回有客户端信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

getColumnPrivileges

```
ResultSet getColumnPrivileges(String catalog, String schema, String table,
String columnNamePattern) throws SQLException
```

- 操作：返回拥有对用表的column的column privilege信息的ResultSet没有对应表或没有对应column名称模式的column时返回empty ResultSet
- 异常：从服务器收到错误时发生SQLException

getColumns

```
ResultSet getColumns(String catalog, String schemaPattern, String
tableNamePattern, String columnNamePattern) throws SQLException
```

- 操作：返回拥有对应表的所有column信息的ResultSet没有对应表或没有对应column名称模式的column时返回empty ResultSet
- 异常：从服务器收到错误时发生SQLException

getConnection

```
Connection getConnection() throws SQLException
```

- 操作：返回生成此DatabaseMetaData对象的connection对象
- 异常：不发生

getCrossReference

```
ResultSet getCrossReference(String parentCatalog, String parentSchema,
String parentTable, String foreignCatalog, String foreignSchema, String
foreignTable) throws SQLException
```

- 操作：返回指定foreign key表中参考指定parent table的foreign key的ResultSet没有参考关系时返回empty ResultSet
- 异常：Foreign table或parent table为null或从服务器收到错误时发生SQLException

getDatabaseMajorVersion

```
int getDatabaseMajorVersion() throws SQLException
```

- 操作：返回产品的major版本
- 异常：从服务器收到错误时发生SQLException

getDatabaseMinorVersion

```
int getDatabaseMinorVersion() throws SQLException
```

- 操作：返回产品的minor版本
- 异常：从服务器收到错误时发生SQLException

getDatabaseProductName

```
String getDatabaseProductName() throws SQLException
```

- 操作：返回产品名称
- 异常：从服务器收到错误时发生SQLException

getDatabaseProductVersion

```
String getDatabaseProductVersion() throws SQLException
```

- 操作：返回产品的product版本
- 异常：从服务器收到错误时发生SQLException

getDefaultTransactionIsolation

```
int getDefaultTransactionIsolation() throws SQLException
```

- 操作：返回default transaction isolation服务器上设置的默认值为

  Connection.TRANSACTION_READ_COMMITTED

- 异常：从服务器收到错误时发生SQLException

getDriverMajorVersion

```
int getDriverMajorVersion() throws SQLException
```

- 操作：返回SUNDB JDBC driver的major版本
- 异常：不发生

getDriverMinorVersion

```
int getDriverMinorVersion() throws SQLException
```

- 操作：返回SUNDB JDBC driver的minor版本
- 异常：不发生

getDriverName

```
String getDriverName() throws SQLException
```

- 操作：返回"SUNDB JDBC Driver"
- 异常：不发生

getDriverVersion

```
String getDriverVersion() throws SQLException
```

- 操作：返回SUNDB JDBC driver版本字符串包含protocol版本
- 异常：不发生

getExportedKeys

```
ResultSet getExportedKeys(String catalog, String schema, String table)
throws SQLException
```

- 操作：返回拥有参考指定表column的foreign key信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

getExtraNameCharacters

```
String getExtraNameCharacters() throws SQLException
```

- 操作：返回"-$"
- 异常：不发生

getFunctionColumns

```
ResultSet getFunctionColumns(String catalog, String schemaPattern, String
functionNamePattern, String columnNamePattern) throws SQLException
```

- 操作：返回empty ResultSet
- 异常：从服务器收到错误时发生SQLException

getFunctions

```
ResultSet getFunctions(String catalog, String schemaPattern, String
functionNamePattern) throws SQLException
```

- 操作：返回empty ResultSet
- 异常：从服务器收到错误时发生SQLException

getIdentifierQuoteString

```
String getIdentifierQuoteString() throws SQLException
```

- 操作：返回identifier quote字符服务器上设置的值为"
- 异常：从服务器收到错误时发生SQLException

getImportedKeys

```
ResultSet getImportedKeys(String catalog, String schema, String table)
throws SQLException
```

- 操作：返回拥有参考指定表的foreign key column的parent key信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

getIndexInfo

```
ResultSet getIndexInfo(String catalog, String schema, String table,
boolean unique, boolean approximate) throws SQLException
```

- 操作：返回拥有指定表的索引信息ResultSetunique为true时仅显示unique index信息忽略Approximate参数
- 异常：从服务器收到错误时发生SQLException

getJDBCMajorVersion

```
int getJDBCMajorVersion() throws SQLException
```

- 操作：返回SUNDB JDBC driver的JDBC major版本根据使用的jar文件会有所不同
- 异常：不发生

getJDBCMinorVersion

```
int getJDBCMinorVersion() throws SQLException
```

- 操作：返回SUNDB JDBC driver的JDBC minor版本根据使用的jar文件会有所不同
- 异常：不发生

getMaxBinaryLiteralLength

```
int getMaxBinaryLiteralLength() throws SQLException
```

- 操作：从服务器获取二进制的最大长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getMaxCatalogNameLength

```
int getMaxCatalogNameLength() throws SQLException
```

- 操作：从服务器获取catalog名称的最大长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getMaxCharLiteralLength

```
int getMaxCharLiteralLength() throws SQLException
```

- 操作：从服务器获取liter的最大长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getMaxColumnNameLength

```
int getMaxColumnNameLength() throws SQLException
```

- 操作：从服务器获取column名称的最大长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getMaxColumnsInGroupBy

```
int getMaxColumnsInGroupBy() throws SQLException
```

- 操作：从服务器获取可用于group by的column的最大数量0指最大数量为无限数量
- 异常：从服务器收到错误时发生SQLException

getMaxColumnsInIndex

```
int getMaxColumnsInIndex() throws SQLException
```

- 操作：从服务器获取可用于索引的column的最大数量0指最大数量为无限数量
- 异常：从服务器收到错误时发生SQLException

getMaxColumnsInOrderBy

```
int getMaxColumnsInOrderBy() throws SQLException
```

- 操作：从服务器获取可用于order by语句的column的最大数量0指最大数量为无限数量
- 异常：从服务器收到错误时发生SQLException

getMaxColumnsInSelect

```
int getMaxColumnsInSelect() throws SQLException
```

- 操作：从服务器获取可出现在select target语句的column的最大数量0指最大数量为无限数量
- 异常：从服务器收到错误时发生SQLException

getMaxColumnsInTable

```
int getMaxColumnsInTable() throws SQLException
```

- 操作：从服务器获取可出现在表的column的最大数量0指最大数量为无限数量

- 异常：从服务器收到错误时发生SQLException

getMaxConnections

```
int getMaxConnections() throws SQLException
```

- 操作：从服务器获取与服务器之间的connection的最大数量0指最大数量为无限数量
- 异常：从服务器收到错误时发生SQLException

getMaxCursorNameLength

```
int getMaxCursorNameLength() throws SQLException
```

- 操作：从服务器获取游标名称的最大长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getMaxIndexLength

```
int getMaxIndexLength() throws SQLException
```

- 操作：从服务器获取一个索引key可拥有的最大大小
- 异常：从服务器收到错误时发生SQLException

> Caution:
>
> 在JDBC规格中定义以byte返回一整个索引可拥有的最大大小但索引的大小无限制因此此值无意义只是为了与ODBC相同而使用

getMaxProcedureNameLength

```
int getMaxProcedureNameLength() throws SQLException
```

- 操作：从服务器获取Procedure名称的最大长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getMaxRowSize

```
int getMaxRowSize() throws SQLException
```

- 操作：从服务器获取一个表可容纳的最大row数量0指最大数量为无限数量
- 异常：从服务器收到错误时发生SQLException

getMaxSchemaNameLength

```
int getMaxSchemaNameLength() throws SQLException
```

- 操作：从服务器获取schema名称的最大长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getMaxStatementLength

```
int getMaxStatementLength() throws SQLException
```

- 操作：从服务器获取一个SQL语句可拥有的最大字符串长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getMaxStatements

```
int getMaxStatements() throws SQLException
```

- 操作：从服务器获取每次可open的statement的最大数量0指最大数量为无限数量
- 异常：从服务器收到错误时发生SQLException

getMaxTableNameLength

```
int getMaxTableNameLength() throws SQLException
```

- 操作：从服务器获取表名称可拥有的最大字符串长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getMaxTablesInSelect

```
int getMaxTablesInSelect() throws SQLException
```

- 操作：从服务器获取select语句中可使用的表的最大数量0指最大数量为无限数量
- 异常：从服务器收到错误时发生SQLException

getMaxUserNameLength

```
int getMaxUserNameLength() throws SQLException
```

- 操作：从服务器获取user name的最大字符串长度0指最大长度为无限长度
- 异常：从服务器收到错误时发生SQLException

getNumericFunctions

```
String getNumericFunctions() throws SQLException
```

- 操作：从服务器获取属于标准SQL的数字相关函数的目录各函数名称以逗号（）区分
- 异常：从服务器收到错误时发生SQLException

getPrimaryKeys

```
ResultSet getPrimaryKeys(String catalog, String schema, String table)
throws SQLException
```

- 操作：返回包含指定表的所有primary key的ResultSet
- 异常：从服务器收到错误时发生SQLException

getProcedureColumns

```
ResultSet getProcedureColumns(String catalog, String schemaPattern, String
procedureNamePattern, String columnNamePattern) throws SQLException
```

- 操作：返回对指定procedure拥有指定名称模式的column信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

getProcedures

```
ResultSet getProcedures(String catalog, String schemaPattern, String
procedureNamePattern) throws SQLException
```

- 操作：返回拥有指定名称模式的procedure信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

getProcedureTerm

```
String getProcedureTerm() throws SQLException
```

- 操作：从服务器获取指Procedure的关键字
- 异常：从服务器收到错误时发生SQLException

getResultSetHoldability

```
int getResultSetHoldability() throws SQLException
```

- 操作：返回ResultSet的默认holdability属性是ResultSet.HOLD_CURSORS_OVER_COMMIT
- 异常：不发生

getRowIdLifetime

```
RowIdLifetime getRowIdLifetime() throws SQLException
```

- 操作：总是返回 RowIdLifetime.ROWID_VALID_FOREVER
- 异常：不发生

getSchemas

```
ResultSet getSchemas() throws SQLException
```

- 操作：返回拥有服务器的所有schema信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

```
ResultSet getSchemas(String catalog, String schemaPattern) throws
SQLException
```

- 操作：返回满足指定schema名称模式的所有schema信息的ResultSet忽略catalog
- 异常：从服务器收到错误时发生SQLException

getSchemaTerm

```
String getSchemaTerm() throws SQLException
```

- 操作：从服务器获取指schema的关键字
- 异常：从服务器收到错误时发生SQLException

getSearchStringEscape

```
String getSearchStringEscape() throws SQLException
```

- 操作：以字符串返回like语句中使用的escape字符
- 异常：从服务器收到错误时发生SQLException

getSQLKeywords

```
String getSQLKeywords() throws SQLException
```

- 操作：返回不能用于SQL语句的关键字并以“,”区分
- 异常：从服务器收到错误时发生SQLException

getSQLStateType

```
int getSQLStateType() throws SQLException
```

- 操作：总是返回DatabaseMetaData.sqlStateSQL99
- 异常：不发生

getStringFunctions

```
String getStringFunctions() throws SQLException
```

- 操作：获取操作属于标准SQL的字符串的函数目录各函数名称以逗号（）区分
- 异常：从服务器收到错误时发生SQLException

getSuperTables

```
ResultSet getSuperTables(String catalog, String schemaPattern, String
tableNamePattern) throws SQLException
```

- 操作：返回Empty ResultSet
- 异常：从服务器收到错误时发生SQLException

getSuperTypes

```
ResultSet getSuperTypes(String catalog, String schemaPattern, String
typeNamePattern) throws SQLException
```

- 操作：返回Empty ResultSet
- 异常：从服务器收到错误时发生SQLException

getSystemFunctions

```
String getSystemFunctions() throws SQLException
```

- 操作：获取属于标准SQL的系统函数目录各函数以逗号（）区分
- 异常：从服务器收到错误时发生SQLException

getTablePrivileges

```
ResultSet getTablePrivileges(String catalog, String schemaPattern, String
tableNamePattern) throws SQLException
```

- 操作：返回拥有满足对应条件的所有表的privilege信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

getTables

```
ResultSet getTables(String catalog, String schemaPattern, String
tableNamePattern, String[] types) throws SQLException
```

- 操作：返回拥有满足对应条件的所有表信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

getTableTypes

```
ResultSet getTableTypes() throws SQLException
```

- 操作：返回拥有所有表类型信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

getTimeDateFunctions

```
String getTimeDateFunctions() throws SQLException
```

- 操作：获取属于标准SQL的TimeDate相关函数目录各函数以逗号（）区分
- 异常：从服务器收到错误时发生SQLException

getTypeInfo

```
ResultSet getTypeInfo() throws SQLException
```

- 操作：返回拥有数据类型信息的ResultSet
- 异常：从服务器收到错误时发生SQLException

getUDTs

```
ResultSet getUDTs(String catalog, String schemaPattern, String
```

```
typeNamePattern, int[] types) throws SQLException
```

- 操作：返回Empty ResultSet
- 异常：从服务器收到错误时发生SQLException

getURL

```
String getURL() throws SQLException
```

- 操作：返回连接服务器时使用的URL
- 异常：不发生

getUserName

```
String getUserName() throws SQLException
```

- 操作：返回维持会话的当前user name
- 异常：从服务器收到错误时发生SQLException

> Caution:
>
> 会话中可以变更user因此可能与第一次连接时的user name不同

getVersionColumns

```
ResultSet getVersionColumns(String catalog, String schema, String table)

throws SQLException
```

- 操作：返回Empty ResultSet
- 异常：从服务器收到错误时发生SQLException

insertsAreDetected

```
boolean insertsAreDetected(int type) throws SQLException
```

- 操作：与type无关总是返回false
- 异常：不发生

isCatalogAtStart

```
boolean isCatalogAtStart() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

isReadOnly

```
boolean isReadOnly() throws SQLException
```

- 操作：从服务器获取当前connection是否为read-only模式
- 异常：从服务器收到错误时发生SQLException

locatorsUpdateCopy

```
boolean locatorsUpdateCopy() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

nullPlusNonNullIsNull

```
boolean nullPlusNonNullIsNull() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

nullsAreSortedAtEnd

```
boolean nullsAreSortedAtEnd() throws SQLException
```

- 操作：总是返回false 不单独排序null
- 异常：不发生

nullsAreSortedAtStart

```
boolean nullsAreSortedAtStart() throws SQLException
```

- 操作：总是返回false 不单独排序null
- 异常：不发生

nullsAreSortedHigh

```
boolean nullsAreSortedHigh() throws SQLException
```

- 操作：总是返回truenull默认位于last
- 异常：不发生

nullsAreSortedLow

```
boolean nullsAreSortedLow() throws SQLException
```

- 操作：总是返回false null默认位于last
- 异常：不发生

othersDeletesAreVisible

```
boolean othersDeletesAreVisible(int type) throws SQLException
```

- 操作：type为ResultSet.TYPE_SCROLL_SENSITIVE时返回true否则返回false
- 异常：不发生

othersInsertsAreVisible

```
boolean othersInsertsAreVisible(int type) throws SQLException
```

- 操作：与type无关总是返回false
- 异常：不发生

othersUpdatesAreVisible

```
boolean othersUpdatesAreVisible(int type) throws SQLException
```

- 操作：Type为ResultSet.TYPE_SCROLL_SENSITIVE时返回true否则返回false
- 异常：不发生

ownDeletesAreVisible

```
boolean ownDeletesAreVisible(int type) throws SQLException
```

- 操作：Type为ResultSet.TYPE_SCROLL_SENSITIVE时返回true否则返回false
- 异常：不发生

ownInsertsAreVisible

```
boolean ownInsertsAreVisible(int type) throws SQLException
```

- 操作：与Type无关总是返回false
- 异常：不发生

ownUpdatesAreVisible

```
boolean ownUpdatesAreVisible(int type) throws SQLException
```

- 操作：Type为ResultSet.TYPE_SCROLL_SENSITIVE时返回true否则返回false
- 异常：不发生

storesLowerCaseIdentifiers

```
boolean storesLowerCaseIdentifiers() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

storesLowerCaseQuotedIdentifiers

```
boolean storesLowerCaseQuotedIdentifiers() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

storesMixedCaseIdentifiers

```
boolean storesMixedCaseIdentifiers() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

storesMixedCaseQuotedIdentifiers

```
boolean storesMixedCaseQuotedIdentifiers() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

storesUpperCaseIdentifiers

```
boolean storesUpperCaseIdentifiers() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

storesUpperCaseQuotedIdentifiers

```
boolean storesUpperCaseQuotedIdentifiers() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsAlterTableWithAddColumn

```
boolean supportsAlterTableWithAddColumn() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsAlterTableWithDropColumn

```
boolean supportsAlterTableWithDropColumn() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsANSI92EntryLevelSQL

```
boolean supportsANSI92EntryLevelSQL() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsANSI92FullSQL

```
boolean supportsANSI92FullSQL() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsANSI92IntermediateSQL

```
boolean supportsANSI92IntermediateSQL() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsBatchUpdates

```
boolean supportsBatchUpdates() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsCatalogsInDataManipulation

```
boolean supportsCatalogsInDataManipulation() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsCatalogsInIndexDefinitions

```
boolean supportsCatalogsInIndexDefinitions() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsCatalogsInPrivilegeDefinitions

```
boolean supportsCatalogsInPrivilegeDefinitions() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsCatalogsInProcedureCalls

```
boolean supportsCatalogsInProcedureCalls() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsCatalogsInTableDefinitions

```
boolean supportsCatalogsInTableDefinitions() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsColumnAliasing

```
boolean supportsColumnAliasing() throws SQLException
```

- 操作：从服务器获取是否支持Column aliasing
- 异常：从服务器收到错误时发生SQLException

supportsConvert

```
boolean supportsConvert() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsConvert

```
boolean supportsConvert(int fromType, int toType) throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsCoreSQLGrammar

```
boolean supportsCoreSQLGrammar() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsCorrelatedSubqueries

```
boolean supportsCorrelatedSubqueries() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsDataDefinitionAndDataManipulationTransactions

```
boolean supportsDataDefinitionAndDataManipulationTransactions() throws
SQLException
```

- 操作：总是返回true 可通过一个事务执行DML和DDL
- 异常：不发生

supportsDataManipulationTransactionsOnly

```
boolean supportsDataManipulationTransactionsOnly() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsDifferentTableCorrelationNames

```
boolean supportsDifferentTableCorrelationNames() throws SQLException
```

- 操作：从服务器获取Table correlation名称是否要与表名不一致
- 异常：从服务器收到错误时发生SQLException

supportsExpressionsInOrderBy

```
boolean supportsExpressionsInOrderBy() throws SQLException
```

- 操作：从服务器获取Order by语句中是否可以使用表达式
- 异常：从服务器收到错误时发生SQLException

supportsExtendedSQLGrammar

```
boolean supportsExtendedSQLGrammar() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsFullOuterJoins

```
boolean supportsFullOuterJoins() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsGetGeneratedKeys

```
boolean supportsGetGeneratedKeys() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsGroupBy

```
boolean supportsGroupBy() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsGroupByBeyondSelect

```
boolean supportsGroupByBeyondSelect() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsGroupByUnrelated

```
boolean supportsGroupByUnrelated() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsIntegrityEnhancementFacility

```
boolean supportsIntegrityEnhancementFacility() throws SQLException
```

- 操作：从服务器获取是否支持SQL完整性增强功能
- 异常：从服务器收到错误时发生SQLException

supportsLikeEscapeClause

```
boolean supportsLikeEscapeClause() throws SQLException
```

- 操作：从服务器获取Like语句中是否支持escape子句
- 异常：从服务器收到错误时发生SQLException

supportsLimitedOuterJoins

```
boolean supportsLimitedOuterJoins() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsMinimumSQLGrammar

```
boolean supportsMinimumSQLGrammar() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsMixedCaseIdentifiers

```
boolean supportsMixedCaseIdentifiers() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsMixedCaseQuotedIdentifiers

```
boolean supportsMixedCaseQuotedIdentifiers() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsMultipleOpenResults

```
boolean supportsMultipleOpenResults() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsMultipleResultSets

```
boolean supportsMultipleResultSets() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsMultipleTransactions

```
boolean supportsMultipleTransactions() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsNamedParameters

```
boolean supportsNamedParameters() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsNonNullableColumns

```
boolean supportsNonNullableColumns() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsOpenCursorsAcrossCommit

```
boolean supportsOpenCursorsAcrossCommit() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsOpenCursorsAcrossRollback

```
boolean supportsOpenCursorsAcrossRollback() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsOpenStatementsAcrossCommit

```
boolean supportsOpenStatementsAcrossCommit() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsOpenStatementsAcrossRollback

```
boolean supportsOpenStatementsAcrossRollback() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsOrderByUnrelated

```
boolean supportsOrderByUnrelated() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsOuterJoins

```
boolean supportsOuterJoins() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsPositionedDelete

```
boolean supportsPositionedDelete() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsPositionedUpdate

```
boolean supportsPositionedUpdate() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsResultSetConcurrency

```
boolean supportsResultSetConcurrency(int type, int concurrency) throws
SQLException
```

- 操作：对所有ResultSet类型与所有concurrency返回true对不正确的因子返回false

- 异常：不发生

supportsResultSetHoldability

```
boolean supportsResultSetHoldability(int holdability) throws SQLException
```

- 操作：Holdability为ResultSet.CLOSE_CURSORS_AT_COMMIT或ResultSet.HOLD_CURSORS_OVER_COMMIT时返回true否则返回false
- 异常：不发生

supportsResultSetType

```
boolean supportsResultSetType(int type) throws SQLException
```

- 操作：Type为ResultSet.TYPE_FORWARD_ONLY或ResultSet.TYPE_SCROLL_INSENSITIVE或ResultSet.TYPE_SCROLL_SENSITIVE时返回true否则返回false
- 异常：不发生

supportsSavepoints

```
boolean supportsSavepoints() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSchemasInDataManipulation

```
boolean supportsSchemasInDataManipulation() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSchemasInIndexDefinitions

```
boolean supportsSchemasInIndexDefinitions() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSchemasInPrivilegeDefinitions

```
boolean supportsSchemasInPrivilegeDefinitions() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSchemasInProcedureCalls

```
boolean supportsSchemasInProcedureCalls() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSchemasInTableDefinitions

```
boolean supportsSchemasInTableDefinitions() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSelectForUpdate

```
boolean supportsSelectForUpdate() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsStatementPooling

```
boolean supportsStatementPooling() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsStoredFunctionsUsingCallSyntax

```
boolean supportsStoredFunctionsUsingCallSyntax() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

supportsStoredProcedures

```
boolean supportsStoredProcedures() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSubqueriesInComparisons

```
boolean supportsSubqueriesInComparisons() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSubqueriesInExists

```
boolean supportsSubqueriesInExists() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSubqueriesInIns

```
boolean supportsSubqueriesInIns() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsSubqueriesInQuantifieds

```
boolean supportsSubqueriesInQuantifieds() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsTableCorrelationNames

```
boolean supportsTableCorrelationNames() throws SQLException
```

- 操作：从服务器获取是否支持Table correlation名称
- 异常：从服务器收到错误时发生SQLException

supportsTransactionIsolationLevel

```
boolean supportsTransactionIsolationLevel(int level) throws SQLException
```

- 操作：从服务器获取是否支持对应事务的隔离（isolation）级别
- 异常：从服务器收到错误时发生SQLException

supportsTransactions

```
boolean supportsTransactions() throws SQLException
```

- 操作：总是返回true
- 异常：不发生

supportsUnion

```
boolean supportsUnion() throws SQLException
```

- 操作：从服务器获取是否支持Union运算
- 异常：从服务器收到错误时发生SQLException

supportsUnionAll

```
boolean supportsUnionAll() throws SQLException
```

- 操作：从服务器获取是否支持Union all运算
- 异常：从服务器收到错误时发生SQLException

updatesAreDetected

```
boolean updatesAreDetected(int type) throws SQLException
```

- 操作：Type为ResultSet.TYPE_SCROLL_SENSITIVE时返回true否则返回false
- 异常：不发生

usesLocalFilePerTable

```
boolean usesLocalFilePerTable() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

usesLocalFiles

```
boolean usesLocalFiles() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

DataSource

getConnection

```
Connection getConnection() throws SQLException
```

- 操作：打开并返回新的connection对象Connection所需的信息为非标准method需要提前设置
- 异常：与服务器连接失败时发生SQLException

```
Connection getConnection(String username, String password) throws SQLException
```

- 操作：通过username与password打开的新的connection对象并返回Connection所需的信息为非标准method需要提前设置
- 异常：与服务器连接失败时发生SQLException

isWrapperFor

```
boolean isWrapperFor(Class<?> iface) throws SQLException
```

- 操作：此CLASS并非由其他CLASS的wrapper实现此对象为iface的instance时返回true否则返回false
- 异常：不发生

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- 操作：此CLASS并非由其他CLASS的wrapper实现因此返回this
- 异常：此对象不是iface的instance时发生SQLException

Driver

acceptsURL

```
boolean acceptsURL(String url) throws SQLException
```

- 操作：url不是nullurl以"jdbc:sundb:"开头时返回true否则返回false
- 异常：不发生

connect

```
Connection connect(String url, Properties info) throws SQLException
```

- 操作：生成并返回新的connection对象url应包含服务器地址DB名称和端口
- 异常：错误的url或连接服务器失败时发生SQLException

getMajorVersion

```
int getMajorVersion() throws SQLException
```

- 操作：返回SUNDB JDBC driver的major版本
- 异常：不发生

getMinorVersion

```
int getMinorVersion() throws SQLException
```

- 操作：返回SUNDB JDBC driver的minor版本
- 异常：不发生

getPropertyInfo

```
DriverPropertyInfo[] getPropertyInfo(String url, Properties info) throws
SQLException
```

- 操作：获取SUNDB JDBC的driver连接时使用的属性目录
- 异常：不发生

jdbcCompliant

```
boolean jdbcCompliant() throws SQLException
```

- 操作：总是返回false
- 异常：不发生

NClob

CLASS未实现

free

```
void free() throws SQLException
```

getAsciiStream

```
InputStream getAsciiStream() throws SQLException
```

getCharacterStream

```
Reader getCharacterStream() throws SQLException
```

```
Reader getCharacterStream(long pos, long length) throws SQLException
```

getSubString

```
String getSubString(long pos, int length) throws SQLException
```

length

```
long length() throws SQLException
```

position

```
long position(Clob searchstr, long start) throws SQLException
```

```
long position(String searchstr, long start) throws SQLException
```

setAsciiStream

```
OutputStream setAsciiStream(long pos) throws SQLException
```

setCharacterStream

```
Writer setCharacterStream(long pos) throws SQLException
```

setString

```
int setString(long pos, String str) throws SQLException
```

```
int setString(long pos, String str, int offset, int len) throws
SQLException
```

truncate

```
void truncate(long len) throws SQLException
```

ParameterMetaData

ParameterMetaData对象可以通过PreparedStatement.getParameterMetaData()获取但SUNDB JDBC的ParameterMetaData对除了in/ out与否信息外的类型等相关信息不使用实际数据库的信息而以默认类型Varchar为准由于Prepare后可从服务器获取的参数相关信息只有in/out例如通过"select * from t1 where a=?"语句进行prepare时条件语句中的参数类型为尚未指定的状态服务器默认假设为varchar类型

getParameterClassName

```
String getParameterClassName(int param) throws SQLException
```

- 操作：返回java.lang.String 对应参数类型视为varchar
- 异常：param值超出参数数量范围时发生SQLException

getParameterCount

```
int getParameterCount() throws SQLException
```

- 操作：返回参数数量也是语句中使用的？的数量
- 异常：不发生

getParameterMode

```
int getParameterMode(int param) throws SQLException
```

- 操作：返回对应第param个参数的in/out模式返回

ParameterMetaData.parameterModeInParameterMetaData.parameterModeInOutParameterMetaData.parameterModeOutParameterMetaData.parameterModeUnknown中的一个目前没有实际返回parameterModeUnknown值的情况

- 异常：param值超出参数数量范围时发生SQLException

getParameterType

```
int getParameterType(int param) throws SQLException
```

- 操作：对所有参数返回Types.VARCHAR
- 异常：param值超出参数数量范围时发生SQLException

getParameterTypeName

```
String getParameterTypeName(int param) throws SQLException
```

- 操作：对所有参数返回"VARCHAR"
- 异常：param值超出参数数量范围时发生SQLException

getPrecision

```
int getPrecision(int param) throws SQLException
```

- 操作：对所有参数返回4000参数默认视为varchar(4000)
- 异常：param值超出参数数量范围时发生SQLException

getScale

```
int getScale(int param) throws SQLException
```

- 操作：对所有参数返回0参数默认视为varchar(4000)
- 异常：param值超出参数数量范围时发生SQLException

isNullable

```
int isNullable(int param) throws SQLException
```

- 操作：总是返回ParameterMetaData.parameterNullableUnknown
- 异常：param值超出参数数量范围时发生SQLException

isSigned

```
boolean isSigned(int param) throws SQLException
```

- 操作：总是返回false
- 异常：param值超出参数数量范围时发生SQLException

isWrapperFor

```
boolean isWrapperFor(Class<?> iface) throws SQLException
```

- 操作：询问此对象是否为iface的实例如果是则返回true否则返回false
- 异常：不发生

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- 操作：如果对应对象是iface的实例则将此对象转换为iface类型并返回
- 异常：对应对象不是iface的实例时发生SQLException

PooledConnection

addConnectionEventListener

```
void addConnectionEventListener(ConnectionEventListener listener) throws
SQLException
```

- 操作：登记ConnectionEventListener对象之后调用从此PooledConnection获取的Connection对象 (logical connection)的close()或实际上connection断开时对登记的listener发生ConnectionEvent
- 异常：不发生

addStatementEventListener

```
void addStatementEventListener(StatementEventListener listener) throws
SQLException
```

- 操作：不做任何操作 SUNDB JDBC driver不支持此method由第三方中间件负责Statement pooling功能
- 异常：不发生

close

```
void close() throws SQLException
```

- 操作：调用对应对象的physical connection的close

- 异常：Physical connection的close中可能发生

getConnection

```
Connection getConnection() throws SQLException
```

- 操作：返回对应对象的logical connection
- 异常：不发生

removeConnectionEventListener

```
void removeConnectionEventListener(ConnectionEventListener listener)
throws SQLException
```

- 操作：清除登记过的ConnectionEventListener之后对对应listener不再传送ConnectionEvent
- 异常：不发生

removeStatementEventListener

```
void removeStatementEventListener(StatementEventListener listener) throws
SQLException
```

- 操作：不做任何操作
- 异常：不发生

PreparedStatement

addBatch

```
void addBatch() throws SQLException
```

- 操作：把当前绑定的数据登记为batch job只要登记一个batch job则执行execute()executeUpdate()executeQuery()时报错只要有一个未绑定Parameter则会报错执行addBatch后在未以setXXX()类method绑定的情况下再次执行addBatch则用之前绑定的值登记batch job
- 异常：存在一次都没绑定过的参数时发生异常

clearParameters

```
void clearParameters() throws SQLException
```

- 操作：清除所有当前绑定的数据和信息但是只要有一个登记的batch job就不进行任何操作
- 异常：不发生

execute

```
boolean execute() throws SQLException
```

- 操作：基于当前参数绑定的值执行prepare的statement执行的statement为select语句时返回true否则返回false即返回true时可以从对应对象获取ResultSet
- 异常：已登记batch job或绑定的参数不足或execution时如果服务器中报错则会发生异常

executeQuery

```
ResultSet executeQuery() throws SQLException
```

- 操作：执行基于绑定当前参数的数据prepare的statement并执行fetch后生成并返回ResultSet对象
- 异常：已登记batch job或绑定的参数不足或服务器中报错时可能发生异常

executeUpdate

```
int executeUpdate() throws SQLException
```

- 操作：执行基于绑定当前参数的数据prepare的statement返回更新的记录数如没有通过DDL语句等更新的记录时返回0
- 异常：已登记batch job或statement是select语句或绑定的参数不足或服务器中报错时可能发生异常

getMetaData

```
ResultSetMetaData getMetaData() throws SQLException
```

- 操作：获取ResultSetMetaData对象prepare的语句不是select时即不是返回ResultSet的语句时返回ResultSetMetaData此method可以在execute之前调用
- 异常：服务器上发生错误时可能发生

getParameterMetaData

```
ParameterMetaData getParameterMetaData() throws SQLException
```

- 操作：获取ParameterMetaData对象execute之前可以调用但是服务器无法查看参数的准确类型相关信息因此假设所有参数为varchar(4000)
- 异常：服务器上发生错误时可能发生

setArray

```
void setArray(int parameterIndex, Array x) throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

setAsciiStream

```
void setAsciiStream(int parameterIndex, InputStream x) throws SQLException
```

- 操作：在对应parameter索引以LONG VARCHAR类型绑定InputStream对象以二进制形式插入数据因此不执行encoding操作所以不考虑字符集假设为ascii数据
- 异常：parameterIndex小于0或大于参数数量时发生

> Caution:
>
> 以LONG VARCHAR绑定因此在服务器中会发生转换为VARCHAR的费用如果可查看长度最好使用void setAsciiStream(int parameterIndex, InputStream x, int length)

```
void setAsciiStream(int parameterIndex, InputStream x, int length) throws
SQLException
```

- 操作：在对应parameter索引以LONG VARCHAR类型绑定InputStream对象以二进制形式插入数据所以不执行encoding操作因此不考虑字符集假设为ascii数据插入英文数据或服务器端客户端的字符集环境相同时在varchar插入字符串时使用此method比使用setCharacterStream或setString稍快一些
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setAsciiStream(int parameterIndex, InputStream x, long length) throws
SQLException
```

- 操作：在对应parameter索引以LONG VARCHAR类型绑定InputStream对象以二进制形式插入数据所以不执行encoding操作因此不考虑字符集假设为ascii数据插入英文数据或服务器端客户端的字符集环境相同时在varchar插入字符串时使用此method比使用setCharacterStream或setString稍快一些
- 异常：parameterIndex小于0或大于参数数量时发生

setBigDecimal

```
void setBigDecimal(int parameterIndex, BigDecimal x) throws SQLException
```

- 操作：在对应parameter索引以NUMBER类型绑定BigDecimal对象
- 异常：parameterIndex小于0或大于参数数量时发生

setBinaryStream

```
void setBinaryStream(int parameterIndex, InputStream x) throws
SQLException
```

- 操作：在对应parameter索引以LONG VARBINARY类型绑定InputStream对象
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setBinaryStream(int parameterIndex, InputStream x, int length) throws
SQLException
```

- 操作：在对应parameter索引以LONG VARBINARY类型绑定InputStream对象
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setBinaryStream(int parameterIndex, InputStream x, long length)
throws SQLException
```

- 操作：在对应parameter索引以LONG VARBINARY类型绑定InputStream对象
- 异常：parameterIndex小于0或大于参数数量时发生

setBlob

```
void setBlob(int parameterIndex, Blob x) throws SQLException
```

- 操作：在对应parameter索引以LONG VARBINARY类型绑定blob对象
- 异常：parameterIndex小于0或者比parameter数大时将发生SQLException

```
void setBlob(int parameterIndex, InputStream inputStream) throws
SQLException
```

- 操作：在对应parameter索引以LONG VARBINARY类型绑定InputStream对象
- 异常：parameterIndex小于0或者比parameter数大时将发生SQLException

```
void setBlob(int parameterIndex, InputStream inputStream, long length)
throws SQLException
```

- 操作：在对应parameter索引以LONG VARBINARY类型绑定InputStream对象InputStream包含length长度的字符数
- 异常：parameterIndex小于0或者比parameter数大时将发生SQLException

setBoolean

```
void setBoolean(int parameterIndex, boolean x) throws SQLException
```

- 操作：在对应parameter索引以BOOLEAN类型绑定数据x
- 异常：parameterIndex小于0或大于参数数量时发生

setByte

```
void setByte(int parameterIndex, byte x) throws SQLException
```

- 操作：在对应parameter索引以NATIVE_SMALLINT类型绑定数据x
- 异常：parameterIndex小于0或大于参数数量时发生

setBytes

```
void setBytes(int parameterIndex, byte[] x) throws SQLException
```

- 操作：在对应parameter索引以VARBINARY或LONG VARBINARY类型绑定数据xx的长度小于4000时以VARBINARY类型绑定大于4000时以LONG VARBINARY类型绑定
- 异常：parameterIndex小于0或大于参数数量时发生

setCharacterStream

```
void setCharacterStream(int parameterIndex, Reader reader) throws
SQLException
```

- 操作：在对应parameter索引以LONG VARCHAR类型绑定reader对象
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setCharacterStream(int parameterIndex, Reader reader, int length)
throws SQLException
```

- 操作：在对应parameter索引以LONG VARCHAR类型绑定reader对象
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setCharacterStream(int parameterIndex, Reader reader, long length)
throws SQLException
```

- 操作：在对应parameter索引以LONG VARCHAR类型绑定reader对象
- 异常：parameterIndex小于0或大于参数数量时发生

setClob

```
void setClob(int parameterIndex, Clob x) throws SQLException
```

- 操作：在对应parameter索引以LONG VARCHAR类型绑定clob对象
- 异常：parameterIndex小于0或者比parameter个数大时将发生SQLException

```
void setClob(int parameterIndex, Reader reader) throws SQLException
```

- 操作：在对应parameter索引以LONG VARCHAR类型绑定reader对象
- 异常：parameterIndex小于0或者比parameter个数大时将发生SQLException

```
void setClob(int parameterIndex, Reader reader, long length) throws
SQLException
```

- 操作：在对应parameter索引以LONG VARCHAR类型绑定reader对象Reader包含length长度的字符数
- 异常：parameterIndex小于0或者比parameter个数大时将发生SQLException

setDate

```
void setDate(int parameterIndex, Date x) throws SQLException
```

- 操作：在对应parameter索引以DATE类型绑定数据x与调用setDate(parameterIndex, x, Calendar.getInstance())相同即适用local timezone
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setDate(int parameterIndex, Date x, Calendar cal) throws SQLException
```

- 操作：在对应parameter索引以DATE类型绑定数据xDate x使用cal的timezone的时间
- 异常：parameterIndex小于0或大于参数数量时发生

setDouble

```
void setDouble(int parameterIndex, double x) throws SQLException
```

- 操作：在对应parameter参数以NATIVE_DOUBLE类型绑定数据x
- 异常：parameterIndex小于0或大于参数数量时发生

setFixedCHAR

```
void setFixedCHAR(int parameterIndex, String x) throws SQLException
```

- 操作：将数据x以CHAR类型绑定到相应的参数索引上
- 异常：parameterIndex小于0或大于parameter数量时例外

setFloat

```
void setFloat(int parameterIndex, float x) throws SQLException
```

- 操作：在对应parameter索引以NATIVE_REAL类型绑定数据x
- 异常：parameterIndex小于0或大于参数数量时发生

setInt

```
void setInt(int parameterIndex, int x) throws SQLException
```

- 操作：在对应parameter参数以NATIVE_INTEGER类型绑定数据x
- 异常：parameterIndex小于0或大于参数数量时发生

setLong

```
void setLong(int parameterIndex, long x) throws SQLException
```

- 操作：在对应parameter参数以NATIVE_BIGINT类型绑定数据x
- 异常：parameterIndex小于0或大于参数数量时发生

setNCharacterStream

```
void setNCharacterStream(int parameterIndex, Reader value) throws
SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

```
void setNCharacterStream(int parameterIndex, Reader value, long length)
throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

setNClob

```
void setNClob(int parameterIndex, NClob value) throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

```
void setNClob(int parameterIndex, Reader reader) throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

```
void setNClob(int parameterIndex, Reader reader, long length) throws
SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

setNString

```
void setNString(int parameterIndex, String value) throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

setNull

```
void setNull(int parameterIndex, int sqlType) throws SQLException
```

- 操作：在对应parameter索引以属于sqlType的SUNDB类型绑定null映射sqlType的SUNDB类型参考**SQL类型 -> SUNDB类型**
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setNull(int parameterIndex, int sqlType, String typeName) throws
SQLException
```

- 操作：在对应parameter参数以属于sqlType的SUNDB类型绑定null映射sqlType的SUNDB类型参考**SQL类型 -> SUNDB类型**由于不支持REF或用户类型忽略第三个参数typeName
- 异常：parameterIndex小于0或大于参数数量时发生

setObject

```
void setObject(int parameterIndex, Object x) throws SQLException
```

- 操作：在对应parameter索引以映射的对应SUNDB类型绑定数据x
- 异常：parameterIndex小于0或大于参数数量时发生

JAVA CLASS	SUNDB类型
null	VARCHAR
Boolean	BOOLEAN

JAVA CLASS	SUNDB类型
Byte	NATIVE_SMALLINT
Short	NATIVE_SMALLINT
Integer	NATIVE_INTEGER
Long	NATIVE_BIGINT
Float	NATIVE_REAL
Double	NATIVE_DOUBLE
BigInteger	NATIVE_BIGINT
BigDecimal	NUMBER
String	VARCHAR or LONG VARCHAR
byte[]	VARBINARY or LONG VARBINARY
Date	DATE
Time	TIME
Timestamp	TIMESTAMP
Blob	N/A
Clob	N/A
InputStream	LONG VARBINARY
Reader	LONG VARCHAR
SUNDBInterval	对应INTERVAL类型

JAVA CLASS	SUNDB类型
RowId	ROWID
其他	N/A

Table 3-2 JAVA对象 → SUNDB类型

```
void setObject(int parameterIndex, Object x, int targetSqlType) throws
SQLException
```

- 操作：在对应parameter参数以属于targetSqlType的SUNDB类型绑定数据x映射targetSqlType的SUNDB类型参考**SQL类型 -> SUNDB类型**
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setObject(int parameterIndex, Object x, int targetSqlType, int
scaleOrLength) throws SQLException
```

- 操作：在对应parameterIndex以targetSqlType对应的SUNDB类型绑定数据x映射targetSqlType的SUNDB类型参考**SQL类型 -> SUNDB类型**x为InputStream或reader时scaleOrLength表示数据长度对其他类型忽略此值
- 异常：parameterIndex小于0或大于参数数量时发生

setRef

```
void setRef(int parameterIndex, Ref x) throws SQLException
```

- 操作：未实现

- 异常：总是发生SQLFeatureNotSupportedException

setRowId

```
void setRowId(int parameterIndex, RowId x) throws SQLException
```

- 操作：在对应parameter索引以ROWID类型绑定数据x
- 异常：parameterIndex小于0或大于参数数量时发生

setShort

```
void setShort(int parameterIndex, short x) throws SQLException
```

- 操作：在对应parameter索引以NATIVE_SMALLINT类型绑定数据x
- 异常：parameterIndex小于0或大于参数数量时发生

setSQLXML

```
void setSQLXML(int parameterIndex, SQLXML xmlObject) throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

setString

```
void setString(int parameterIndex, String x) throws SQLException
```

- 操作：在对应parameter索引以VARCHAR或LONG VARCHAR类型绑定数据xx的长度小于4000时以VARCHAR类型绑定大于4000时以LONG VARCHAR类型绑定
- 异常：parameterIndex小于0或大于参数数量时发生

setTime

```
void setTime(int parameterIndex, Time x) throws SQLException
```

- 操作：在对应parameter索引以TIME类型绑定数据x与调用setTime(parameterIndex, x, Calendar.getInstance())相同即适用local timezone
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setTime(int parameterIndex, Time x, Calendar cal) throws SQLException
```

- 操作：在对应parameter索引以TIME类型绑定数据xTime x使用cal的timezone的时间
- 异常：parameterIndex小于0或大于参数数量时发生

setTimestamp

```
void setTimestamp(int parameterIndex, Timestamp x) throws SQLException
```

- 操作：在对应parameter索引以TIME类型绑定数据x与调用setTimestamp(parameterIndex, x, Calendar.getInstance())相同即适用local timezone
- 异常：parameterIndex小于0或大于参数数量时发生

```
void setTimestamp(int parameterIndex, Timestamp x, Calendar cal) throws SQLException
```

- 操作：在对应parameter索引以TIMESTAMP类型绑定数据xTimestamp x使用cal的timezone的时间
- 异常：parameterIndex小于0或大于参数数量时发生

setUnicodeStream

```
void setUnicodeStream(int parameterIndex, InputStream x, int length)
throws SQLException
```

- 操作：未实现 (deprecad的method)
- 异常：总是发生SQLFeatureNotSupportedException

setURL

```
void setURL(int parameterIndex, URL x) throws SQLException
```

- 操作：未实现
- 异常：总是发生SQLFeatureNotSupportedException

isWrapperFor

```
boolean isWrapperFor(Class<?> iface) throws SQLException
```

- 操作：查询此对象是否为实现iface接口的CLASS如果是则返回true否则返回falseSUNDB PreparedStatement对象不是其他CLASS的wrapper因此不判断wrapper的存在与否只询问是否实现了指定的因子CLASS类型
- 异常：不发生

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- 操作：SUNDB PreparedStatement不是其他CLASS的wrapper因此即使unwrap也会返回自己转换为对应类型后返回iface为isWrapperFor() method的因子并返回false的值时此method发生异常
- 异常：iface不是此对象的类型时（此对象指定未implements的类型时）发生SQLException

executeBatchAtomic

```
boolean executeBatchAtomic() throws SQLException
```

- 操作：与executeBatch()相同但atomic的执行即batch job全部成功或全部失败中的一个比executeBatch()执行的快执行的语句为select时返回true否则返回false
- 异常：未登记Batch job或服务器中报错时发生SQLException

> Note:
>
> 是SUNDB JDBC的固有功能将PreparedStatement对象转换为SUNDBPreparedStatement后可使用
>
> 例: ((SundbPreparedStatement)pstmt).executeBatchAtomic();

setTimeTimeZone

```
void setTimeTimeZone(int parameterIndex, Time x, Calendar cal) throws
SQLException
```

- 操作：在对应parameter索引以TIME WITH TIME ZONE类型绑定数据xTime x使用cal的timezone时间数据库column的timezone信息参考cal的timezone
- 异常：parameterIndex小于0或大于参数数量时发生

> Note:
>
> 是SUNDB JDBC的固有功能将PreparedStatement对象转换为SUNDBPreparedStatement后可使用
>
> 例: ((SUNDBPreparedStatement)pstmt).setTimeTimeZone(1, aTime, aCalendar);

setTimestampTimeZone

```
void setTimestampTimeZone(int parameterIndex, Timestamp x, Calendar cal)
throws SQLException
```

- 操作：在对应parameter索引以TIMESTAMP WITH TIME ZONE类型绑定数据xTimestamp x视为cal的timezone时间数据库column的timezone信息参考cal的timezone
- 异常：parameterIndex小于0或大于参数数量时发生

> Note:
>
> 是SUNDB JDBC的固有功能将PreparedStatement对象转换为SUNDBPreparedStatement后可使用
>
> 例: ((SUNDBPreparedStatement)pstmt).setTimestampTimeZone(1, aTimestamp, aCalendar);

Ref

CLASS未实现

getBaseTypeName

```
String getBaseTypeName() throws SQLException
```

getObject

```
Object getObject() throws SQLException
```

```
Object getObject(Map<String,Class<?>> map) throws SQLException
```

setObject

```
void setObject(Object value) throws SQLException
```

ResultSet

absolute

```
boolean absolute(int row) throws SQLException
```

- 操作：获取的row游标位于第row行第一行为10指before first负数表示从最后一行开始即-1表示最后一行-2表示倒数第2行如果可在获取的行缓存内放置游标则仅更改缓存中的位置信息如果无法放置在缓存内则从服务器上重新获取row超出范围时将游标位于before first或after last并返回false其他情况将游标位于对应行并返回true
- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY或获取时服务器中出错时发生异常

> Note:
>
> 不在缓存内而需要从服务器重新获取时如果row位于当前位置的后面则为了有利于next()从服务器获取从row位置开始n（从服务器获取的记录条数）个如果row位于当前位置的前面则为了有利于previous()从(row-n+1)位置开始获取n个

afterLast

```
void afterLast() throws SQLException
```

- 操作：将Row游标位于after last行缓存为最后一个row set（指整个result set的一部分的用语）时仅变更游标位置否则获取最后一个row set（整个记录数-n+1开始n个行n为获取时从服务器获取的记录条数）后将游标位于after last

- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY或获取时服务器中出错时发生异常

beforeFirst

```
void beforeFirst() throws SQLException
```

- 操作：将Row游标位于before first行缓存为第一个row set（指整个result set的一部分）时只变更游标位置否则获取第一个row set（从1开始n个行n为从服务器获取时获取的记录条数）后将游标位于before first
- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY或服务器中出错时发生异常

cancelRowUpdates

```
void cancelRowUpdates() throws SQLException
```

- 操作：目前未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考操作

clearWarnings

```
void clearWarnings() throws SQLException
```

- 操作：清除ResultSet对象拥有的所有SQLWarning对象

- 异常：不发生

close

```
void close() throws SQLException
```

- 操作：服务器中的对应游标已打开时关闭其游标（服务器中游标已关闭时不执行此操作即没有通讯交互）并将当前ResultSet对象的状态变更为closed如果已关闭则不做任何操作
- 异常：服务器中出错时发生SQLException

deleteRow

```
void deleteRow() throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

findColumn

```
int findColumn(String columnLabel) throws SQLException
```

- 操作：返回对应column名称的索引第一个column的索引为1
- 异常：已关闭或找不到对应名称的column时发生SQLException

first

```
boolean first() throws SQLException
```

- 操作：将Row游标位于first（第1行）行缓存为第一个row set（指整个result set的一部分）时只变更游标位置否则获取第一个row set（从1开始n个行n为从服务器获取时获取的记录条数）后将游标位于first有对应行时返回true否则返回false
- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY或获取时服务器中出错时发生异常

getArray

```
Array getArray(int columnIndex) throws SQLException
```

- 操作：不支持Array类型
- 异常：总是发生SQLFeatureNotSupportedException

```
Array getArray(String columnLabel) throws SQLException
```

- 操作：不支持Array类型SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：总是发生SQLFeatureNotSupportedException

getAsciiStream

```
InputStream getAsciiStream(int columnIndex) throws SQLException
```

- 操作：以InputStream类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为InputStream时发生SQLException

```
InputStream getAsciiStream(String columnLabel) throws SQLException
```

- 操作：以InputStream类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到对应columnLabel或对应类型无法转换为InputStream时发生SQLException

getBigDecimal

```
BigDecimal getBigDecimal(int columnIndex) throws SQLException
```

- 操作：以BigDecimal类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为BigDecimal时发生SQLException

```
BigDecimal getBigDecimal(int columnIndex, int scale) throws SQLException
```

- 操作：Deprecated的method与getBigDecimal(int columnIndex)的操作相同忽略scale
- 异常：参考**getBigDecimal**(int columnIndex)

```
BigDecimal getBigDecimal(String columnLabel) throws SQLException
```

- 操作：以BigDecimal类型获取名称为columnLabel的column数据SUNDB类型的额外支持部分参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为BigDecimal时发生SQLException

```
BigDecimal getBigDecimal(String columnLabel, int scale) throws
SQLException
```

- 操作：Deprecated的method与getBigDecimal(String columnLabel)的操作相同忽略Scale
- 异常：参考**getBigDecimal**(int columnIndex)

getBinaryStream

```
InputStream getBinaryStream(int columnIndex) throws SQLException
```

- 操作：与getAsciiStream(int columnIndex)相同
- 异常：参考**getAsciiStream**(int columnIndex)

```
InputStream getBinaryStream(String columnLabel) throws SQLException
```

- 操作：与getAsciiStream(String columnLabel)相同
- 异常：参考**getAsciiStream**(String columnLabel)

getBlob

```
Blob getBlob(int columnIndex) throws SQLException
```

- 操作：以blob类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为blob时发生SQLException

```
Blob getBlob(String columnLabel) throws SQLException
```

- 操作：以blob类型获取名称为columnLabel的column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为blob时发生SQLException

getBoolean

```
boolean getBoolean(int columnIndex) throws SQLException
```

- 操作：以boolean类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为boolean时发生SQLException

```
boolean getBoolean(String columnLabel) throws SQLException
```

- 操作：以boolean类型获取名称为columnLabel的column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为boolean时发生SQLException

getByte

```
byte getByte(int columnIndex) throws SQLException
```

- 操作：以byte类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为byte时发生SQLException

```
byte getByte(String columnLabel) throws SQLException
```

- 操作：以byte类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为byte类型时发生SQLException

getBytes

```
byte[] getBytes(int columnIndex) throws SQLException
```

- 操作：以byte[]类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围时发生SQLException

> Note:
>
> 对SUNDB的所有数据类型执行getBytes时获取存储于DB的二进制形式

```
byte[] getBytes(String columnLabel) throws SQLException
```

- 操作：以byte[]类型获取名称为columnLabel的column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel时发生SQLException

> Note:
>
> 对SUNDB的所有数据类型执行getBytes时获取存储于数据库的二进制形式

getCharacterStream

```
Reader getCharacterStream(int columnIndex) throws SQLException
```

- 操作：以reader类型获取columnIndex column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为reader时发生SQLException

```
Reader getCharacterStream(String columnLabel) throws SQLException
```

- 操作：以reader类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为reader类型时发生SQLException

getClob

```
Clob getClob(int columnIndex) throws SQLException
```

- 操作：以clob类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为clob时发生SQLException

```
Clob getClob(String columnLabel) throws SQLException
```

- 操作：以clob类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为clob类型时发生SQLException

getConcurrency

```
int getConcurrency() throws SQLException
```

- 操作：返回当前ResultSet对象的concurrency目前仅支持ResultSet.CONCUR_READ_ONLY
- 异常：不发生

getCursorName

```
String getCursorName() throws SQLException
```

- 操作：获取此ResultSet指向的服务器的游标名称与服务器发生通讯
- 异常：ResultSet已关闭或服务器中出错时发生SQLException

getDate

```
Date getDate(int columnIndex) throws SQLException
```

- 操作：以date类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Date对象时使用local timezone
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为date时发生SQLException

```
Date getDate(int columnIndex, Calendar cal) throws SQLException
```

- 操作：以date类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Date对象时使用cal的timezone
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为date时发生SQLException

```
Date getDate(String columnLabel) throws SQLException
```

- 操作：以date类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**生成Date对象时使用local timezone
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为date时发生SQLException

```
Date getDate(String columnLabel, Calendar cal) throws SQLException
```

- 操作：以date类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Date对象时使用cal的timezone
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为date时发生SQLException

getDouble

```
double getDouble(int columnIndex) throws SQLException
```

- 操作：以double类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为double时发生SQLException

```
double getDouble(String columnLabel) throws SQLException
```

- 操作：以double类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为double时发生SQLException

getFetchDirection

```
int getFetchDirection() throws SQLException
```

- 操作：总是返回ResultSet.FETCH_FORWARD不支持Backward fetch
- 异常：ResultSet已关闭时发生SQLException

getFetchSize

```
int getFetchSize() throws SQLException
```

- 操作：获取每次fetch时从服务器获取的行数量0时自动计算每次传送时通信packet可容纳的最大行数默认值为0
- 异常：ResultSet已关闭时发生SQLException

getFloat

```
float getFloat(int columnIndex) throws SQLException
```

- 操作：以float类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为float时发生SQLException

```
float getFloat(String columnLabel) throws SQLException
```

- 操作：以float类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为float时发生SQLException

getHoldability

```
int getHoldability() throws SQLException
```

- 操作：返回当前ResultSet的holdability生成ResultSet对象时指定此值中间无法变更默认值为ResultSet.HOLD_CURSOR_OVER_COMMIT
- 异常：ResultSet已关闭时发生SQLException

getInt

```
int getInt(int columnIndex) throws SQLException
```

- 操作：以int类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为int时发生SQLException

```
int getInt(String columnLabel) throws SQLException
```

- 操作：以int类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为int类型时发生SQLException

getLong

```
long getLong(int columnIndex) throws SQLException
```

- 操作：以long类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为long时发生

SQLException

```
long getLong(String columnLabel) throws SQLException
```

- 操作：以long类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为long时发生SQLException

getMetaData

```
ResultSetMetaData getMetaData() throws SQLException
```

- 操作：生成并返回可获取Column详细信息的ResultSetMetaData对象
- 异常：ResultSet已关闭或从服务器获取column详细信息的过程中出错时发生SQLException

getNCharacterStream

```
Reader getNCharacterStream(int columnIndex) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 异常：总是返回SQLFeatureNotSupportedException

```
Reader getNCharacterStream(String columnLabel) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 异常：总是返回SQLFeatureNotSupportedException

getNClob

```
NClob getNClob(int columnIndex) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 异常：总是返回SQLFeatureNotSupportedException

```
NClob getNClob(String columnLabel) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 异常：总是返回SQLFeatureNotSupportedException

getNString

```
String getNString(int columnIndex) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 异常：总是返回SQLFeatureNotSupportedException

```
String getNString(String columnLabel) throws SQLException
```

- 操作：目前不支持NCHAR系列类型
- 异常：总是返回SQLFeatureNotSupportedException

getObject

```
Object getObject(int columnIndex) throws SQLException
```

- 操作：以最合适的JAVA对象类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围时发生SQLException

```
Object getObject(int columnIndex, Map<String,Class<?>> map) throws
SQLException
```

- 操作：不支持
- 异常：总是返回SQLFeatureNotSupportedException

```
Object getObject(String columnLabel) throws SQLException
```

- 操作：以最合适的JAVA对象类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel时发生SQLException

```
Object getObject(String columnLabel, Map<String,Class<?>> map) throws
SQLException
```

- 操作：不支持
- 异常：总是返回SQLFeatureNotSupportedException

getRef

```
Ref getRef(int columnIndex) throws SQLException
```

- 操作：不支持REF类型

- 异常：总是返回SQLFeatureNotSupportedException

```
Ref getRef(String columnLabel) throws SQLException
```

- 操作：不支持REF类型
- 异常：总是返回SQLFeatureNotSupportedException

getRow

```
int getRow() throws SQLException
```

- 操作：返回当前ResultSet对象的游标位置第一行为1before first时返回 0
- 异常：ResultSet已关闭时发生SQLException

getRowId

```
RowId getRowId(int columnIndex) throws SQLException
```

- 操作：以RowId类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为RowId时发生SQLException

```
RowId getRowId(String columnLabel) throws SQLException
```

- 操作：以RowId类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**

- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为RowId时发生SQLException

getShort

```
short getShort(int columnIndex) throws SQLException
```

- 操作：以short类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为Short时发生SQLException

```
short getShort(String columnLabel) throws SQLException
```

- 操作：以short类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为short时发生SQLException

getSQLXML

```
SQLXML getSQLXML(int columnIndex) throws SQLException
```

- 操作：不支持SQLXML类型
- 异常：总是范围SQLFeatureNotSupportedException

```
SQLXML getSQLXML(String columnLabel) throws SQLException
```

- 操作：不支持SQLXML类型
- 异常：总是返回SQLFeatureNotSupportedException

getStatement

```
Statement getStatement() throws SQLException
```

- 操作：返回生成此ResultSet对象的Statement对象
- 异常：ResultSet已关闭时发生SQLException

getString

```
String getString(int columnIndex) throws SQLException
```

- 操作：以string类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 对BINARYVARBINARYLONG VARBINARY执行getString()时返回hex code的字符串
- 异常：ResultSet已关闭或columnIndex超出范围时发生SQLException

```
String getString(String columnLabel) throws SQLException
```

- 操作：以string类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1**对BINARYVARBINARYLONG VARBINARY执行getString()时返回hex code的字符串
- 异常：ResultSet已关闭或无法找到columnLabel时发生SQLException

getTime

```
Time getTime(int columnIndex) throws SQLException
```

- 操作：以time类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Time对象时使用local timezone
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为time时发生SQLException

```
Time getTime(int columnIndex, Calendar cal) throws SQLException
```

- 操作：以time类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Time对象时使用cal的timezone
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为time时发生SQLException

```
Time getTime(String columnLabel) throws SQLException
```

- 操作：以time类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Time对象时使用local timezone
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为time时发生SQLException

```
Time getTime(String columnLabel, Calendar cal) throws SQLException
```

- 操作：以time类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Time对象时使用local timezone

- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为time时发生SQLException

getTimestamp

```
Timestamp getTimestamp(int columnIndex) throws SQLException
```

- 操作：以timestamp类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Timestamp对象时使用local timezone
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为timestamp时发生SQLException

```
Timestamp getTimestamp(int columnIndex, Calendar cal) throws SQLException
```

- 操作：以timestamp类型获取第columnIndex个column的数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Timestamp对象时使用cal的timezone.
- 异常：ResultSet已关闭或columnIndex超出范围或对应类型无法转换为timestamp时发生SQLException

```
Timestamp getTimestamp(String columnLabel) throws SQLException
```

- 操作：以timestamp类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Timestamp对象时使用local timezone
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为timestamp类型时发生SQLException

```
Timestamp getTimestamp(String columnLabel, Calendar cal) throws
```

```
SQLException
```

- 操作：以timestamp类型获取名称为columnLabel的column数据SUNDB各类型的支持与否参考**对SUNDB类型的getter method支持与否-1** 生成Timestamp对象时使用cal的timezone
- 异常：ResultSet已关闭或无法找到columnLabel或对应类型无法转换为timestamp时发生SQLException

getType

```
int getType() throws SQLException
```

- 操作：返回当前ResultSet类型返回ResultSet.TYPE_FORWARD_ONLYResultSet.TYPE_SCROLL_INSENSITIVEResultSet.TYPE_SCROLL_SENSITIVE中的一个
- 异常：ResultSet已关闭时发生SQLException

getUnicodeStream

```
InputStream getUnicodeStream(int columnIndex) throws SQLException
```

- 操作：Deprecated的method未实现
- 异常：总是发生SQLFeatureNotSupportedException

```
InputStream getUnicodeStream(String columnLabel) throws SQLException
```

- 操作：Deprecated的method未实现
- 异常：总是发生SQLFeatureNotSupportedException

getURL

```
URL getURL(int columnIndex) throws SQLException
```

- 操作：不支持URL类型
- 异常：总是返回SQLFeatureNotSupportedException

```
URL getURL(String columnLabel) throws SQLException
```

- 操作：不支持URL类型
- 异常：总是返回SQLFeatureNotSupportedException

getWarnings

```
SQLWarning getWarnings() throws SQLException
```

- 操作：返回此对象到目前为止积累的SQLWarning目录从服务器收到告警时生成SQLWarning如果不执行clearWarning则一直积累没有时返回null
- 异常：不发生

insertRow

```
void insertRow() throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException

- 异常：已关闭时发生SQLException其他情况参考实际操作

isAfterLast

```
boolean isAfterLast() throws SQLException
```

- 操作：询问当前游标位置是否为after last如果是则返回true否则返回false
- 异常：ResultSet已关闭时发生SQLException

isBeforeFirst

```
boolean isBeforeFirst() throws SQLException
```

- 操作：询问当前游标位置是否为before first 如果是则返回true否则返回false
- 异常：ResultSet已关闭时发生SQLException

isClosed

```
boolean isClosed() throws SQLException
```

- 操作：询问当前ResultSet是否关闭如果关闭则返回true否则返回false即使用户未调用close关闭服务器的游标时也可能关闭ResultSet例如生成ResultSet的Statement已关闭或holdability为ResultSet.CLOSE_CURSOR_AT_COMMIT模式时提交事务或fetch过程中服务器返回错误等
- 异常：不发生

isFirst

```
boolean isFirst() throws SQLException
```

- 操作：询问当前游标位置是否为first(第一行) 如果是则返回true否则返回false
- 异常：ResultSet已关闭时发生SQLException

isLast

```
boolean isLast() throws SQLException
```

- 操作：询问当前游标位置是否为last(最后一行) 如果是则返回true否则返回false
- 异常：ResultSet已关闭时发生SQLException

last

```
boolean last() throws SQLException
```

- 操作：将行游标位于last（最后一行）行缓存为最后一个row set（整个result set的一部分）时只变更游标位置否则获取最后一个row set（从last-n+1开始last行n为每次从服务器获取的行数）后将游标位于last有对应行时返回true反之返回false
- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY或fetch时在服务器出错时发生异常

moveToCurrentRow

```
void moveToCurrentRow() throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

moveToInsertRow

```
void moveToInsertRow() throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

next

```
boolean next() throws SQLException
```

- 操作：将Row游标位于当前行的下一个行当前行为行缓存的最后一个行时从服务器获取下一个行缓存有对应行时返回true否则返回false
- 异常：ResultSet已关闭或fetch过程中服务器返回错误时发生异常

previous

```
boolean previous() throws SQLException
```

- 操作：将Row游标位于当前位置的上一行当前行是行缓存的第一个行时从服务器获取上一

个行缓存(从x-n到x-1的n个行x为当前行索引)有对应行时返回true否则返回false

- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY或fetch过程中在服务器出错时发生异常

refreshRow

```
void refreshRow() throws SQLException
```

- 操作：ResultSet类型为ResultSet.SCROLL_SENSITIVE时从服务器获取当前行缓存如果此过程中有变更（被同一个事务或其他事务）的行数据则反映此变更数据ResultSet类型为ResultSet.Scroll_INSENSITIVE时不做任何操作
- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY或fetch过程中在服务器出错时发生异常

relative

```
boolean relative(int rows) throws SQLException
```

- 操作：将Row游标从当前位置开始移动至与rows数相同的数如果可以在当前行缓存内移动则只变更游标位置否则从服务器获取对应行缓存后定位游标此时如果要移动的位置位于当前行的后面（next方向）则从服务器feth从rows到rows+n-1的行缓存（有利于next）如果要移动的位置位于当前行的前面（previous方向）则从服务器feth从rows-n+1到rows的行缓存（有利于previous）
- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY或fetch过程中服务器中出错时发生异常

rowDeleted

```
boolean rowDeleted() throws SQLException
```

- 操作：询问是否已delete（被同一个事务或其他事务）当前游标位置的行已Delete时返回true否则返回false
- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY时发生异常

rowInserted

```
boolean rowInserted() throws SQLException
```

- 操作：询问是否已insert（被同一个事务或其他事务）当前游标位置的行已Insert时返回true否则返回false
- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY时发生异常

rowUpdated

```
boolean rowUpdated() throws SQLException
```

- 操作：询问是否已update（被同一个事务或其他事务）当前游标位置的行已update时返回true否则返回false
- 异常：ResultSet已关闭或ResultSet类型为ResultSet.TYPE_FORWARD_ONLY时发生异常

setFetchDirection

```
void setFetchDirection(int direction) throws SQLException
```

- 操作：服务器不支持back-ward fetch因此只允许ResultSet.FETCH_FORWARD输入其他值时生成SQLWarning
- 异常：ResultSet已关闭或输入了其他非定义的值时发生异常

setFetchSize

```
void setFetchSize(int rows) throws SQLException
```

- 操作：指定从服务器一次fetch的行数0表示由服务器自动决定0时对forward only游标指定为通信packet可一次性容纳的行数scrollable游标时指定为100个
- 异常：已关闭ResultSet时发生SQLException

updateArray

```
void updateArray(int columnIndex, Array x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateArray(String columnLabel, Array x) throws SQLException
```

- 操作：目前尚未支持cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateAsciiStream

```
void updateAsciiStream(int columnIndex, InputStream x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateAsciiStream(int columnIndex, InputStream x, int length) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateAsciiStream(int columnIndex, InputStream x, long length) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateAsciiStream(String columnLabel, InputStream x) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateAsciiStream(String columnLabel, InputStream x, int length)
throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateAsciiStream(String columnLabel, InputStream x, long length)
throws SQLException
```

- 操作：目前尚未支持cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateBigDecimal

```
void updateBigDecimal(int columnIndex, BigDecimal x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发

生SQLFeatureNotSupportedException

- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBigDecimal(String columnLabel, BigDecimal x) throws
SQLException
```

- 操作：目前尚未支持cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateBinaryStream

```
void updateBinaryStream(int columnIndex, InputStream x) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBinaryStream(int columnIndex, InputStream x, int length) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException

- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBinaryStream(int columnIndex, InputStream x, long length)
throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBinaryStream(String columnLabel, InputStream x) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBinaryStream(String columnLabel, InputStream x, int length)
throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBinaryStream(String columnLabel, InputStream x, long length)
```

```
throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateBlob

```
void updateBlob(int columnIndex, Blob x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBlob(int columnIndex, InputStream inputStream) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBlob(int columnIndex, InputStream inputStream, long length)
throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBlob(String columnLabel, Blob x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBlob(String columnLabel, InputStream inputStream) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBlob(String columnLabel, InputStream inputStream, long length)
throws SQLException
```

- 操作：目前尚未支持cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateBoolean

```
void updateBoolean(int columnIndex, boolean x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBoolean(String columnLabel, boolean x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateByte

```
void updateByte(int columnIndex, byte x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateByte(String columnLabel, byte x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateBytes

```
void updateBytes(int columnIndex, byte[] x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateBytes(String columnLabel, byte[] x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateCharacterStream

```
void updateCharacterStream(int columnIndex, Reader x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发

生SQLFeatureNotSupportedException

- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateCharacterStream(int columnIndex, Reader x, int length) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateCharacterStream(int columnIndex, Reader x, long length) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateCharacterStream(String columnLabel, Reader reader) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateCharacterStream(String columnLabel, Reader reader, int length)
throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateCharacterStream(String columnLabel, Reader reader, long length)
throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateClob

```
void updateClob(int columnIndex, Clob x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateClob(int columnIndex, Reader reader) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateClob(int columnIndex, Reader reader, long length) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateClob(String columnLabel, Clob x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateClob(String columnLabel, Reader reader) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateClob(String columnLabel, Reader reader, long length) throws
SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateDate

```
void updateDate(int columnIndex, Date x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateDate(String columnLabel, Date x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateDouble

```
void updateDouble(int columnIndex, double x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateDouble(String columnLabel, double x) throws SQLException
```

- 操作：目前尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateFloat

```
void updateFloat(int columnIndex, float x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateFloat(String columnLabel, float x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateInt

```
void updateInt(int columnIndex, int x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateInt(String columnLabel, int x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateLong

```
void updateLong(int columnIndex, long x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateLong(String columnLabel, long x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateNCharacterStream

```
void updateNCharacterStream(int columnIndex, Reader x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNCharacterStream(int columnIndex, Reader x, long length) throws
SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNCharacterStream(String columnLabel, Reader reader) throws
SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生

SQLFeatureNotSupportedException

- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNCharacterStream(String columnLabel, Reader reader, long
length) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateNClob

```
void updateNClob(int columnIndex, NClob nClob) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNClob(int columnIndex, Reader reader) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNClob(int columnIndex, Reader reader, long length) throws
SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNClob(String columnLabel, NClob nClob) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNClob(String columnLabel, Reader reader) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNClob(String columnLabel, Reader reader, long length) throws
SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生

SQLFeatureNotSupportedException

- 异常：已关闭时发生SQLException其他情况参考实际操作

updateNString

```
void updateNString(int columnIndex, String nString) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNString(String columnLabel, String nString) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateNull

```
void updateNull(int columnIndex) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateNull(String columnLabel) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateObject

```
void updateObject(int columnIndex, Object x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateObject(int columnIndex, Object x, int scaleOrLength) throws
SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateObject(String columnLabel, Object x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY

时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException

- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateObject(String columnLabel, Object x, int scaleOrLength) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateRef

```
void updateRef(int columnIndex, Ref x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateRef(String columnLabel, Ref x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateRow

```
void updateRow() throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateRowId

```
void updateRowId(int columnIndex, RowId x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateRowId(String columnLabel, RowId x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateShort

```
void updateShort(int columnIndex, short x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateShort(String columnLabel, short x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateSQLXML

```
void updateSQLXML(int columnIndex, SQLXML xmlObject) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateSQLXML(String columnLabel, SQLXML xmlObject) throws
SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateString

```
void updateString(int columnIndex, String x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateString(String columnLabel, String x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateTime

```
void updateTime(int columnIndex, Time x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生

SQLFeatureNotSupportedException

- 异常：已关闭时发生SQLException其他情况参考实际操作

```
void updateTime(String columnLabel, Time x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

updateTimestamp

```
void updateTimestamp(int columnIndex, Timestamp x) throws SQLException
```

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

void updateTimestamp(String columnLabel, Timestamp x) throws SQLException

- 操作：尚未实现cursor update功能ResultSet concurrency为ResultSet.CONCUR_READ_ONLY时发生SQLException为ResultSet.CONCUR_UPDATABLE时发生SQLFeatureNotSupportedException
- 异常：已关闭时发生SQLException其他情况参考实际操作

wasNull

```
boolean wasNull() throws SQLException
```

- 操作：询问最后读取的column值是否为NULLNULL时返回true否则返回false
- 异常：ResultSet已关闭或没读取过column值时发生SQLException

isWrapperFor

```
boolean isWrapperFor(Class<?> iface)
```

- 操作：询问此对象是否为实现iface接口的CLASS如果是则返回true否则返回falseSUNDB ResultSet对象不是其他CLASS的wrapper因此不判断wrapper的存在与否只询问是否实现了对应因子CLASS类型
- 异常：不发生

unwrap

```
<T> T unwrap(Class<T> iface)
```

- 操作：SUNDB ResultSet不是其他CLASS的wrapper因此即使unwrap也会返回自己转换为对应类型后返回iface为isWrapperFor() method的因子并返回false时此method发生异常
- 异常：iface不是此对象的类型时（此对象为未执行implements的类型时）发生SQLException

ResultSetMetaData

getCatalogName

```
String getCatalogName(int column) throws SQLException
```

- 操作：返回对应column的catalog名称
- 异常：Column为错误值时发生SQLException

getColumnClassName

```
String getColumnClassName(int column) throws SQLException
```

- 操作：返回最适合对应column类型的Java CLASS名称指定为java.math.BigDecimal等参考Java内部的getName()method与Binary类型相同时参考byte[].class.getName()因此可表示为'[B'等
- 异常：Column为错误值时发生SQLException

getColumnCount

```
int getColumnCount() throws SQLException
```

- 操作：返回ResultSet拥有的column数量
- 异常：不发生

getColumnDisplaySize

```
int getColumnDisplaySize(int column) throws SQLException
```

- 操作：返回输出对应column值时的最大宽度
- 异常：Column为错误值时发生SQLException

getColumnLabel

```
String getColumnLabel(int column) throws SQLException
```

- 操作：获取对应column的label例如"select C1 + 1 from t1"等语句中label为"C1 + 1"name为""
- 异常：Column为错误值时发生SQLException

getColumnName

```
String getColumnName(int column) throws SQLException
```

- 操作：对应column的别名JDBC规格指定获取column的原名而不是别名但各种视图的原名没有意义或复杂因此最好使用别名例如"select C1 as C2 from t1"等语句中name与label均为"C2"name与label不同时参考**getColumnLabel**
- 异常：Column为错误值时发生SQLException

getColumnType

```
int getColumnType(int column) throws SQLException
```

- 操作：返回对应column的类型返回值为types定义的值对SUNDB interval系列类型返回Types.OTHERS对其他类型返回对应types的常数值
- 异常：Column为错误值时发生SQLException

getColumnTypeName

```
String getColumnTypeName(int column) throws SQLException
```

- 操作：返回对应column的SUNDB column类型名
- 异常：Column为错误值时发生SQLException

getPrecision

```
int getPrecision(int column) throws SQLException
```

- 操作：返回对应column的precision 对没有precision的类型返回0
- 异常：Column为错误值时发生SQLException

getScale

```
int getScale(int column) throws SQLException
```

- 操作：返回对应column的scale 对没有scale的类型返回0
- 异常：Column为错误值时发生SQLException

getSchemaName

```
String getSchemaName(int column) throws SQLException
```

- 操作：返回对应column的schema名称
- 异常：Column为错误值时发生SQLException

getTableName

```
String getTableName(int column) throws SQLException
```

- 操作：返回对应column的表名
- 异常：Column为错误值时发生SQLException

isAutoIncrement

```
boolean isAutoIncrement(int column) throws SQLException
```

- 操作：返回对应column是否为被自动赋予固有值的column如果是则返回true否则返回false
- 异常：Column为错误值时发生SQLException

isCaseSensitive

```
boolean isCaseSensitive(int column) throws SQLException
```

- 操作：返回对应column是否区分大小写如果区分则返回true否则返回false
- 异常：Column为错误值时发生SQLException

isCurrency

```
boolean isCurrency(int column) throws SQLException
```

- 操作：服务器无法判断column的currency因此总是返回false
- 异常：Column为错误值时发生SQLException

isDefinitelyWritable

```
boolean isDefinitelyWritable(int column) throws SQLException
```

- 操作：返回对应column的updatable与否SUNDB不支持definitely writable总是返回与isUpdatable()相同的值
- 异常：Column为错误值时发生SQLException

isNullable

```
int isNullable(int column) throws SQLException
```

- 操作：返回对应column是否可以有NULL值返回columnNullable或columnNoNulls中的一个
- 异常：Column为错误值时发生SQLException

isReadOnly

```
boolean isReadOnly(int column) throws SQLException
```

- 操作：返回对应column的read-only与否总是返回isUpdatable()的相反值

- 异常：Column为错误值时发生SQLException

isSearchable

```
boolean isSearchable(int column) throws SQLException
```

- 操作：返回对应column是否可以在条件语句中使用SUNDB的所有的target column均可用于条件语句中因此总是返回true
- 异常：Column为错误值时发生SQLException

isSigned

```
boolean isSigned(int column) throws SQLException
```

- 操作：返回对应column是否有符号有符号时返回true否则返回false
- 异常：Column为错误值时发生SQLException

isWritable

```
boolean isWritable(int column) throws SQLException
```

- 操作：返回对应column是否为updatable
- 异常：Column为错误值时发生SQLException

isWrapperFor

```
boolean isWrapperFor(Class<?> iface)
```

- 操作：询问此对象是否为实现iface接口的CLASS如果是则返回true否则返回falseSUNDB ResultSetMetaData对象不是其他CLASS的wrapper因此不判断wrapper的存在与否只询问是否实现了对应因子CLASS类型
- 异常：不发生

unwrap

```
<T> T unwrap(Class<T> iface)
```

- 操作：SUNDB ResultSetMetaData对象不是其他CLASS的wrapper因此即使unwrap也会返回自己转换为对应类型后返回iface为isWrapperFor() method的因子并返回false时此method发生异常
- 异常：iface不是此对象的类型时（此对象赋予未implement的类型时）发生SQLException

RowId

equals

```
boolean equals(Object obj) throws SQLException
```

- 操作：此对象的RowId与obj相同时返回true否则返回false
- 异常：不发生

getBytes

```
byte[] getBytes() throws SQLException
```

- 操作：返回RowId的byte数组值
- 异常：不发生

hashCode

```
int hashCode() throws SQLException
```

- 操作：返回hash code值
- 异常：不发生

toString

```
String toString() throws SQLException
```

- 操作：返回RowId值的base-64字符串
- 异常：不发生

RowSet

CLASS未实现

addRowSetListener

```
void addRowSetListener(RowSetListener listener) throws SQLException
```

clearParameters

```
void clearParameters() throws SQLException
```

execute

```
void execute() throws SQLException
```

getCommand

```
String getCommand() throws SQLException
```

getDataSourceName

```
String getDataSourceName() throws SQLException
```

getEscapeProcessing

```
boolean getEscapeProcessing() throws SQLException
```

getMaxFieldSize

```
int getMaxFieldSize() throws SQLException
```

getMaxRows

```
int getMaxRows() throws SQLException
```

getPassword

```
String getPassword() throws SQLException
```

getQueryTimeout

```
int getQueryTimeout() throws SQLException
```

getTransactionIsolation

```
int getTransactionIsolation() throws SQLException
```

getTypeMap

```
Map<String,Class<?>> getTypeMap() throws SQLException
```

getUrl

```
String getUrl() throws SQLException
```

getUsername

```
String getUsername() throws SQLException
```

isReadOnly

```
boolean isReadOnly() throws SQLException
```

removeRowSetListener

```
void removeRowSetListener(RowSetListener listener) throws SQLException
```

setArray

```
void setArray(int i, Array x) throws SQLException
```

setAsciiStream

```
void setAsciiStream(int parameterIndex, InputStream x) throws SQLException
```

```
void setAsciiStream(int parameterIndex, InputStream x, int length) throws
SQLException
```

```
void setAsciiStream(String parameterName, InputStream x) throws
SQLException
```

```
void setAsciiStream(String parameterName, InputStream x, int length)
throws SQLException
```

setBigDecimal

```
void setBigDecimal(int parameterIndex, BigDecimal x) throws SQLException
```

```
void setBigDecimal(String parameterName, BigDecimal x) throws SQLException
```

setBinaryStream

```
void setBinaryStream(int parameterIndex, InputStream x) throws
SQLException
```

```
void setBinaryStream(int parameterIndex, InputStream x, int length) throws
SQLException
```

```
void setBinaryStream(String parameterName, InputStream x) throws
SQLException
```

```
void setBinaryStream(String parameterName, InputStream x, int length)
throws SQLException
```

setBlob

```
void setBlob(int i, Blob x) throws SQLException
```

```
void setBlob(int parameterIndex, InputStream inputStream) throws
SQLException
```

```
void setBlob(int parameterIndex, InputStream inputStream, long length)
throws SQLException
```

```
void setBlob(String parameterName, Blob x) throws SQLException
```

```
void setBlob(String parameterName, InputStream inputStream) throws
SQLException
```

```
void setBlob(String parameterName, InputStream inputStream, long length)
throws SQLException
```

setBoolean

```
void setBoolean(int parameterIndex, boolean x) throws SQLException
```

```
void setBoolean(String parameterName, boolean x) throws SQLException
```

setByte

```
void setByte(int parameterIndex, byte x) throws SQLException
```

```
void setByte(String parameterName, byte x) throws SQLException
```

setBytes

```
void setBytes(int parameterIndex, byte[] x) throws SQLException
```

```
void setBytes(String parameterName, byte[] x) throws SQLException
```

setCharacterStream

```
void setCharacterStream(int parameterIndex, Reader reader) throws
SQLException
```

```
void setCharacterStream(int parameterIndex, Reader reader, int length)
throws SQLException
```

```
void setCharacterStream(String parameterName, Reader reader) throws
SQLException
```

```
void setCharacterStream(String parameterName, Reader reader, int length)
throws SQLException
```

setClob

```
void setClob(int i, Clob x) throws SQLException
```

```
void setClob(int parameterIndex, Reader reader) throws SQLException
```

```
void setClob(int parameterIndex, Reader reader, long length) throws
SQLException
```

```
void setClob(String parameterName, Clob x) throws SQLException
```

```
void setClob(String parameterName, Reader reader) throws SQLException
```

```
void setClob(String parameterName, Reader reader, long length) throws
SQLException
```

setCommand

```
void setCommand(String cmd) throws SQLException
```

setConcurrency

```
void setConcurrency(int concurrency) throws SQLException
```

setDataSourceName

```
void setDataSourceName(String name) throws SQLException
```

setDate

```
void setDate(int parameterIndex, Date x) throws SQLException
```

```
void setDate(int parameterIndex, Date x, Calendar cal) throws SQLException
```

```
void setDate(String parameterName, Date x) throws SQLException
```

```
void setDate(String parameterName, Date x, Calendar cal) throws
SQLException
```

setDouble

```
void setDouble(int parameterIndex, double x) throws SQLException
```

```
void setDouble(String parameterName, double x) throws SQLException
```

setEscapeProcessing

```
void setEscapeProcessing(boolean enable) throws SQLException
```

setFloat

```
void setFloat(int parameterIndex, float x) throws SQLException
```

```
void setFloat(String parameterName, float x) throws SQLException
```

setInt

```
void setInt(int parameterIndex, int x) throws SQLException
```

```
void setInt(String parameterName, int x) throws SQLException
```

setLong

```
void setLong(int parameterIndex, long x) throws SQLException
```

```
void setLong(String parameterName, long x) throws SQLException
```

setMaxFieldSize

```
void setMaxFieldSize(int max) throws SQLException
```

setMaxRows

```
void setMaxRows(int max) throws SQLException
```

setNCharacterStream

```
void setNCharacterStream(int parameterIndex, Reader value) throws
SQLException
```

```
void setNCharacterStream(int parameterIndex, Reader value, long length)
throws SQLException
```

```
void setNCharacterStream(String parameterName, Reader value) throws
SQLException
```

```
void setNCharacterStream(String parameterName, Reader value, long length)
throws SQLException
```

setNClob

```
void setNClob(int parameterIndex, NClob value) throws SQLException
```

```
void setNClob(int parameterIndex, Reader reader) throws SQLException
```

```
void setNClob(int parameterIndex, Reader reader, long length) throws
SQLException
```

```
void setNClob(String parameterName, NClob value) throws SQLException
```

```
void setNClob(String parameterName, Reader reader) throws SQLException
```

```
void setNClob(String parameterName, Reader reader, long length) throws
SQLException
```

setNString

```
void setNString(int parameterIndex, String value) throws SQLException
```

```
void setNString(String parameterName, String value) throws SQLException
```

setNull

```
void setNull(int parameterIndex, int sqlType) throws SQLException
```

```
void setNull(int paramIndex, int sqlType, String typeName) throws
SQLException
```

```
void setNull(String parameterName, int sqlType) throws SQLException
```

```
void setNull(String parameterName, int sqlType, String typeName) throws
SQLException
```

setObject

```
void setObject(int parameterIndex, Object x) throws SQLException
```

```
void setObject(int parameterIndex, Object x, int targetSqlType) throws
SQLException
```

```
void setObject(int parameterIndex, Object x, int targetSqlType, int
scaleOrLength) throws SQLException
```

```
void setObject(String parameterName, Object x) throws SQLException
```

```
void setObject(String parameterName, Object x, int targetSqlType) throws
SQLException
```

```
void setObject(String parameterName, Object x, int targetSqlType, int
scale) throws SQLException
```

setPassword

```
void setPassword(String password) throws SQLException
```

setQueryTimeout

```
void setQueryTimeout(int seconds) throws SQLException
```

setReadOnly

```
void setReadOnly(boolean value) throws SQLException
```

setRef

```
void setRef(int i, Ref x) throws SQLException
```

setRowId

```
void setRowId(int parameterIndex, RowId x) throws SQLException
```

```
void setRowId(String parameterName, RowId x) throws SQLException
```

setShort

```
void setShort(int parameterIndex, short x) throws SQLException
```

```
void setShort(String parameterName, short x) throws SQLException
```

setSQLXML

```
void setSQLXML(int parameterIndex, SQLXML xmlObject) throws SQLException
```

```
void setSQLXML(String parameterName, SQLXML xmlObject) throws SQLException
```

setString

```
void setString(int parameterIndex, String x) throws SQLException
```

```
void setString(String parameterName, String x) throws SQLException
```

setTime

```
void setTime(int parameterIndex, Time x) throws SQLException
```

```
void setTime(int parameterIndex, Time x, Calendar cal) throws SQLException
```

```
void setTime(String parameterName, Time x) throws SQLException
```

```
void setTime(String parameterName, Time x, Calendar cal) throws
SQLException
```

setTimestamp

```
void setTimestamp(int parameterIndex, Timestamp x) throws SQLException
```

```
void setTimestamp(int parameterIndex, Timestamp x, Calendar cal) throws
SQLException
```

```
void setTimestamp(String parameterName, Timestamp x) throws SQLException
```

```
void setTimestamp(String parameterName, Timestamp x, Calendar cal) throws
SQLException
```

setTransactionIsolation

```
void setTransactionIsolation(int level) throws SQLException
```

setType

```
void setType(int type) throws SQLException
```

setTypeMap

```
void setTypeMap(Map<String,Class<?>> map) throws SQLException
```

setURL

```
void setURL(int parameterIndex, URL x) throws SQLException
```

setUrl

```
void setUrl(String url) throws SQLException
```

setUsername

```
void setUsername(String name) throws SQLException
```

RowSetMetaData

CLASS未实现

setAutoIncrement

```
void setAutoIncrement(int columnIndex, boolean property) throws
SQLException
```

setCaseSensitive

```
void setCaseSensitive(int columnIndex, boolean property) throws
SQLException
```

setCatalogName

```
void setCatalogName(int columnIndex, String catalogName) throws
SQLException
```

setColumnCount

```
void setColumnCount(int columnCount) throws SQLException
```

setColumnDisplaySize

```
void setColumnDisplaySize(int columnIndex, int size) throws SQLException
```

setColumnLabel

```
void setColumnLabel(int columnIndex, String label) throws SQLException
```

setColumnName

```
void setColumnName(int columnIndex, String columnName) throws SQLException
```

setColumnType

```
void setColumnType(int columnIndex, int SQLType) throws SQLException
```

setColumnTypeName

```
void setColumnTypeName(int columnIndex, String typeName) throws
SQLException
```

setCurrency

```
void setCurrency(int columnIndex, boolean property) throws SQLException
```

setNullable

```
void setNullable(int columnIndex, int property) throws SQLException
```

setPrecision

```
void setPrecision(int columnIndex, int precision) throws SQLException
```

setScale

```
void setScale(int columnIndex, int scale) throws SQLException
```

setSchemaName

```
void setSchemaName(int columnIndex, String schemaName) throws SQLException
```

setSearchable

```
void setSearchable(int columnIndex, boolean property) throws SQLException
```

setSigned

```
void setSigned(int columnIndex, boolean property) throws SQLException
```

setTableName

```
void setTableName(int columnIndex, String tableName) throws SQLException
```

Savepoint

getSavepointId

```
int getSavepointId() throws SQLException
```

- 操作：返回自动被赋予的ID值
- 异常：指定名称后生成Savepoint对象时没有ID因此发生SQLException

getSavepointName

```
String getSavepointName() throws SQLException
```

- 操作：返回生成Savepoint对象时指定的名称
- 异常：以自动ID值生成Savepoint对象时发生SQLException

SQLData

CLASS未实现

getSQLTypeName

```
String getSQLTypeName() throws SQLException
```

readSQL

```
void readSQL(SQLInput stream, String typeName) throws SQLException
```

writeSQL

```
void writeSQL(SQLOutput stream) throws SQLException
```

SQLXML

CLASS未实现

free

```
void free() throws SQLException
```

getBinaryStream

```
InputStream getBinaryStream() throws SQLException
```

getCharacterStream

```
Reader getCharacterStream() throws SQLException
```

getSource

```
<T extends Source> T getSource(Class<T> sourceClass) throws SQLException
```

getString

```
String getString() throws SQLException
```

setBinaryStream

```
OutputStream setBinaryStream() throws SQLException
```

setCharacterStream

```
Writer setCharacterStream() throws SQLException
```

setResult

```
<T extends Result> T setResult(Class<T> resultClass) throws SQLException
```

setString

```
void setString(String value) throws SQLException
```

Statement

addBatch

```
void addBatch(String sql) throws SQLException
```

- 操作：在batch job中添加sql语句
- 异常：不发生

cancel

```
void cancel() throws SQLException
```

- 操作：不支持
- 异常：总是返回SQLFeatureNotSupportedException

clearBatch

```
void clearBatch() throws SQLException
```

- 操作：清除所有登记的batch job没有batch job时不做任何操作
- 异常：不发生

clearWarnings

```
void clearWarnings() throws SQLException
```

- 操作：清除Statement对象拥有的所有SQLWarning对象
- 异常：不发生

close

```
void close() throws SQLException
```

- 操作：关闭当前statement对象解除服务器中分配的statement相关信息如果有此对象生成的ResultSet则全部关闭从生成此Statement对象的connection对象中解除此对象
- 异常：从服务器解除statement信息时报错则发生异常

execute

```
boolean execute(String sql) throws SQLException
```

- 操作：执行sql如果执行的sql有ResultSet则返回true否则返回false
- 异常：Statement已关闭或登记了batch job或执行过程中服务器报错时发生异常

```
boolean execute(String sql, int autoGeneratedKeys) throws SQLException
```

- 操作：执行sql如果autoGeneratedKey为Statement.RETURN_GENERATED_KEYSsql是INSERT语句时可获得查询自动生成key的ResultSet除此之外和execute(String sql)相同
- 异常：Statement已关闭或登记了batch job或执行过程中服务器中报错时发生异常

```
boolean execute(String sql, int[] columnIndexes) throws SQLException
```

- 操作：执行sqlcolumnIndexes不为nullsql是INSERT语句时可获得使用指定的索引查询自动

生成key的ResultSet除此之外与execute(String sql)相同

- 异常：Statement已关闭或登记了batch job或执行过程中服务器报错时发生异常

```
boolean execute(String sql, String[] columnNames) throws SQLException
```

- 操作：执行sqlcolumnNames不为nullsql是INSERT语句时可获得使用指定的列名自动生成key的ResultSet除此之外与execute(String sql)相同
- 异常：Statement已关闭或登记了batch job或执行过程中服务器中报错时发生异常

executeBatch

```
int[] executeBatch() throws SQLException
```

- 操作：按照顺序execute登记的batchjob每一个batch job均与服务器产生通讯每次执行batch job后返回反映更新的行数量的数组
- 异常：Statement已关闭或没有batch job或执行过程中服务器报错时发生异常

> Note:
>
> 由于不是一次性将batch job传送给服务器并执行的结构因此与普通execute()相比在性能上没有很大的优势为了快速处理请使用PreparedStatement的batch execution

executeQuery

```
ResultSet executeQuery(String sql) throws SQLException
```

- 操作：执行指定的sql语句收到部分结果后生成ResultSet

- 异常：Statement已关闭或登记了batch job或执行过程中服务器报错或sql不是select语句时发生异常

> Caution:
>
> 与execute()的操作稍微不同对同样的sql语句执行execute()后执行getResultSet()时与服务器产生两次通讯execute()时执行执行命令getResultSet()时执行fetch相关命令相反executeQuery()假设sql语句就是select语句包含fetch通过一次通讯执行所有

executeUpdate

```
int executeUpdate(String sql) throws SQLException
```

- 操作：执行sql语句返回执行时更新的行数量
- 异常：Statement已关闭或登记了batch job或执行过程中服务器报错或sql语句为select语句时发生异常

```
int executeUpdate(String sql, int autoGeneratedKeys) throws SQLException
```

- 操作：执行sql语句返回执行时更新的行数量autoGeneratedKeys为Statement.RETURN_GENERATED_KEYSsql是INSERT语句时可获得查询自动生成key的ResultSet除此之外与executeUpdate(String sql)相同
- 异常：Statement已关闭或登记了batch job或执行过程中服务器报错或sql语句为select语句时发生异常

```
int executeUpdate(String sql, int[] columnIndexes) throws SQLException
```

- 操作：执行sql返回执行时更新的行数量columnIndexes不为nullsql是INSERT语句时可获得使用指定的索引查询自动生成key的ResultSet除此之外与executeUpdate(String sql)相同
- 异常：Statement已关闭或登记了batch job或执行过程中服务器报错或sql为select语句时发生异常

```
int executeUpdate(String sql, String[] columnNames) throws SQLException
```

- 操作：执行sql语句返回执行时更新的行数量columnNames不为nullsql是INSERT语句时可获得使用指定的列名查询自动生成key的ResultSet除此之外与executeUpdate(String sql)相同
- 异常：Statement已关闭或登记了batch job或执行过程中服务器报错或sql为select语句时发生异常

getConnection

```
Connection getConnection() throws SQLException
```

- 操作：返回生成此对象的connection对象使用通过PooledConnection获取的logical connection生成 Statement对象时用户通过此method获取logical connection而不是physical connection
- 异常：Statement已关闭时发生异常

getExplainPlan

```
String getExplainPlan() throws SQLException
```

- 操作：非标准的method是SUNDBStatement的固有method获取生成的plan text为了使用此

method需要通过setExplainPlanOption() method设置为可生成plan text详细的用法参考查看Plan Text

- 异常：Statement已关闭或服务器报错时发生异常

getExplainPlanOption

```
int getExplainPlanOption() throws SQLException
```

- 操作：非标准的method是SundbStatement的固有method获取当前设置的plan text生成相关选项返回值为以下值中的一个默认值为SundbStatement.EXPLAIN_PLAN_OPTION_OFF
  - SundbStatement.EXPLAIN_PLAN_OPTION_OFF
  - SundbStatement.EXPLAIN_PLAN_OPTION_ON
  - SundbStatement.EXPLAIN_PLAN_OPTION_ON_VERBOSE
  - SundbStatement.EXPLAIN_PLAN_OPTION_ONLY
- 异常：不发生

getFetchDirection

```
int getFetchDirection() throws SQLException
```

- 操作：总是返回ResultSet.FETCH_FORWARD
- 异常：Statement已关闭时发生异常

getFetchSize

```
int getFetchSize() throws SQLException
```

- 操作：返回从此statement对象获取的ResultSet的默认fetch size默认值为00表示服务器自动决定要fetch的行数量详细内容参考ResultSet的**getFetchSize**
- 异常：Statement已关闭时发生异常

getGeneratedKeys

```
ResultSet getGeneratedKeys() throws SQLException
```

- 操作：返回可查询此statement对象执行结果的自动生成key的ResultSetstatement对象不生成key时将返回关闭的ResultSet对象
- 异常：Statement已经关闭或者服务器发生了错误时将发生异常

getMaxFieldSize

```
int getMaxFieldSize() throws SQLException
```

- 操作：返回max field size此值限制column的最大长度fetch时如果column的值大于此长度则剩余的值将被截断默认值为00表示不限制最大长度
- 异常：Statement已关闭时发生异常

getMaxRows

```
int getMaxRows() throws SQLException
```

- 操作：返回max rowsmax rows为从Statement获取的ResultSet的最大行数量最大行数量以上的行将被忽略默认值为00表示没有限制
- 异常：Statement已关闭时发生异常

getMoreResults

```
boolean getMoreResults() throws SQLException
```

- 操作：移动到Statement的下一结果当前的ResultSet将被关闭
- 异常：Statement已关闭时发生异常

```
boolean getMoreResults(int current) throws SQLException
```

- 操作：移动到Statement的下一结果根据当前值处理当前ResultSet仅支持Statement.CLOSE_CURRENT_RESULT和Statement.CLOSE_ALL_RESULTS
- 异常：Statement已关闭时发生异常

getQueryTimeout

```
int getQueryTimeout() throws SQLException
```

- 操作：获取query timeout值此值为服务器execution时适用的timeout值执行时间超过此值时将取消execution用户收到timeout相关报错单位为秒用户不特殊设置时获取会话的默认值未通过参数指定会话的默认值时为00表示没有限制
- 异常：Statement已关闭时发生异常

getResultSet

```
ResultSet getResultSet() throws SQLException
```

- 操作：对当前执行的execution执行fetch收到fetch结果的一部分后生成ResultSet并返回

JDBC规格指定每次execution只调用一次此method但实际上是调用多次此method也会返回同样的对象

- 异常：Statement已关闭时发生异常Fetch时服务器中报错时发生异常

getResultSetConcurrency

```
int getResultSetConcurrency() throws SQLException
```

- 操作：返回ResultSet concurrency此值决定从此对象生成的ResultSet的concurrency默认值为ResultSet.CONCUR_READ_ONLY 目前暂不支持updatable cursor
- 异常：Statement已关闭时发生异常

getResultSetHoldability

```
int getResultSetHoldability() throws SQLException
```

- 操作：返回ResultSet holdability此值决定从此对象生成的ResultSet的holdability默认值为ResultSet.HOLD_CURSORS_OVER_COMMIT
- 异常：Statement已关闭时发生异常

getResultSetType

```
int getResultSetType() throws SQLException
```

- 操作：返回ResultSet type此值决定从此对象生成的ResultSet的type默认值为ResultSet.TYPE_FORWARD_ONLY
- 异常：Statement已关闭时发生异常

getUpdateCount

```
int getUpdateCount() throws SQLException
```

- 操作：返回最后执行的execution更新的行数量如果最后执行的sql语句不是UPDATEINSERT语句时返回 -1
- 异常：Statement已关闭时不发生

getUpdateRowCount

```
long getUpdateRowCount() throws SQLException
```

- 操作：与getUpdateCount相同但返回类型为long为非标准method需要转换为SundbStatement后才能使用
- 异常：不发生

getWarnings

```
SQLWarning getWarnings() throws SQLException
```

- 操作：返回此对象累积的SQLWarning 没有则返回null
- 异常：不发生

isClosed

```
boolean isClosed() throws SQLException
```

- 操作：返回此Statement是否被关闭关闭时返回true否则返回false除了用户明确调用close()外也可以被服务器或connection对象关闭
- 异常：不发生

isPoolable

```
boolean isPoolable() throws SQLException
```

- 操作：返回该对象是否可进行statement pooling
- 异常：不发生

setCursorName

```
void setCursorName(String name) throws SQLException
```

- 操作：设置由当前执行的statement生成的游标名
- 异常：如果设置游标名时服务器中报错则发生异常

setEscapeProcessing

```
void setEscapeProcessing(boolean enable) throws SQLException
```

- 操作：sql语句的escape由服务器的parser部分处理因此JDBC无法屏蔽此功能不做任何操作
- 异常：Statement已关闭时发生异常

setExplainPlanOption

```
void setExplainPlanOption(int option) throws SQLException
```

- 操作：非标准的method是SundbStatement的固有method指定plan text生成选项选项应设置为以下值中的一个意义如下
  - SundbStatement.EXPLAIN_PLAN_OPTION_OFF：不生成plan text
  - SundbStatement.EXPLAIN_PLAN_OPTION_ON：执行时生成plan text
  - SundbStatement.EXPLAIN_PLAN_OPTION_ON_VERBOSE：执行时生成更详细的plan text
  - SundbStatement.EXPLAIN_PLAN_OPTION_ONLY：执行时生成plan text但实际不执行
- 异常：设置非以上4个值中的一个时发生异常

setFetchDirection

```
void setFetchDirection(int direction) throws SQLException
```

- 操作：设置fetch方向SUNDB仅支持forward fetch因此direction不是ResultSet.FETCH_FORWARD时发生异常
- 异常：Statement已关闭或direction不是FETCH_FORWARD时发生异常

setFetchSize

```
void setFetchSize(int rows) throws SQLException
```

- 操作：设置从此Statement对象获得的ResultSet的默认fetch size默认值为00表示服务器自动决定fetch时的行数量详细内容参考ResultSet的**getFetchSize**

- 异常：Statement已关闭时发生异常

setMaxFieldSize

```
void setMaxFieldSize(int max) throws SQLException
```

- 操作：设置max field size此值限制column的最大长度fetch时如果column的值超过此长度则剩余数据被截断默认值为00表示没有限制只对CHARVARCHARLONG VARCHARBINARYVARBINARYLONG VARBINARY类型有效
- 异常：Statement已关闭时发生异常

setMaxRows

```
void setMaxRows(int max) throws SQLException
```

- 操作：设置max rowsmax rows表示从Statement获取的ResultSet的最大行数最大行数量以上的行将被忽略默认值为00表示没有限制
- 异常：Statement已关闭时发生异常

setPoolable

```
void setPoolable(boolean poolable) throws SQLException
```

- 操作：设置是否进行statement pooling
- 异常：不发生

setQueryTimeout

```
void setQueryTimeout(int seconds) throws SQLException
```

- 操作：设置query timeout值此值为服务器execution时适用的timeout值执行时间超过此时间时将取消execution用户收到timeout相关报错单位为秒用户未特殊设置时适用会话的默认值未通过参数指定会话的默认值时为00表示没有限制
- 异常：Statement已关闭时发生异常

isWrapperFor

```
boolean isWrapperFor(Class<?> iface) throws SQLException
```

- 操作：询问此对象是否为实现iface接口的CLASS如果是则返回true否则返回falseSUNDB statement对象不是其他CLASS的wrapper因此不判断wrapper的存在与否仅询问是否实现了对应因子CLASS类型
- 异常：不发生

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- 操作：SUNDB statement对象不是其他CLASS的wrapper因此即使unwrap也会返回自己转换为对应类型后返回iface为isWrapperFor() method的因子并返回false时此method发生异常
- 异常：iface不是此对象的类型时（传此对象未implements的类型时）发生SQLException

Struct

CLASS未实现

getAttributes

```
Object[] getAttributes() throws SQLException
```

getAttributes

```
Object[] getAttributes(Map<String,Class<?>> map) throws SQLException
```

getSQLTypeName

```
String getSQLTypeName() throws SQLException
```

XAConnection

getXAResource

```
XAResource getXAResource() throws SQLException
```

- 操作：返回可执行XA命令的XAResource对象多次调用method时总是返回同样的结果
- 异常：不发生

XADataSource

getXAConnection

```
XAConnection getXAConnection() throws SQLException
```

- 操作：生成并返回XAConnection对象Connection所需的信息为非标准method需提前设置
- 异常：与服务器连接失败时发生SQLException

```
XAConnection getXAConnection(String user, String password) throws
SQLException
```

- 操作：通过username与password打开新的XAConnection对象并返回Connection所需的其余信息为非标准method需提前设置
- 异常：与服务器连接失败时发生SQLException

XAResource

commit

```
void commit(Xid xid, boolean onePhase) throws XAException
```

- 操作：对全局事务xid执行XA commit命令onePhase设置为true时执行one phase commit
- 异常：执行结果报错时发生XAException

end

```
void end(Xid xid, int flags) throws XAException
```

- 操作：对全局事务xid执行XA end命令flags为TMSUCCESSTMFAIL或TMSUSPEND中的一个
- 异常：执行结果报错时发生XAException

forget

```
void forget(Xid xid) throws XAException
```

- 操作：对全局事务xid执行XA forget命令
- 异常：执行结果报错时发生XAException

getTransactionTimeout

```
int getTransactionTimeout() throws XAException
```

- 操作：SUNDB不支持事务超时总是返回0
- 异常：不发生

isSameRM

```
boolean isSameRM(XAResource xares) throws XAException
```

- 操作：生成XAResource对象时会有固有的rmid用此rmid判断是否为同一个XAResource对象
- 异常：不发生

prepare

```
int prepare(Xid xid) throws XAException
```

- 操作：对全局事务xid执行XA prepare命令
- 异常：执行结果报错时发生XAException

recover

```
Xid[] recover(int flag) throws XAException
```

- 操作：用指定的flag执行XA recover命令返回prepare的事务branch的数组flag为TMSTARTRSCANTMENDRSCANTMNOFLAGTMNOFLAGS中的一个值
- 异常：执行结果报错时发生XAException

rollback

```
void rollback(Xid xid) throws XAException
```

- 操作：对全局事务xid执行XA rollback命令
- 异常：执行结果报错时发生XAException

setTransactionTimeout

```
boolean setTransactionTimeout(int seconds) throws XAException
```

- 操作：SUNDB不支持事务超时不做任何操作
- 异常：不发生

start

```
void start(Xid xid, int flags) throws XAException
```

- 操作：用指定的flag开始全局事务flag可以为TMNOFLAGSTMJOINTMRESUME中的一个值
- 异常：执行结果报错时发生XAException

SUNDBInterval

使用SundbInterval对象赋予SUNDB的Column值需参考使用添加类型

createIntervalYear

```
public static SUNDBInterval createIntervalYear(int yearPrecision, boolean
sign, int year) throws SQLException
```

- 操作：用指定的year值生成SundbInterval对象yearPrecision为year的位数year的值应大于0时间为正数时sign为true负数时为false
- 异常：指定的year值超出yearPrecision时发生异常

```
public static SUNDBInterval createIntervalYear(int yearPrecision, String
year) throws SQLException
```

- 操作：用指定的year值生成SundbInterval对象 yearPrecision为year的位数year的值应为大于0的正数
- 异常：指定的year值超过yearPrecision时发生异常

createIntervalMonth

```
public static SUNDBInterval createIntervalMonth(int monthPrecision,
boolean sign, int month) throws SQLException
```

- 操作：用指定的month值生成SundbInterval对象monthPrecision为month的位数month的值

应大于0时间为正数时sign为true负数时为false

- 异常：指定的month值超过monthPrecision时发生异常

```
public static SUNDBInterval createIntervalMonth(int monthPrecision, String
month) throws SQLException
```

- 操作：用指定的month值生成SundbInterval对象 monthPrecision为month的位数month的值应大于0的正数
- 异常：指定的month值超过monthPrecision时发生异常

createIntervalYearToMonth

```
public static SUNDBInterval createIntervalYearToMonth(int yearPrecision,
boolean sign, int year, int month) throws SQLException
```

- 操作：用指定的yearmonth值生成SundbInterval对象yearPrecision为year的位数yearmonth的值应大于0时间为正数时sign为true负数时为false
- 异常：指定的year值超过yearPrecision时发生异常

```
public static SUNDBInterval createIntervalYearToMonth(int yearPrecision,
String yearToMonth) throws SQLException
```

- 操作：用指定的yearToMonth值生成SundbInterval对象 yearPrecision为year的位数yearToMonth应满足"yy-mm"模式
- 异常：指定的year值超过yearPrecision时发生异常

createIntervalDay

```
public static SUNDBInterval createIntervalDay(int dayPrecision, boolean
sign, int day) throws SQLException
```

- 操作：用指定的day值生成SundbInterval对象dayPrecision为day的位数day的值应大于0时间为正数时sign为true负数时为false
- 异常：指定的day值超过dayPrecision时发生异常

```
public static SUNDBInterval createIntervalDay(int dayPrecision, String
day) throws SQLException
```

- 操作：用指定的day值生成SundbInterval 对象 dayPrecision为day的位数day值应为大于0的正数
- 异常：指定的day值超过dayPrecision时发生异常

createIntervalHour

```
public static SUNDBInterval createIntervalHour(int hourPrecision, boolean
sign, int hour) throws SQLException
```

- 操作：用指定的hour值生成SundbInterval对象hourPrecision为hour的位数hour的值应大于0时间为正数时sign为true负数时为false
- 异常：指定的hour值超过hourPrecision时发生异常

```
public static SUNDBInterval createIntervalHour(int hourPrecision, String
hour) throws SQLException
```

- 操作：用指定的hour值生成SundbInterval对象 hourPrecision为hour的位数hour的值应为大于0的正数
- 异常：指定的hour值超过hourPrecision时发生异常

createIntervalMinute

```
public static SUNDBInterval createIntervalMinute(int minutePrecision,
boolean sign, int minute) throws SQLException
```

- 操作：用指定的minute值生成SundbInterval对象minutePrecision为minute的位数minute的值应大于0时间为正数时sign为true负数时为false
- 异常：指定的minute值超过minutePrecision时发生异常

```
public static SUNDBInterval createIntervalMinute(int minutePrecision,
String minute) throws SQLException
```

- 操作：用指定的minute值生成SundbInterval对象minutePrecision为minute的位数minute的值应为大于0的正数
- 异常：指定的minute值超过minutePrecision时发生异常

createIntervalSecond

```
public static SUNDBInterval createIntervalSecond(int secondPrecision, int
fractionalPrecision, boolean sign, int second, int microsecond) throws
SQLException
```

- 操作：用指定的second值生成SundbInterval对象secondPrecision为second的位数

fractionalPrecision为microsecond的位数second和microsecond的值应大于0时间为正数时sign为true负数时为false

- 异常：指定的second值超过secondPrecision或microSecond值超过fractionalPrecision时发生异常

```
public static SUNDBInterval createIntervalSecond(int secondPrecision, int fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond) throws SQLException
```

- 操作：用指定的dayhourminutesecondmicrosecond值生成SundbInterval对象 secondPrecision是将dayhourminutesecond换算为second时second的位数 fractionalPrecision为microsecond的位数dayhourminutesecondmicrosecond的值应大于0时间为正数时sign为true负数时为false
- 异常：换算的second值超过secondPrecision或microSecond值超过fractionalPrecision时发生异常

```
public static SUNDBInterval createIntervalSecond(int secondPrecision, int fractionalPrecision, String second) throws SQLException
```

- 操作：用指定的second字符串生成SundbInterval对象secondPrecision为second的位数 fractionalPrecision为microsecond的位数second字符串格式应为"dd hh:mm:ss.ffffff"或"dd hh:mm:ss""dd hh:mm""dd hh""hh:mm""ss.ffffff""ss"模式中的一个
- 异常：换算的second值超过secondPrecision或microSecond值超过fractionalPrecision时发生异常指定的字符串不满足格式时也发生异常

createIntervalDayToHour

```
public static SUNDBInterval createIntervalDayToHour(int dayPrecision,
boolean sign, int day, int hour) throws SQLException
```

- 操作：用指定的dayhour值生成SundbInterval对象 dayPrecision为day的位数dahour的值应大于0时间为正数时sign为true负数时为false
- 异常：指定的day值超过dyaPrecision时发生异常

```
public static SUNDBInterval createIntervalDayToHour(int dayPrecision,
String dayToHour) throws SQLException
```

- 操作：用指定的dayToHour字符串生成SundbInterval对象dayPrecision为day的位数dayToHour应满足"dd hh"格式或"dd hh:mm:ss"等格式时除dd, hh外其他均应为0
- 异常：指定的day值超过dayPrecision或指定的字符串不满足格式时发生异常

createIntervalDayToMinute

```
public static SUNDBInterval createIntervalDayToMinute(int dayPrecision,
boolean sign, int day, int hour, int minute) throws SQLException
```

- 操作：用指定的dayhourminute值生成SundbInterval对象dayPrecision为day的位数Dayhourminute的值应大于0时间为正数时sign为true负数时为false
- 异常：指定的day值超过dyaPrecision时发生异常

```
public static SUNDBInterval createIntervalDayToMinute(int dayPrecision,
String dayToMinute) throws SQLException
```

- 操作：用指定的dayToMinute字符串生成SundbInterval对象 dayPrecision为day的位数 dayToMinute应满足"dd hh:mm"格式或"dd hh:mm:ss"等格式时除ddhhmm外其他均应为0
- 异常：指定的day值超过dayPrecision或指定的字符串不满足格式时发生异常

createIntervalDayToSecond

```
public static SUNDBInterval createIntervalDayToSecond(int dayPrecision,
int fractionalPrecision, boolean sign, int day, int hour, int minute, int
second, int microsecond) throws SQLException
```

- 操作：用指定的dayhourminutesecondmicrosecond值生成SundbInterval对象dayPrecision为day的位数fractionalPrecision为microsecond的位数dayhourminutesecondmicrosecond的值应大于0时间为正数时sign为true负数时为false
- 异常：Day值超过dayPrecision或microSecond值超过fractionalPrecision时发生异常

```
public static SUNDBInterval createIntervalDayToSecond(int dayPrecision,
String dayToSecond) throws SQLException
```

- 操作：用指定的dayToSecond字符串生成SundbInterval对象dayPrecision为day的位数 fractionalPrecision为microsecond的位数dayToSecond应满足"dd hh:mm:ss.ffffff"dd hh:mm:ss"hh:mm:ss"hh:mm"格式中的一个
- 异常：Day值或换算的day值超过dayPrecision或microSecond值超过fractionalPrecision或字符串不满足格式时发生异常

createIntervalHourToMinute

```
public static SUNDBInterval createIntervalHourToMinute(int hourPrecision,
```

```
boolean sign, int hour, int minute) throws SQLException
```

- 操作：用指定的houminute值生成SundbInterval对象hourPrecision为hour的位数hourminute的值应大于0时间为正数时sign为ture负数时为false
- 异常：Hour值超过hourPrecision时发生异常

```
public static SUNDBInterval createIntervalHourToMinute(int hourPrecision,
boolean sign, int day, int hour, int minute) throws SQLException
```

- 操作：用指定的dayhourminute值生成SundbInterval对象hourPrecision为hour的位数dayhourminute的值应大于0时间为正数时sign为true负数时为false
- 异常：Hour值或换算的hour值超过hourPrecision时发生异常

```
public static SUNDBInterval createIntervalHourToMinute(int hourPrecision,
String hourToMinute) throws SQLException
```

- 操作：用指定的hourToMinute字符串生成SundbInterval对象hourPrecision为hour的位数hourToMinute应满足"hh:mm"格式或其他格式时ss或ffffff的值均应为0
- 异常：Hour值或换算的hour值超过hourPrecision时发生异常指定的字符串不满足格式时发生异常

createIntervalHourToSecond

```
public static SUNDBInterval createIntervalHourToSecond(int hourPrecision,
int fractionalPrecision, boolean sign, int hour, int minute, int second,
int microsecond) throws SQLException
```

- 操作：用指定的hourminutesecondmicrosecond值生成SundbInterval 对象hourPrecision为hour的位数fractionalPrecision为microsecond的位数hourminutesecondmicrosecond的值应大于0时间为正数时sign为true负数时为false
- 异常：Hour值超过hourPrecision或microSecond值超过fractionalPrecision时发生异常

```
public static SUNDBInterval createIntervalHourToSecond(int hourPrecision, int fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond) throws SQLException
```

- 操作：用指定的dayhourminutesecondmicrosecond值生成SundbInterval对象hourPrecision为hour的位数fractionalPrecision为microsecond的位数Dayhourminutesecondmicrosecond的值应大于0时间为正数时sign为true负数时为false
- 异常：Hour值或换算的hour值超过hourPrecision或microSecond值超过fractionalPrecision时发生异常

```
public static SUNDBInterval createIntervalHourToSecond(int hourPrecision, int fractionalPrecision, String hourToSecond) throws SQLException
```

- 操作：用指定的hourToSecond字符串生成SundbInterval对象hourPrecision为hour的位数fractionalPrecision为microsecond的位数hourToSecond应满足"hh:mm:ss.ffffff""hh:mm:ss""hh:mm:ss""hh:mm"格式中的一个包含dd时换算的hour值不能超过hourPrecision
- 异常：Hour值或换算的hour值超过hourPrecision或microSecond值超过fractionalPrecision或指定的字符串不满足格式时发生异常

createIntervalMinuteToSecond

```
public static SUNDBInterval createIntervalMinuteToSecond(int
minutePrecision, int fractionalPrecision, boolean sign, int minute, int
second, int microsecond) throws SQLException
```

- 操作：用指定的 minutesecondmicrosecond值生成SundbInterval对象minutePrecision为minute的位数fractionalPrecision为microsecond的位数minutesecondmicrosecond的值应大于0时间为正数时sign为true负数时为false
- 异常：Minute值超过minutePrecision或microSecond值超过fractionalPrecision时发生异常

```
public static SUNDBInterval createIntervalMinuteToSecond(int
minutePrecision, int fractionalPrecision, boolean sign, int day, int hour,
int minute, int second, int microsecond) throws SQLException
```

- 操作：用指定的dayhourminutesecondmicrosecond值生成SundbInterval对象minutePrecision为minute的位数fractionalPrecision为microsecond的位数dayhourminutesecondmicrosecond的值应大于0时间为正数时sign为true负数时为false
- 异常：Minute值或换算的minute值超过minutePrecision或microSecond值超过fractionalPrecision时发生异常

```
public static SUNDBInterval createIntervalMinuteToSecond(int
minutePrecision, int fractionalPrecision, String minuteToSecond) throws
SQLException
```

- 操作：用指定的minuteToSecond字符串生成SundbInterval对象minutePrecision为minute的

位数fractionalPrecision为microsecond的位数minuteToSecond应满足"mm:ss.ffffff"或"mm:ss"格式包含dd或hh时换算的minute值不能超过minutePrecision

- 异常：Minute值或换算的minute值超过minutePrecision或microSecond值超过fractionalPrecision或指定的字符串不满足格式时发生异常

getSign

```
public int getSign()
```

- 操作：时间为正数时返回1负数时返回-1
- 异常：不发生

getYear

```
public int getYear()
```

- 操作：返回year值Interval对象的负数与否并非通过getYear()返回
- 异常：不发生

getMonth

```
public int getMonth()
```

- 操作：返回month值Interval对象的负数与否并非通过getMonth()返回
- 异常：不发生

getAccumulatedMonth

```
public int getAccumulatedMonth()
```

- 操作：返回将year与month换算为month的值
- 异常：不发生

getDay

```
public int getDay()
```

- 操作：返回day值Interval对象的负数与否并非通过getDay()返回
- 异常：不发生

getHour

```
public int getHour()
```

- 操作：返回hour值Interval对象的负数与否并非通过getHour()返回
- 异常：不发生

getAccumulatedHour

```
public int getAccumulatedHour()
```

- 操作：返回将day与hour换算为hour的值
- 异常：不发生

getMinute

```
public int getMinute()
```

- 操作：返回minute值Interval对象的负数与否并非通过getMinute()返回
- 异常：不发生

getAccumulatedMinute

```
public int getAccumulatedMinute()
```

- 操作：返回将dayhourminute换算为minute的值
- 异常：不发生

getSecond

```
public int getSecond()
```

- 操作：返回second值Interval对象的负数与否并非通过getSecond()返回
- 异常：不发生

getAccumulatedSecond

```
public int getAccumulatedSecond()
```

- 操作：返回将dayhourminutesecond换算为second的值
- 异常：不发生

getMicroSecond

```
public int getMicroSecond()
```

- 操作：返回microsecond值Interval对象的负数与否并非通过getMicroSecond()返回
- 异常：不发生

getAccumulatedMicroSecond

```
public long getAccumulatedMicroSecond()
```

- 操作：返回将dayhourminutesecondmicrosecond换算为microsecond的值
- 异常：不发生

getTypeName

```
public String getTypeName()
```

- 操作：返回类型名称
- 异常：不发生

getSqlType

```
public int getSqlType()
```

- 操作：以SundbTypes定义的类型常数返回对应此对象的类型
- 异常：不发生

toString

```
public String toString()
```

- 操作：以字符串返回此对象表示的interval值
- 异常：不发生

SUNDBTypes

常数定义

```
public static final int INTERVAL_YEAR;

public static final int INTERVAL_MONTH;

public static final int INTERVAL_DAY;

public static final int INTERVAL_HOUR;

public static final int INTERVAL_MINUTE;

public static final int INTERVAL_SECOND;

public static final int INTERVAL_YEAR_TO_MONTH;

public static final int INTERVAL_DAY_TO_HOUR;

public static final int INTERVAL_DAY_TO_MINUTE;

public static final int INTERVAL_DAY_TO_SECOND;

public static final int INTERVAL_HOUR_TO_MINUTE;

public static final int INTERVAL_HOUR_TO_SECOND;

public static final int INTERVAL_MINUTE_TO_SECOND;

public static final int TIME_WITH_TIME_ZONE;

public static final int TIMESTAMP_WITH_TIME_ZONE;

public static final int REF_CURSOR;
```

这些常数的用法与java.sql.Types的用法相同即PreparedStatement的setObject或ResultSet的getObject中指定类型时使用JDBC标准未定义这些类型因此通过SUNDBTypes单独提供

类型转换

通过下表说明类型转换

SQL类型	SUNDB类型
SundbTypes.INTERVAL_DAY	INTERVAL DAY
SundbTypes.INTERVAL_DAY_TO_HOUR	INTERVAL DAY TO HOUR
SundbTypes.INTERVAL_DAY_TO_MINUTE	INTERVAL DAY TO MINUTE
SundbTypes.INTERVAL_DAY_TO_SECOND	INTERVAL DAY TO SECOND
SundbTypes.INTERVAL_HOUR	INTERVAL HOUR
SundbTypes.INTERVAL_HOUR_TO_MINUTE	INTERVAL HOUR TO MINUTE
SundbTypes.INTERVAL_HOUR_TO_SECOND	INTERVAL HOUR TO SECOND
SundbTypes.INTERVAL_MINUTE	INTERVAL MINUTE
SundbTypes.INTERVAL_MINUTE_TO_SECOND	INTERVAL MINUTE TO SECOND
SundbTypes.INTERVAL_MONTH	INTERVAL MONTH
SundbTypes.INTERVAL_SECOND	INTERVAL SECOND
SundbTypes.INTERVAL_YEAR	INTERVAL YEAR
SundbTypes.INTERVAL_YEAR_TO_MONTH	INTERVAL YEAR TO MONTH
SundbTypes.REF_CURSOR	REF CURSOR
SundbTypes.TIME_WITH_TIME_ZONE	TIME WITH TIME ZONE

SQL类型	SUNDB类型
SundbTypes.TIMESTAMP_WITH_TIME_ZONE	TIMESTAMP WITH TIME ZONE
Types.ARRAY	N/A
Types.BIGINT	NATIVE_BIGINT
Types.BINARY	BINARY(2000)
Types.BIT	BOOLEAN
Types.BLOB	LONG VARBINARY
Types.BOOLEAN	BOOLEAN
Types.CHAR	CHAR(2000)
Types.CLOB	LONG VARCHAR
Types.DATALINK	N/A
Types.DATE	DATE
Types.DECIMAL	DECIMAL
Types.DISTINCT	N/A
Types.DOUBLE	NATIVE_DOUBLE
Types.FLOAT	FLOAT
Types.INTEGER	NATIVE_INTEGER
Types.JAVA_OBJECT	N/A
Types.LONGNVARCHAR	N/A

SQL类型	SUNDB类型
Types.LONGVARBINARY	LONG VARBINARY
Types.LONGVARCHAR	LONG VARCHAR
Types.NCHAR	N/A
Types.NCLOB	N/A
Types.NUMERIC	NUMBER
Types.NVARCHAR	N/A
Types.OTHER	N/A
Types.REAL	NATIVE_REAL
Types.REF	N/A
Types.REF_CURSOR	REF CURSOR
Types.ROWID	ROWID
Types.SMALLINT	NATIVE_SMALLINT
Types.SQLXML	N/A
Types.STRUCT	N/A
Types.TIME	TIME
Types.TIME_WITH_TIMEZONE	TIME WITH TIME ZONE
Types.TIMESTAMP	TIMESTAMP
Types.TIMESTAMP_WITH_TIMEZONE	TIMESTAMP WITH TIME ZONE

SQL类型	SUNDB类型
Types.TINYINT	NATIVE_SMALLINT
Types.VARBINARY	VARBINARY(4000)
Types.VARCHAR	VARCHAR(4000)

Table 3-3 SQL类型 → SUNDB类型

	NATIVE_SMALLINT	NATIVE_INTEGER	NATIVE_BIGINT	NATIVE_REAL	NATIVE_DOUBL
getByte	0	0	0	0	0
getShort	0	0	0	0	0
getInt	0	0	0	0	0
getLong	0	0	0	0	0
getFloat	0	0	0	0	0
getDouble	0	0	0	0	0
getBigDecimal	0	0	0	0	0
getBoolean	只有01时可行	只有01时可行	只有01时可行	只有01时可行	只有01时可行
getString	0	0	0	0	0
getBytes	raw数据	raw数据	raw数据	raw数据	raw数据
getDate	X	X	X	X	X
getTime	X	X	X	X	X

	NATIVE_SMALLINT	NATIVE_INTEGER	NATIVE_BIGINT	NATIVE_REAL	NATIVE_DOUBL
getTimestamp	X	X	X	X	X
getAsciiStream	raw数据	raw数据	raw数据	raw数据	raw数据
getBinaryStream	raw数据	raw数据	raw数据	raw数据	raw数据
getCharacterStream	X	X	X	X	X
getClob	O	O	O	O	O
getBlob	raw数据	raw数据	raw数据	raw数据	raw数据
getArray	X	X	X	X	X
getRef	X	X	X	X	X
getURL	X	X	X	X	X
getObject	Short	Integer	Long	Float	Double
getRowId	X	X	X	X	X

Table 3-4 对SUNDB类型的getter method支持与否-1

	BOOLEAN	FLOAT/ NUMBER	CHAR/ VARCHAR/ LONG VARCHAR	BINARY/ VARBINARY/LONG VARBINARY	ROWID
getByte	0 or 1	O	只有数字时可行	X	X

	BOOLEAN	FLOAT/ NUMBER	CHAR/ VARCHAR/ LONG VARCHAR	BINARY/ VARBINARY/LONG VARBINARY	ROWID
getShort	0 or 1	O	只有数字时可行	X	X
getInt	0 or 1	O	只有数字时可行	X	X
getLong	0 or 1	O	只有数字时可行	X	X
getFloat	0 or 1	O	只有数字时可行	X	X
getDouble	0 or 1	O	只有数字时可行	X	X
getBigDecimal	0 or 1	O	只有数字时可行	X	X

	BOOLEAN	FLOAT/ NUMBER	CHAR/ VARCHAR/ LONG VARCHAR	BINARY/ VARBINARY/LONG VARBINARY	ROWID
getBoolean	O	只有01时可行	只有"t", "f", "true", "false", "y", "n", "yes", "no", "on", "off", "1", "0" 时可行（不区分大小写）	X	X
getString	"TRUE" or "FALSE"	O	O	O	O
getBytes	raw数据	raw数据	raw数据	O	raw数据
getDate	X	X	只有date format时可行	X	X
getTime	X	X	只有time format时可行	X	X

	BOOLEAN	FLOAT/ NUMBER	CHAR/ VARCHAR/ LONG VARCHAR	BINARY/ VARBINARY/LONG VARBINARY	ROWID
getTimestamp	X	X	只有timestamp format时可行	X	X
getAsciiStream	raw数据	raw数据	raw数据	O	raw数据
getBinaryStream	raw数据	raw数据	raw数据	O	raw数据
getCharacterStream	X	X	O	X	X
getClob	"TRUE" or "FALSE"	O	O	O	O
getBlob	raw数据	raw数据	raw数据	raw数据	raw数据
getArray	X	X	X	X	X
getRef	X	X	X	X	X
getURL	X	X	X	X	X
getObject	Boolean	BigDecimal	String	byte[]	RowId

	BOOLEAN	FLOAT/ NUMBER	CHAR/ VARCHAR/ LONG VARCHAR	BINARY/ VARBINARY/LONG VARBINARY	ROWID
getRowId	X	X	X	X	O

Table 3-5 对SUNDB类型的getter method支持与否-2

	DATE	TIME/ TIME WITH TIME ZONE	TIMESTAMP/ TIMESTAMP WITH TIME ZONE	INTERVAL	REF CURSOR
getByte	X	X	X	只单个项目类型可行	X
getShort	X	X	X	只单个项目类型可行	X
getInt	X	X	X	只单个项目类型可行	X
getLong	X	X	X	只单个项目类型可行	X
getFloat	X	X	X	只单个项目类型可行	X

	DATE	TIME/ TIME WITH TIME ZONE	TIMESTAMP/ TIMESTAMP WITH TIME ZONE	INTERVAL	REF CURSOR
getDouble	X	X	X	只单个项目类型可行	X
getBigDecimal	X	X	X	只单个项目类型可行	X
getBoolean	X	X	X	X	X
getString	O	O	O	O	X
getBytes	raw数据	raw数据	raw数据	raw数据	X
getDate	O	O	O	X	X
getTime	O	O	O	X	X
getTimestamp	O	O	O	X	X
getAsciiStream	raw数据	raw数据	raw数据	raw数据	X
getBinaryStream	raw数据	raw数据	raw数据	raw数据	X
getCharacterStream	X	X	X	X	X
getClob	O	O	O	O	X
getBlob	raw数据	raw数据	raw数据	raw数据	X

	DATE	TIME/ TIME WITH TIME ZONE	TIMESTAMP/ TIMESTAMP WITH TIME ZONE	INTERVAL	REF CURSOR
getArray	X	X	X	X	X
getRef	X	X	X	X	X
getURL	X	X	X	X	X
getObject	Date	Time	Timestamp	SUNDBInterval	ResultSet
getRowId	X	X	X	X	X

Table 3-6 对SUNDB类型的getter method支持与否-3

4.Embedded SQL

4.1 Precompiler

概要

SUNDB的预编译器（precompiler）是可以使在高级（high-level）编程语言中使用embedded SQL的开发工具目前SUNDB仅支持对C/C++语言的预编译其工具名为gpec

Embedded SQL应用程序开发

如**Embedded SQL Application Development**用户编写包含embedded SQL的C源程序并通过gpec预编译器对其进行转换即生成将源代码上的嵌入式SQL转换为调用SUNDB库的内容的纯C代码该C代码使用系统的C编译器编译为object代码后链接SUNDB提供的embedded SQL库libsundbesql.a创建最终的应用程序

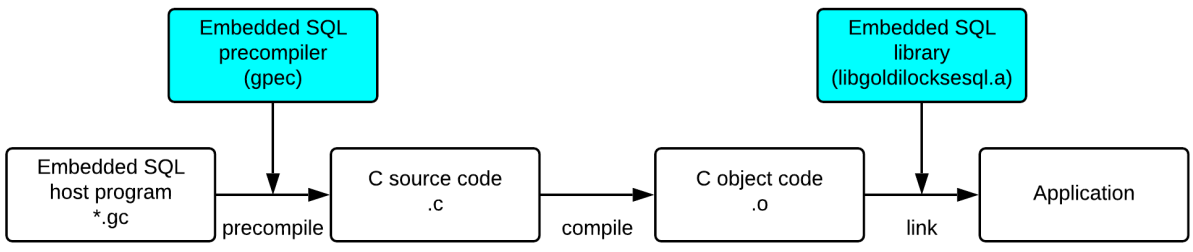


Figure 4-1 Embedded SQL application development

Embedded SQL应用程序开发工具构成

SUNDB的embedded SQL应用程序的构成如下

Directory or file	Description
\bin\gpec	SUNDB Precompiler Embedded SQL for C
\include\sundbesql.h	Embedded SQL library header file. 预编译器自动插入因此用户无需进行其他操作
\include\sqlca.h	SQLCA资料结构相关header file
\lib\libsundbesql.a, \lib\libsundbesqls.so	是Embedded SQL run-time library
\lib\libsundb.a, \lib\libsundbs.so	是SUNDB DA/ CS混用mode library
\lib\libsundba.a, \lib\libsundbas.so	是SUNDB DA mode library
\lib\libsundbc.a, \lib\libsundbcs.so	是SUNDB CS mode library
\sample\EmbeddedSQL	是Sample program

Building Application

本章节说明构建SUNDB的embedded SQL源代码程序后创建其可执行的应用程序的过程

Precompile

说明

预编译用户使用embedded SQL编写的C/C++源代码后生成纯C/C++源代码此过程的核心是将用户编写的嵌入式SQL转换为SUNDB提供的库调用（library call）并且不转换除嵌入式SQL外的C / C ++源代码

用法

SUNDB的预编译器名为gpec位于$SUNDB_HOME/bin/

gpec的用法为如下

```
$ gpec [OPTION]... <input file>
```

gpec输入<input file>并经过预编译过程后生成C/C++源代码<input file>的默认拥有*.gc扩展名其扩展名可省略如果<input file>没有*.gc扩展名则必须指定文件的扩展名

gpec的选项相关详细内容参考**Precompiler Options**

Example

```
$ gpec sample1


FileName: sample1
Pre-compile sample1.gc -> sample1.c
```

Compile

经过预编译过程生成的代码是C/C++源代码此源代码使用platform提供的C/C++编译器生成object code此过程的详细内容参考用户各自的Platform提供的C/C++编译器使用手册

Link

链接通过上述过程生成的Object Code创建ApplicationSUNDB为了嵌入式SQL而提供libsundbesql.a此library包含预编译器转换Embedded SQL的SUNDB API因此创建Embedded SQL应用程序时必须要有

此外SUNDB根据不同的运行模式需要不同的library根据当前应用程序的运行模式选择并链接如下library

工作 mode	Static library	Shared object
DA专用	libsundba.a	libsundbas.so
CS专用	libsundbc.a	libsundbcs.so
DA/CS混合模式	libsundb.a	libsundbs.so

其他的Link过程也与一般C/C++应用程序的生成过程相同可参考platform提供的linker手册

Example

为了方便执行上述precompilecompilelink过程经常使用make以下为简单的创建sample程序的makefile示例用户可以参考此示例创建使用符合各自环境的makefile

```
GPEC = gpec
```

```
GPECFLAGS =

#GPECFLAGS = --unsafe-null --no-prompt


CC = gcc

CFLAGS = -g -Wall

INC = -I$(SUNDB_HOME)/include


LFLAGS = -L$(SUNDB_HOME)/lib

LIB = -lsundbesql -lsundbc -lpthread -lm -lrt


BINS = overview sample1 sample2 sample3 sample4 sample5 dyn1 dyn2 number

date_time thread1 fetch_struct_array


ifneq ($(MAKECMDGOALS), clean)

ifneq ($(MAKECMDGOALS), all)

TARGET = $(MAKECMDGOALS)

OBJECT = $(TARGET).o

C_SRC  = $(TARGET).c

endif

endif
```

- 默认规则

```
.SUFFIXES: .gc .c .o


.gc.c:
```

```
    $(GPEC) $(GPECFLAGS) $^          ❶ Precompile


.c.o:
    $(CC) $(CFLAGS) -c $(INC) $^     ❷ Compile
```

- Build规则

```
NoTarget :
    @echo "Syntax : make {all | sample_name | clean}"
    @echo "sample_name is one of '$(BINS)'"


all :
    for target in $(BINS); do \
        $(MAKE) $$target;     \
    done


$(OBJECT) : $(C_SRC)
$(TARGET) : $(OBJECT)
    $(CC) -o $@ $^ $(LFLAGS) $(LIB)    ❸ Link


clean :
    rm -rf $(BINS) *.o *.c *~ core
```

Sample

SUNDB提供简单的嵌入式SQL样本代码帮助用户了解如何编写嵌入式SQL应用程序样本代码位于

$SUNDB_HOME/sample/EmbeddedSQL目录下要运行其样本需先执行对应目录中的sample.sql

```
$ cd $SUNDB/sample/EmbeddedSQL
$ gsql test test -i sample.sql
$ make
Syntax : make {all | sample_name | clean}
sample_name is one of 'overview sample1 sample2 sample3 sample4 sample5
dyn1 dyn2 number date_time thread1 fetch_struct_array xa long_binary psm
whenever preprocess'
```

make all构建所有样本如果要单独构建特定样本则执行make <sample_name>即可

<sample_name>参考上述信息构建并执行 sample2则会显示如下结果

```
$ make sample2
gpec  sample2.gc


FileName: sample2.gc
Pre-compile sample2.gc -> sample2.c
gcc -g -Wall -c -I/home/mycomman/work/product/Gliese/home/include
sample2.c
gcc -o sample2 sample2.o -L/home/mycomman/work/product/Gliese/home/lib -
lsundbesql -lpthread -lm -lrt -lsundba
$ ./sample2
Connect SUNDB ...
 EMPNO    ENAME                JOB       SALARY
====== ==================== ========== ========
```

```
  2854                Park        RND      800

  2098                 Kim    SALESMAN     1600

  2175                Choi    SALESMAN     1250

  2306                 Lee     SUPPORT     2975

  2122                 Lyu    SALESMAN     1250

  2999                 Ohn     SUPPORT     2850

  2012               Cheon     SUPPORT     2450

  2168                Sohn        RND     3000

  2836                 Seo        CEO     5000

  2022                Song    SALESMAN     1500

  2232               Jeong        RND     1100

  2676                Kang        RND      950

  2714                 Cho        RND     3000

  2441                Yoon        RND     1300

====== ==================== ========== ========

Record Count = 14

====== ==================== ========== ========


SUCCESS

############################
```

Precompiler Options

本章节说明gpec的选项

--no-prompt, -n

说明

不输出版本信息

使用示例

```
$ gpec --no-prompt sample2

FileName: sample2

Pre-compile sample2.gc -> sample2.c

$
```

--version, -v

说明

仅输出版本信息并退出

使用示例

```
$ gpec --version


$
```

--help, -h

说明

输出帮助信息即使gpec未赋予任何选项或<input file>也同样执行

使用示例

```
$ gpec --help


gpec is the SUNDB embedded SQL precompiler for C programs.


Usage:
  gpec [OPTION]... <input file>


Options:
  --no-prompt    No Print version information
  --version      Print version information and exit
  --help         Print help message
  --output       Describe output filename
  --unsafe-null  Allow a NULL fetch without indicator variable
  --include-path Describe header file path
  --no-lineinfo  Exclude line information
  --char_map     Mapping of character arrays ( CHARZ | STRING )
  --cumulative   SQLERRD[2] records cumulative sum of rows processed for
FETCH CURSOR.
```

```
  --autocommit   Set autocommit TRUE to all connection.
$
```

--output, -o

说明

指定预编译结果的文件名不指定此选项时自动生成与<input file>文件名相同并扩展名为.c的文件

使用示例

- 不使用output时

```
$ gpec sample2.gc


FileName: sample2.gc
Pre-compile sample2.gc -> sample2.c
$ ls
sample2.c  sample2.gc
```

- 使用output时

```
$ gpec --output outfile.cpp sample2.gc
FileName: sample2.gc
Pre-compile sample2.gc -> outfile.cpp
$ ls
```

```
outfile.cpp  sample2.gc
```

--unsafe-null

说明

不使用host indicator variable时发生NULL fetch时返回成功其仅表示运算成功无法获取NULL值

使用示例

```
$ gpec --unsafe-null sample2

FileName: sample2

Option : --unsafe-null

Pre-compile sample2.gc -> sample2.c

$
```

--include-path, -I

说明

指定预编译时参考的头文件的路径执行预编译的过程中通过EXEC SQL INCLUDE语句查找其他头文件首先从当前文件的目录开始查找如果没有则根据此选项所属的目录按照顺序进行查找

使用示例

头文件存在于include目录时的示例

- 报错时

```
$ gpec sample.gc


FileName: sample.gc

Pre-compile sample.gc -> sample.c


ERR-42000(41000): syntax error

Error at line 12, in file sample.gc

ERR-42000(41004): "decl.h": file not exist


ERR-42000(41000): syntax error

rsEmpRecord gRecord[] = {

^

Error at line 15, in file sample.gc
```

- 使用 -I 选项时

```
$ gpec -Iinclude sample.gc

FileName: sample.gc

Pre-compile sample.gc -> sample.c

$
```

--no-lineinfo

说明

GPEC默认将gc文件转换为c文件时为了可以debugging为gc文件而添加#liine信息但使用此选项

生成c文件时不再通过#line preprocessor添加line信息

使用示例

```
$ gpec --no-lineinfo sample2

FileName: sample2

Pre-compile sample2.gc -> sample2.c

$
```

--char_map, -c

说明

设置DECLARE SECTION中声明的char类型数据的映射（mapping）类型默认值为“STRING”是以空值终止的数据类型“CHARZ”是空格填充（space padding）并以NULL终止的数据类型

使用示例

```
$ gpec --char_map=STRING overview

FileName: overview

Pre-compile overview.gc -> overview.c


$ gpec --char_map=CHARZ overview

FileName: overview

Pre-compile overview.gc -> overview.c
```

--define, -D

说明

gpec中使用的定义（define）名称设置为1

使用示例

```
$ gpec --define=AAA preprocess

FileName: preprocess

Pre-compile preprocess.gc -> preprocess.c



$ gpec -D BBB preprocess

FileName: preprocess

Pre-compile preprocess.gc -> preprocess.c

ERR-42000(41028): 'BBB' macro is already defined at line 45, in file

preprocess.gc
```

--cumulative

说明

针对FETCH CURSOR语句sqlerrd[2]将按照累积的和进行处理

使用示例

```
$ gpec --cumulative sample.gc
```

```
FileName: sample Option : --cumulative

Pre-compile sample.gc -> sample.c

$
```

--autocommit

说明

将所有连接的auto commit设置为TRUE

使用示例

```
$ gpec --autocommit sample.gc

FileName: sample Option : --autocommit

Pre-compile sample.gc -> sample.c

$
```

4.2 Embedded SQL

Preprocessing

概要

GPEC中在预编译（precompiling）之前执行预处理（preprocessing）

GPEC中支持的预处理（preprocess）指令是#if#ifdef#if

defined#ifndef#else#elif#endif#define#undef 等

可以通过gpec的 --define选项使用预定义（ predefine）如果预定义为gpec的选项则定义为1

例: gpec --define=_DEV_ Test.gc => 与Test.gc文件中的 #define _DEV_ (1) 相同

适用范围

gpec的SQL预编译功能仅在declare section中应用主机变量并用于EXEC SQL语句（Declare section之外的主机变量不可用于EXEC SQL语句）

gpec预处理器（preprocessor）适用于整个source

Include 文件仅适用以 EXEC SQL INCLUDE 声明的 header文件

```
#define _DEV1_
EXEC SQL BEGIN DECLARE SECTION;
#define _DEV2_
char username[10];
```

```
char password[10];
#ifdef _DEV1_
VARCHAR conn_str[20]; ❶
#elif defined _DEV2_
VARCHAR conn_str[30]; ❷
#endif
EXEC SQL END DECLARE SECTION;
```

Note:

❶ 由于_DEV1_的定义与DECLARE SECTION的位置无关因此在gpec中处理为主机变量

❷ 由于定义了_DEV1_因此该位置变为false并在gpec中处理为空白

使用 gpec将上述文件创建为c文件时如下可看到conn_str[20]被转换#ifdef#elif#endif preprocessor句子和conn_str[30]变为空白

```
/* EXEC SQL BEGIN DECLARE SECTION; */

#define _DEV2_
char username[10];
char password[10];

/* VARCHAR conn_str[20]; */
struct { int len; char arr[20]; } conn_str; ❶

                              ❷
```

```
/* EXEC SQL END DECLARE SECTION; */
```

Note:

❶ 由于_DEV1_的定义与DECLARE SECTION的位置无关因此在gpec中处理为主机变量

❷ 由于定义了_DEV1_因此该位置变为false并在gpec中处理为空白

类型

#if

- 语法

```
#if constant
```

或

```
#if defined identifier
```

或

```
#if !defined identifier
```

- 示例

```
#if 0
int sVar1;
```

```
#endif


#if 3-2    ❶ 可运算
int sVar2;
#endif


#if defined _DEV_
int sVar3;
#endif


#if !defined (_DEV_)
int sVar4;
#endif
```

#ifdef, #ifndef

- 语法

```
#ifdef identifier
```

或

```
#if !defined identifier
```

- 示例

```
#ifdef _DEV_
```

```
int sVar1;
#endif

#ifndef _DEV_
int sVar2;
#else
int sVar3;
#endif
```

#else, #elif, #endif

- 语法

```
#else
```

或

```
#endif
```

或

```
#elif constant
```

或

```
#elif defined identifier
```

- 示例

```
#if 1
int sVar1;
#else
int sVar2;
#endif

#if 0
int sVar3;
#elif 1
int sVar4;
#else
int sVar5;
#endif



#ifdef _DEV1_
int sVar6;

#elif defined _DEV2_
int sVar7;

#elif !defined _DEV3_
int sVar8;
```

```
#endif
```

#define, #undef

- 语法

```
#define identifier
```

或

```
#define identifier constant
```

或

```
#undef identifier
```

- 示例

```
#define _DEV1_
EXEC SQL BEGIN DECLARE SECTION;
#define _DEV2_
char username[10];
char password[10];
#ifdef _DEV1_
VARCHAR conn_str[20]; ❶
#elif defined _DEV2_
VARCHAR conn_str[30]; ❷
```

```
#endif
#undef _DEV2_
#ifdef _DEV2_
VARCHAR sDept[10];  ❸
#endif
EXEC SQL END DECLARE SECTION;
```

Note:

❶ 由于定义了_DEV1_因此在gpec中处理为主机变量

❷ _DEV2_的#ifdef _DEV1_为 1因此在gpec中处理为空白

❸ _DEV2_被undef因此在gpec中处理为空白

Caution:

可以在#define中使用注释 但如果#define分别写在多行中则无法正确处理注释

```
#define _DEF1_  1 \ ❶
    + 1
#define _DEF2_   1 \ /* this ❷
is comment */ + 1
#define _DEF3_  1  /* this is comment */ + 1  ❸
```

Note:

> ❶ _DEF1_处理为 1 + 1
>
> ❷ _DEF2_处理为 1
>
> ❸_DEF3_处理为 1 + 1

约束事项

即使定义在声明部分内部也不会扩展至EXEC SQL语句

```
EXEC SQL BEGIN DECLARE SECTION;
#define C_EMP_NO   14
char username[10];
EXEC SQL END DECLARE SECTION;



EXEC SQL
    SELECT USERNAME INTO :username
    FROM EMP

    WHERE EMPNO = C_EMP_NO; ❶ 错误的MACRO使用
```

扩展

MACRO可以在c声明语句的中间或EXEC SQL语句的中间使用

```
#define _DEV_
EXEC SQL BEGIN DECLARE SECTION;
```

```
char
#ifdef _DEV_
sTrue[10];
#else
sFalse[10];
#endif
EXEC SQL END DECLARE SECTION;


EXEC SQL SELECT
#ifdef _DEV_
        "true" INTO :sTrue
#else
        "false" INTO :sFalse
#endif
        FROM DUAL;
```

gpec先行处理preprocessor后可如下生成c code以下为省略将SQL statement转换为c code的示例

```
#define _DEV_
EXEC SQL BEGIN DECLARE SECTION;
char


sTrue[10];
```

```
EXEC SQL END DECLARE SECTION;


EXEC SQL SELECT

        "true" INTO :sTrue




        FROM DUAL;
```

Connection

连接数据库

Embedded SQL应用程序为了访问数据库并执行操作需要连接数据库服务器的过程

以下为SUNDB连接数据库的语句

```
EXEC SQL [ AT <db_name> ] CONNECT <user_name> IDENTIFIED BY <password>

[ AT <db_name> ] [ USING <conn_string> ]


<db_name> := dbname | :hostvar

<user_name> := username | :hostvar

<password> := password | :hostvar

<conn_string> := connection_string | :hostvar
```

以下为连接数据库的最基本的方法

```
EXEC SQL BEGIN DECLARE SECTION;
char username[10];
char password[10];
EXEC SQL END DECLARE SECTION;
strcpy( username, "test" );
strcpy( password, "test" );
...

EXEC SQL CONNECT :username IDENTIFIED BY :password;
```

SUNDB支持直接访问共享内存的DA模式与通过TCP通信连接数据库的CS模式以D/A模式运行时直接访问数据库的相同主机因此不需要额外包含上述服务器信息也可以使用但通过CS模式运行时需要指定DSN(Data Source Name) 访问数据库DSN相关详细内容参考**数据源构成**

使用DSN时需要指定connection_string信息此时为了使用connection_string信息而使用USING语句以下为使用名为"SUNDB"的DSN与USING子句的连接语句示例

```
EXEC SQL BEGIN DECLARE SECTION;
char username[10];
char password[10];
char conn_str[20];
EXEC SQL END DECLARE SECTION;
strcpy( username, "test" );
strcpy( password, "test" );
```

```
strcpy( conn_str, "DSN=SUNDB" );
...

EXEC SQL CONNECT :username IDENTIFIED BY :password USING :conn_str;
```

开发应用程序有时需要单独识别各个连接比如D/A模式的多线程程序连接各个connection或在C/S模式下连接多个connection此时使用AT语句赋予各connection的名称

AT语句可以在CONNECT语句的最前面或USING语句的前面以下为使用AT语句的CONNECT语句的示例

```
EXEC SQL BEGIN DECLARE SECTION;
char username[10];
char password[10];
char conn_str[20];
char conn_name[10];
EXEC SQL END DECLARE SECTION;
strcpy( username, "test" );
strcpy( password, "test" );
strcpy( conn_str, "DSN=SUNDB" );
strcpy( conn_name, "DBCONN1" );
...

EXEC SQL CONNECT :username IDENTIFIED BY :password AT :conn_name
USING :conn_str;
```

断开数据库连接

在应用程序断开数据库的连接与Connect语句相同断开连接也分为断开基本连接的方法与断开指定连接的方法另外也提供断开当前应用程序的所有连接的语句

断开单个连接

断开单个连接有指定的方式和默认方式

指定的方式使用DISCONNECT语句其使用方法如下

```
EXEC SQL [ AT <db_name> ] DISCONNECT;
```

事务通过commit或Rollback结束其周期此时可以通过在事务终止语句后面添加RELEASE选项来默认断开连接

```
EXEC SQL [ AT <db_name> ] { COMMIT/ROLLBACK } [ WORK ] RELEASE;
```

断开所有连接

使用以下语句一次性断开当前应用程序的所有连接

```
EXEC SQL DISCONNECT ALL;
```

Transaction

数据库应用程序以事务为单位构成因此embedded SQL程序也要能够操作事务本章节说明其方法

事务的开始与结束

事务开始于连接数据库后执行的第一个SQL这样开始的事务维持到执行指定的结束命令结束命令分为提交(COMMIT)命令与回滚(ROLLBACK)命令

COMMIT

指定事务的完成命令时会产生以下操作

- 在数据库永久反映当前事务开始后产生的所有数据更新
- 反映的更新事项visible的适用于后续开始的所有事务或sensitive的游标
- 删除当前事务生成的所有保存点（savepoint）
- 解除当前事务获取的所有锁（lock）
- 关闭当前事务中打开的游标（但holdable游标除外）
- 结束事务

如下使用完成命令

```
EXEC SQL COMMIT [ WORK ];
```

ROLLBACK

以下为回滚(rollback)命令的种类

- 回滚所有事务
- 回滚部分事务
- Statement-level rollback

回滚事务时会产生以下操作

- 取消当前事务开始后产生的所有数据变更事项回滚到事务发生前的状态
- 删除当前事务中生成的所有保存点
- 解除当前事务中获取的所有锁
- 关闭当前事务中打开的游标（但holdable游标除外）
- 结束事务

如下使用事务的回滚命令

```
EXEC SQL ROLLBACK [ WORK ];
```

使用保存点可回滚部分事务应用程序开发人员明示指定保存点并回滚到该保存点实现事务的部分回滚部分回滚事务时会产生以下操作

- 取消回滚的保存点之后发生的所有更新事项
- 删除回滚的保存点之后生成的保存点
- 解除回滚的保存点之后后获取的锁

部分回滚事务时使用以下命令

```
EXEC SQL ROLLBACK TO SAVEPOINT <savepoint_name>;
```

Statement-level rollback可单独回滚当前执行中的语句例如如在表插入数据时发生unique violation无法继续执行当前语句时取消当前语句才能执行下一个操作

此时SUNDB在内部自动取消语句因此不需要额外通过语句指定命令

Auto commit

一般情况下SUNDB的embedded SQL连接数据库时事务以non auto-commit模式运行

但是为了应用程序的开发便利性或在特定应用程序逻辑环境中需要调整自动提交模式这种情况下SUNDB的embedded SQL precompiler中可使用以下语句开启或关闭自动提交模式

```
EXEC SQL [ AT <db_name> ] AUTOCOMMIT { ON | OFF };
```

RELEASE option

结束（提交/回滚）事务时可使用RELEASE选项断开当前使用中的连接

此选项仅适用于结束整个事务无法用于回滚部分事务（ROLLBACK TO SAVEPOINT）

Host Variables and Datatypes

Embedded SQL应用程序的目的在于联动数据库服务器后通过操作(manipulation)并查询(query)数据获取所需结果

为此需要将应用程序的数据传递给数据库服务器并从数据库服务器获取数据的手段执行其功能的媒介定义为主机变量

用C语言的变量声明主机变量因此应用程序使用主机变量的方法与使用C变量的方法相同此变量处理为SQL语句的一部分并负责数据库服务器和应用程序之间的value输出入

声明Host Variable

需在embedded SQL directive内如下声明host variable

```
EXEC SQL BEGIN DECLARE SECTION;
```

- 声明host variable

```
EXEC SQL END DECLARE SECTION;
```

上述领域称为declare section可在declare section内声明主机变量其声明方法与C variable的声明方法相同以下为声明部分主机变量的示例

```
EXEC SQL BEGIN DECLARE SECTION;
    int     empno;
    char    ename[20];
    double  salary;
EXEC SQL END DECLARE SECTION;
```

声明函数参数

当函数参数用作host variable时需要提供函数参数的信息 此信息必须在以下embedded SQL directive中声明

```
EXEC SQL BEGIN ARGUMENT SECTION;
```

- 声明Host variable

```
EXEC SQL END ARGUMENT SECTION;
```

上述区域称为argument section在argument section内host variable必须以与函数参数相同的类型和名称进行声明

```
void func( int empno, char ename[20], double salary )
{
EXEC SQL BEGIN ARGUMENT SECTION;
    int     empno;
    char    ename[20];
    double  salary;
EXEC SQL END ARGUMENT SECTION;
...
```

Pseudo type虽然不能作为函数参数直接使用但可以利用declare section和typedef关键词来使用

```
EXEC SQL BEGIN DECLARE SECTION;
typedef VARCHAR domainName[20];
EXEC SQL END DECLARE SECTION;



void func( int * no, domainName * name )
{
    EXEC SQL BEGIN ARGUMENT SECTION;
    int        no[20];
    domainName name[20];
```

```
    EXEC SQL END ARGUMENT SECTION;



    EXEC SQL INSERT INTO EMP VALUES ( :no, :name );
}
```

> Caution:
>
> 如果Host variable被声明为指针型则无法确定排列长度 即使函数参数为指针型也应在argument section中指定host variable的排列大小

Host Variable的C Data type

用作host variable的C data type有C语言提供的native type和SUNDB额外提供的data type下表为SUNDB的embedded SQL提供的data type

C Native Datatype

C native datatype为C语言提供的基本类型其范围或大小完全从属于开发应用程序时所使用的平台

C Datatype	Description
char	single character
char[n]	最大长度为n的字符串
short	small integer(2 byte)

C Datatype	Description
int	integer(4 byte)
long	large integer(4/8 byte)
long long	very large integer(8 byte)
float	single precision floating-point number
double	double precision floating-point number

Table 4-1 C native datatype

- char

char type表示single character

- char[n]

char[n] type表示最大长度为n的字符串数据

参考以下示例

```
EXEC SQL BEGIN DECLARE SECTION;
char strName[20];
EXEC SQL BEGIN DECLARE SECTION;
```

如上声明时strName表示最大长度为20的字符串数据

当strName用于ESQL的output时将变为 space padding

> Note:
>
> char[]的最大长度不能超过2001

- short

表示2 byte整数型数据类型

- int

表示4 byte整数型数据类型

- long

long type表示large integer但其值的实际范围取决于platform在64 bit Unix/Linux platform中long是8 byte型整数但在32 bit Unix/Linux platform中是4 byte整数

- long long

指very large integer表示8 byte整数型

- float

C语言的float type表示4 byte Single-precision floating-point number

- double

C语言的double type表示Double-precision floating-point number

Pseudo Datatype

Pseudo type是为了支持多种形式的SUNDB type并为了开发的便利性SUNDB embedded SQL precompiler提供的type其大部分内容通过C的结构体实现

Pseudo type	Description
VARCHAR[n]	最大长度为n的可变长度字符串
LONG VARCHAR	最大长度为100M（104857600）的可变长度字符串
BINARY[n]	最大长度为n的binary data
VARBINARY[n]	最大长度为n的可变长度binary data
LONG VARBINARY	最大长度为100M（104857600）的可变长度binary data
NUMBER	有效位数为38位的整数
NUMBER(p)	有效位数为p位的整数
NUMBER(p, s)	有效位数为p, scale为s的实数
BOOLEAN	Boolean type
DATE	日期型数据
TIME	时间型数据
TIME WITH TIMEZONE	有Timezone的时间型数据
TIMESTAMP	日期时间型数据
TIMESTAMP WITH TIMEZONE	有Timezone的日期时间型数据

Pseudo type	Description
INTERVAL YEAR	Interval data type
INTERVAL MONTH	
INTERVAL DAY	
INTERVAL HOUR	
INTERVAL MINUTE	
INTERVAL SECOND	
INTERVAL YEAR TO MONTH	
INTERVAL DAY TO HOUR	
INTERVAL DAY TO MINUTE	
INTERVAL DAY TO SECOND	
INTERVAL HOUR TO MINUTE	
INTERVAL HOUR TO SECOND	
INTERVAL MINUTE TO SECOND	

Table 4-2 SUNDB embedded SQL pseudo type

VARCHAR

VARCHAR type为可存储可变长度字符串的数据类型由以下结构体构成

```
struct {
    int  len;
    char arr[n];
}
```

其中n表示VARCHAR类型的最大长度

```
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR varstr[100];
EXEC SQL END DECLARE SECTION;
```

如上声明时在预编译过程中转换为如下

```
struct VARCHAR_varstr {
    int  len;
    char arr[100];
} varstr;
```

应用程序中可如下使用VARCHAR类型

```
strcpy( varstr.arr, "abcde" );
varstr.len = strlen( varstr.arr );
```

```
EXEC SQL INSERT INTO TEST_T1 VALUES ( :varstr );
```

> Note:
>
> VARCHAR的长度不能超过4000

LONG VARCHAR

LONG VARCHAR type为可存储超长可变长度字符串的数据类型由以下结构体构成

```
typedef struct SQL_LONG_VARIABLE_LENGTH_STRUCT
{
    SQLBIGINT   len;
    SQLCHAR   * arr;
} SQL_LONG_VARIABLE_LENGTH_STRUCT;
```

可如下声明LONG VARCHAR type

```
EXEC SQL BEGIN DECLARE SECTION;
    LONGVARCHAR long_text;
EXEC SQL END DECLARE SECTION;
```

LONG VARCHAR和VARCHAR的区别是LONG VARCHAR长度最长可达100M(10485760)因此声明时不提前分配存储字符串的空间即应用程序声明LONG VARCHAR后实际使用前需要向long_text.arr分配实际内存空间并使用后释放其空间以下为使用LONG VARCHAR的示例

```
long_text.arr = malloc( 1048576 );
```

```
gets( long_text.arr );
long_text.len = strlen( long_text.arr );

EXEC SQL INSERT INTO TEST_T1 VALUES ( :long_text );
...
free( long_text.arr );
```

> Note:
>
> LONG VARCHAR type的长度当前最多可指定声明至100M(104857600)

BINARY

BINARY type为用于操作不规则的raw data的数据类型如下构成以下为声明BINARY type的使用示例

```
EXEC SQL BEGIN DECLARE SECTION;
    BINARY binary[100];
EXEC SQL END DECLARE SECTION;
```

如上声明时在预编译过程中转换为如下

```
char binary[100];
```

从形式上与存储字符串相同如果定义为char[n]是为了存储数据库的字符串BINARY type内容的区别在于即为二进制数据

因此应用程序中可以与处理字符串数据相同的方法处理BINARY type数据

> Note:
>
> BINARY最大长度不能超过2000

VARBINARY

VARBINARY type为可存储可变长度二进制数据的数据类型由以下结构体构成

```
struct {
    int  len;
    char arr[n];
}
```

其中n表示VARCHAR type的最大长度

```
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR varbin[100];
EXEC SQL END DECLARE SECTION;
```

如上声明时在预编译过程中转换为如下

```
struct VARBINARY_varbin {
    int  len;
    char arr[100];
} varbin;
```

VARBINARY type与VARCHAR type形式相同VARCHAR用于存储字符串而VARBINARY存储二进制数据其余使用方法均相同因此应用程序中可如下与使用VARCHAR相同的方式使用VARBINARY type

```
memcpy( varbin.arr, binary_data, 50 );
varbin.len = 50;


EXEC SQL INSERT INTO TEST_T1 VALUES ( :varbin );
```

> Note:
>
> VARBINARY的长度不能超过4000

LONG VARBINARY

LONG VARBINARY type为可存储超长可变长度二进制数据的数据类型由以下结构体构成

```
typedef struct SQL_LONG_VARIABLE_LENGTH_STRUCT
{
    SQLBIGINT   len;
    SQLCHAR   * arr;
} SQL_LONG_VARIABLE_LENGTH_STRUCT;
```

可如下声明LONG VARBINARY type

```
EXEC SQL BEGIN DECLARE SECTION;
    LONGVARBINARY long_bin;
```

```
EXEC SQL END DECLARE SECTION;
```

LONG VARBINARY和VARBINARY的区别是声明LONG VARBINARY type时不提前分配存储binary data的空间(与LONG VARCHAR和VARCHAR的区别相同)即应用程序在声明LONG VARBINARY后实际使用之前应在long_bin.arr中分配实际内存空间并在使用后释放此内存空间以下为使用LONG VARBINARY的示例

```
long_bin.arr = malloc( 1048576 );


memcpy( long_bin.arr, long_binary_data, 1048576);
long_bin.len = 1048576;


EXEC SQL INSERT INTO TEST_T1 VALUES ( :long_bin );
...
free( long_bin.arr );
```

> Note:
>
> LONG VARBINARY type的最大长度当前可指定声明至100M(104857600)

NUMBER

NUMBER type是为了使应用程序能够使用ODBC定义的SQL_NUMERIC_STRUCT而提供的数据类型

```
#define SQL_MAX_NUMERIC_LEN 16
typedef struct tagSQL_NUMERIC_STRUCT
```

```
{
    SQLCHAR precision;
    SQLSCHAR scale;
    SQLCHAR sign; /* 1=pos 0=neg */
    SQLCHAR val[SQL_MAX_NUMERIC_LEN];
} SQL_NUMERIC_STRUCT;
```

NUMBER type可表示有精度（ Precision）和范围（Scale）的实数型数据声明NUMBER type时可指定精度和范围后进行声明根据情况也可省略范围与精度其意义如下

声明NUMBER type	Description
NUMBER	与NUMBER(38, 0)相同
NUMBER(p)	与NUMBER(p, 0)相同
NUMBER(p, s)	Precision为p, Scale为s的实数型数据

Table 4-3 NUMBER type的 precision和 scale

```
EXEC SQL BEGIN DECLARE SECTION;
    NUMBER       number_default;
    NUMBER(20)    number_20;
    NUMBER(30,10) number_30_10;
EXEC SQL END DECLARE SECTION;
```

例如如上声明变量时number_default变量相当于声明为NUMBER(38, 0)number_20变量相当于声明为NUMBER(20, 0)NUMBER type使用ODBC类型的SQL_NUMERIC_STRUCT以下为使用NUMBER

type的示例

```
/*
 * number.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1

#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                            \
        printf("\n");                                            \
        printf(aMsg);                                            \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                    \
               SQLSTATE,                                         \
               sqlca.sqlerrm.sqlerrmc );                          \
    }
```

```
int Connect(char *aHostInfo, char *aUserID, char *sPassword);

int CreateTable();

int DropTable();


unsigned long long ConvertMantisaToDecimal(SQLCHAR *aNumStrValue)

{

    unsigned long long sResult = 0;

    unsigned long long sLast=1;

    unsigned int sCurrent;

    unsigned int sLSD = 0;

    unsigned int sMSD = 0;

    int          i    = 1;


    for(i = 0; i < SQL_MAX_NUMERIC_LEN; i ++)

    {

        sCurrent = (unsigned char) aNumStrValue[i];

        sLSD = sCurrent % 16; //Obtain LSD

        sMSD = sCurrent / 16; //Obtain MSD

        sResult += sLast * sLSD;

        sLast = sLast * 16;

        sResult += sLast * sMSD;

        sLast = sLast * 16;

    }


    return sResult;
```

```
}


void PrintNumber(SQL_NUMERIC_STRUCT *aNumber)
{
    unsigned long long  sDigit;
    unsigned long long  sFraction;
    unsigned long long  sMantisa;
    unsigned long long  sFactor;
    int     i;

    sMantisa = ConvertMantisaToDecimal( aNumber->val );

    sFactor = 1;
    for( i = 0; i < aNumber->scale; i ++ )
    {
        sFactor *= 10;
    }

    sDigit = sMantisa / sFactor;
    sFraction = sMantisa % sFactor;
    if( sFraction != 0 )
    {
        printf("%llu.%-3llu", sDigit, sFraction);
    }
    else
```

```
    {
        printf("%llu", sDigit);
    }
}


int main(int    argc,
         char  **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    NUMBER        sNumber;
    NUMBER(10,5)  sResultNumber1;
    NUMBER(10,5)  sResultNumber2;
    char          sCharNumber[20];
    int           sNo;
    int           sResultNo;
    EXEC SQL END DECLARE SECTION;
    int   i;
    int   sState = 0;


    printf("#### Number Datatype Test ####\n");
    printf("Connect SUNDB ...\n");
    if(Connect("DSN=SUNDB", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
```

```
printf("Create table ...\n");
if(CreateTable() != SUCCESS)
{
    goto fail_exit;
}
sState = 1;

printf("Insert record ...\n");
for(i = 0; i < 20; i ++)
{
    sNo = i + 1;
    memset( &sNumber, 0x00, sizeof(SQL_NUMERIC_STRUCT) );
    sNumber.precision = 38;
    sNumber.scale = 3;
    sNumber.sign = 1;
    /*
     * 0x627d = 25213
     */
    sNumber.val[0] = 0x7d + i;
    sNumber.val[1] = 0x62;

    snprintf( sCharNumber, 20, "25.2%02d", 13 + i );
    EXEC SQL
        INSERT INTO TEST_T1(C1, C2, C3)
```

```
        VALUES(:sNo, :sNumber, :sCharNumber);
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
}

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

printf("Retrive record\n");
EXEC SQL
    DECLARE CUR1 CURSOR FOR
    SELECT C1, C2, C3
    FROM   TEST_T1;

EXEC SQL OPEN CUR1;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
```

```
printf(" NO   Number1  Number2\n");
printf("==== ======== ========\n");

memset( &sResultNumber1, 0x00, sizeof(SQL_NUMERIC_STRUCT) );
memset( &sResultNumber2, 0x00, sizeof(SQL_NUMERIC_STRUCT) );
while( 1 )
{
    EXEC SQL
        FETCH FROM CUR1
        INTO :sResultNo, :sResultNumber1, :sResultNumber2;

    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        /*
         * No more data
         */
        break;
    }

    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    printf("%3d   ", sResultNo);
```

```
    PrintNumber( &sResultNumber1 );
    printf("   ");
    PrintNumber( &sResultNumber2 );
    printf("\n");
}

printf("==== ======== ========\n");

EXEC SQL CLOSE CUR1;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sState = 0;
printf("Drop table ...\n");
if(DropTable() != SUCCESS)
{
    goto fail_exit;
```

```
    }


    printf("Disconnect SUNDB ...\n");
    EXEC SQL COMMIT WORK RELEASE;


    printf("SUCCESS\n");
    printf("############################\n");


    return 0;


fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    printf("FAILURE\n");
    printf("############################\n\n");


    switch(sState)
    {
        case 1:
            printf("Drop table ...\n");
            (void)DropTable();
        default:
            break;
    }


    EXEC SQL ROLLBACK WORK RELEASE;
```

```
    return 0;
}

int CreateTable()
{
    EXEC SQL DROP TABLE IF EXISTS TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

- Create table

```
EXEC SQL CREATE TABLE TEST_T1 ( C1  INTEGER,
                                C2  NUMERIC(38,4),
                                C3  VARCHAR(20) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
```

```
    }


    return SUCCESS;


  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");


    EXEC SQL ROLLBACK WORK;


    return FAILURE;
}
```

- Drop table

```
int DropTable()
{
    EXEC SQL DROP TABLE TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }


    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
```

```
    }


    return SUCCESS;


  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;


    return FAILURE;
}


int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on SUNDB

```
strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
```

```
sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB连接

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

BOOLEAN

BOOLEAN type拥有TRUE或FALSE值该变量实际用作C语言的变量因此变量值为1时为TRUE0时为FALSE

以下为使用BOOLEAN type变量的示例

```
EXEC SQL BEGIN DECLARE SECTION;
    BOOLEAN boolean;
```

```
EXEC SQL END DECLARE SECTION;


EXEC SQL SELECT IsEnable INTO :boolean FROM STATUS WHERE ID = 100;


if( boolean != 0 )
{
    print( "Enable Status : TRUE\n" );
}
else
{
    print( "Enable Status : FALSE\n" );
}
```

DATE

Date type可管理日期和时间Date type拥有ODBC的SQL_TIMESTAMP_STRUCT结构

```
typedef struct tagTIMESTAMP_STRUCT
{
        SQLSMALLINT    year;
        SQLUSMALLINT   month;
        SQLUSMALLINT   day;
        SQLUSMALLINT   hour;
        SQLUSMALLINT   minute;
        SQLUSMALLINT   second;
        SQLUINTEGER    fraction;
```

```
} TIMESTAMP_STRUCT;

typedef TIMESTAMP_STRUCT SQL_TIMESTAMP_STRUCT;
```

声明DATE类型的变量时预编译器转换为如上结构体每个field的意义与ODBC的SQL_TIMESTAMP_STRUCT相同

TIME

TIME type为管理时间的数据类型拥有ODBC的SQL_TIME_STRUCT结构

```
typedef struct tagTIME_STRUCT
{
        SQLUSMALLINT   hour;
        SQLUSMALLINT   minute;
        SQLUSMALLINT   second;
} TIME_STRUCT;
typedef TIME_STRUCT SQL_TIME_STRUCT;
```

声明TIME类型变量时预编译器转换为如上结构体每个field的意义与ODBC的SQL_TIME_STRUCT相同

TIME WITH TIMEZONE

TIME WITH TIMEZONE type是管理有timezone的时间的数据类型拥有如下结构

```
typedef struct tagTIME_WITH_TIMEZONE_STRUCT
{
   SQLUSMALLINT hour;
```

```
   SQLUSMALLINT minute;
   SQLUSMALLINT second;
   SQLUINTEGER  fraction;
   SQLSMALLINT  timezone_hour;
   SQLSMALLINT  timezone_minute;
} TIME_WITH_TIMEZONE_STRUCT;
typedef TIME_WITH_TIMEZONE_STRUCT SQL_TIME_WITH_TIMEZONE_STRUCT;
```

声明TIME WITH TIMEZONE类型变量时预编译器转换为如上结构体每个field的意义如下

Field name	Description
hour	时间
minute	分
second	秒
fraction	小数点后面的秒
timezone_hour	Timezone的时间
timezone_minute	Timezone的分

Table 4-4 TIME WITH TIMEZONE的 field

TIMESTAMP

TIMESTAMP type是管理日期~时间的数据类型拥有ODBC的SQL_TIMESTAMP_STRUCT结构

```
typedef struct tagTIMESTAMP_STRUCT
```

```
{
        SQLSMALLINT    year;
        SQLUSMALLINT   month;
        SQLUSMALLINT   day;
        SQLUSMALLINT   hour;
        SQLUSMALLINT   minute;
        SQLUSMALLINT   second;
        SQLUINTEGER    fraction;
} TIMESTAMP_STRUCT;
typedef TIMESTAMP_STRUCT SQL_TIMESTAMP_STRUCT;
```

声明TIMESTAMP类型的变量时预编译器转换为如上结构体每个field的意义与ODBC的SQL_TIMESTAMP_STRUCT相同

TIMESTAMP WITH TIMEZONE

TIMESTAMP WITH TIMEZONE type是管理有timezone的timestamp的数据类型拥有如下结构

```
typedef struct tagTIMESTAMP_WITH_TIMEZONE_STRUCT
{
   SQLSMALLINT  year;
   SQLUSMALLINT month;
   SQLUSMALLINT day;
   SQLUSMALLINT hour;
   SQLUSMALLINT minute;
   SQLUSMALLINT second;
```

```
   SQLUINTEGER  fraction;
   SQLSMALLINT  timezone_hour;
   SQLSMALLINT  timezone_minute;
} TIMESTAMP_WITH_TIMEZONE_STRUCT;
typedef TIMESTAMP_WITH_TIMEZONE_STRUCT SQL_TIMESTAMP_WITH_TIMEZONE_STRUCT;
```

声明TIMESTAMP WITH TIMEZONE类型变量时预编译器转换为如上结构体每个field的意义如下

Field name	Description
year	年
month	月
day	日
hour	时
minute	分
second	秒
fraction	小数点后面秒
timezone_hour	Timezone的时间
timezone_minute	Timezone的分

Table 4-5 TIMESTAMP WITH TIMEZONE的field

以下为使用DATETIMETIMESTAMPTIME WITH TIMEZONETIMESTAMP WITH TIMEZONE类型的简单示例

```
/*
 * date_time.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1

#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                            \
        printf("\n");                                            \
        printf(aMsg);                                            \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                    \
               SQLSTATE,                                         \
               sqlca.sqlerrm.sqlerrmc );                          \
    }

int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int CreateTable();
```

```
int DropTable();


int main(int    argc,
        char  **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int                    sNo;
    int                    sResultNo;
    DATE                   sDate, sResultDate;
    TIME                   sTime, sResultTime;
    TIME WITH TIMEZONE      sTimeTz, sResultTimeTz;
    TIMESTAMP              sTimestamp, sResultTimestamp;
    TIMESTAMP WITH TIMEZONE  sTimestampTz, sResultTimestampTz;
    EXEC SQL END DECLARE SECTION;
    int  sState = 0;


    printf("#### Datatype Insert Test ####\n");
    printf("Connect SUNDB ...\n");
    if(Connect("DSN=SUNDB", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }


    sState = 1;
    printf("Create table ...\n");
```

```
if(CreateTable() != SUCCESS)
{
    goto fail_exit;
}

sState = 2;
printf("Insert record ...\n");

sNo = 1;
/**
 * Date : 2014-7-14 21:29:30
 */
sDate.year  = 2014;
sDate.month = 7;
sDate.day   = 14;
sDate.hour  = 21;
sDate.minute = 29;
sDate.second = 30;
sDate.fraction = 0;

/**
 * Time : 17:46:35
 */
sTime.hour   = 17;
sTime.minute = 46;
```

```
sTime.second = 35;


/**
 * Time With Timezone : 17:46:35.6789(+9:00)
 */
sTimeTz.hour            = 17;
sTimeTz.minute          = 46;
sTimeTz.second          = 35;
sTimeTz.fraction        = 678900000;
sTimeTz.timezone_hour   = 9;
sTimeTz.timezone_minute  = 0;


/**
 * Timestamp : 2014-02-13 17:46:28.123
 */
sTimestamp.year     = 2014;
sTimestamp.month    = 2;
sTimestamp.day      = 13;
sTimestamp.hour     = 17;
sTimestamp.minute   = 46;
sTimestamp.second   = 28;
sTimestamp.fraction = 123000000;


/**
 * Time With Timezone : 2014-05-18 17:46:35.001(+9:00)
```

```
     */
    sTimestampTz.year     = 2014;
    sTimestampTz.month    = 5;
    sTimestampTz.day      = 18;
    sTimestampTz.hour     = 17;
    sTimestampTz.minute   = 46;
    sTimestampTz.second   = 35;
    sTimestampTz.fraction = 1000000;
    sTimestampTz.timezone_hour    = 9;
    sTimestampTz.timezone_minute  = 0;

    /**
     * Insert record
     */
    EXEC SQL
        INSERT INTO TEST_T1(C1, C2, C3, C4, C5, C6)

VALUES(:sNo, :sDate, :sTime, :sTimeTz, :sTimestamp, :sTimestampTz);
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
```

```
    {
        goto fail_exit;
    }

    printf("Retrive record\n");
    EXEC SQL
        SELECT C1, C2, C3, C4, C5, C6

INTO   :sResultNo, :sResultDate, :sResultTime, :sResultTimeTz, :sResultTim
estamp, :sResultTimestampTz
        FROM   TEST_T1
        WHERE  C1 = 1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }


printf("================================================================\
n");
    printf( "DATE                 : %04d-%02d-%02d %02d:%02d:%02d\n",
            sResultDate.year,
            sResultDate.month,
            sResultDate.day,
            sResultDate.hour,
```

```
            sResultDate.minute,
            sResultDate.second );

    printf( "TIME                  : %02d:%02d:%02d\n",
            sResultTime.hour,
            sResultTime.minute,
            sResultTime.second );

    printf( "TIME WITH
TIMEZONE     : %02d:%02d:%02d.%09u(GMT %+02d:%02d)\n",
            sResultTimeTz.hour,
            sResultTimeTz.minute,
            sResultTimeTz.second,
            sResultTimeTz.fraction,
            sResultTimeTz.timezone_hour,
            sResultTimeTz.timezone_minute );


printf( "TIMESTAMP             : %04d-%02d-%02d %02d:%02d:%02d.%09u\n",
            sResultTimestamp.year,
            sResultTimestamp.month,
            sResultTimestamp.day,
            sResultTimestamp.hour,
            sResultTimestamp.minute,
            sResultTimestamp.second,
```

```
            sResultTimestamp.fraction );


    printf( "TIMESTAMP WITH
TIMEZONE: %04d-%02d-%02d %02d:%02d:%02d.%09u(GMT %+02d:%02d)\n",
            sResultTimestampTz.year,
            sResultTimestampTz.month,
            sResultTimestampTz.day,
            sResultTimestampTz.hour,
            sResultTimestampTz.minute,
            sResultTimestampTz.second,
            sResultTimestampTz.fraction,
            sResultTimestampTz.timezone_hour,
            sResultTimestampTz.timezone_minute );



printf("=================================================================\
n");


    sState = 0;
    printf("Drop table ...\n");
    if(DropTable() != SUCCESS)
    {
        goto fail_exit;
    }
```

```
    sState = 0;
    printf("Disconnect SUNDB ...\n");
    EXEC SQL COMMIT WORK RELEASE;

    printf("SUCCESS\n");
    printf("##############################\n");

    return 0;

fail_exit:
    printf("\n");
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    printf("FAILURE\n");
    printf("##############################\n\n");

    switch(sState)
    {
        case 1:
            printf("Drop table ...\n");
            (void)DropTable();
        default:
            break;
    }
```

```
    EXEC SQL ROLLBACK WORK RELEASE;


    return 0;
}


int CreateTable()
{
    EXEC SQL DROP TABLE IF EXISTS TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

- Create table

```
EXEC SQL CREATE TABLE TEST_T1 ( C1  INTEGER,
                                C2  DATE,
                                C3  TIME,
                                C4  TIME WITH TIME ZONE,
                                C5  TIMESTAMP,
                                C6  TIMESTAMP WITH TIME ZONE );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

```
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropTable()
{
    EXEC SQL DROP TABLE TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

```
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on SUNDB

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB连接

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

INTERVAL Types

INTERAVAL type为表示两个时间之间的间隔的数据类型大致分为year~month系列与

day~second系列每个系列再细分为类型

INTERVAL type的详细区分如下

系列	详细类型	Description
YEAR TO MONTH	INTERVAL YEAR	年份之差
	INTERVAL MONTH	月份之差
	INTERVAL YEAR TO MONTH	年份到月份之差
DAY TO SECOND	INTERVAL DAY	日期之差
	INTERVAL HOUR	时间之差
	INTERVAL MINUTE	分之差
	INTERVAL SECOND	秒之差
	INTERVAL DAY TO HOUR	日期到时间之差
	INTERVAL DAY TO MINUTE	日期到分之差
	INTERVAL DAY TO SECOND	日期到秒之差
	INTERVAL HOUR TO MINUTE	时间到分之差
	INTERVAL HOUR TO SECOND	时间到秒之差
	INTERVAL MINUTE TO SECOND	分到秒之差

Table 4-6 INTERVAL type区分

INTERVAL type拥有ODBC的SQL_INTERVAL_STRUCT结构其内容如下

```
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
{
    SQLUINTEGER  year;
    SQLUINTEGER  month;
} SQL_YEAR_MONTH_STRUCT;
```

```
typedef struct tagSQL_DAY_SECOND
{
    SQLUINTEGER  day;
    SQLUINTEGER  hour;
    SQLUINTEGER  minute;
    SQLUINTEGER  second;
    SQLUINTEGER  fraction;
} SQL_DAY_SECOND_STRUCT;

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

上述所有INTERVAL type使用相同的结构体因此每个INTERVAL type在结构体内再单独区分有效field并用结构体内的interval_type字段区分INTERVAL type通过结构体内的intval共用体表示实际INTERVAL value按照各类型在共用体中使用intval的字段不同各类型的有效feild如下所示

Type	?.interval_type	?.intval有效field
INTERVAL YEAR	SQL_IS_YEAR	*.year_month.year
INTERVAL MONTH	SQL_IS_MONTH	*.year_month.month
INTERVAL YEAR TO MONTH	SQL_IS_YEAR_TO_MONTH	*.year_month.year *.year_month.month
INTERVAL DAY	SQL_IS_DAY	*.day_second.day
INTERVAL HOUR	SQL_IS_HOUR	*.day_second.hour
INTERVAL MINUTE	SQL_IS_MINUTE	*.day_second.minute
INTERVAL SECOND	SQL_IS_SECOND	*.day_second.second *.day_second.fraction
INTERVAL DAY TO HOUR	SQL_IS_DAY_TO_HOUR	*.day_second.day *.day_second.hour
INTERVAL DAY TO MINUTE	SQL_IS_DAY_TO_MINUTE	*.day_second.day *.day_second.hour *.day_second.minute
INTERVAL DAY TO SECOND	SQL_IS_DAY_TO_SECOND	*.day_second.day *.day_second.hour *.day_second.minute *.day_second.second *.day_second.fraction

Type	?.interval_type	?.intval有效field
INTERVAL HOUR TO MINUTE	SQL_IS_HOUR_TO_MINUTE	*.day_second.hour *.day_second.minute
INTERVAL HOUR TO SECOND	SQL_IS_HOUR_TO_SECOND	*.day_second.hour *.day_second.minute *.day_second.second *.day_second.fraction
INTERVAL MINUTE TO SECOND	SQL_IS_MINUTE_TO_SECOND	*.day_second.minute *.day_second.second *.day_second.fraction

Table 4-7 各INTERVAL type的有效field

Yearmonthdayhourminutesecond各表示年月日时分秒fraction表示小数点后面的秒(Fraction field仅在包含SECOND的INTERVAL type中有效)

Special Type

Special type是为编写应用程序提供额外的功能和便利的特殊形式的数据类型而非仅处理数据

Special type包括以下类型

Type	Description
SQL_CONTEXT	在multi-connection结构中管理run-time context
Struct	用结构体构成Column的集合处理row时使用

Type	Description
Typedef	用其他名称重新定义已有的type

Table 4-8 Special type

SQL_CONTEXT

SQL_CONTEXT是管理run-time context的特殊形式的数据类型应用程序执行过程中通过run-time管理connection和connection相关的个别数据时使用run-time context

如下声明SQL_CONTEXT变量

```
EXEC SQL BEGIN DECLARE SECTION;
SQL_CONTEXT my_context;
EXEC SQL BEGIN DECLARE SECTION;
```

声明SQL_CONTEXT变量后应如下分配

```
EXEC SQL CONTEXT ALLOCATE :my_context;
```

使用SQL_CONTEXT变量时使用USE语句

```
EXEC SQL CONTEXT USE :my_context;
```

USE语句指定要使用的context但在应用程序中使用默认context时如下使用

```
EXEC SQL CONTEXT USE DEFAULT;
```

如下释放不再使用的SQL_CONTEXT变量

```
EXEC SQL CONTEXT FREE :my_context;
```

以下为使用SQL_CONTEXT的示例

```
/*
 * thread1.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1

#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                            \
        printf("\n");                                            \
        printf(aMsg);                                            \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                    \
```

```
                SQLSTATE,                                          \
                sqlca.sqlerrm.sqlerrmc );                           \
    }

int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char
*sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();
void *clientThread(void *args);

typedef struct thread_param
{
    int    mNo;
    char  *mJobName;
} thread_param;

#define  THREAD_COUNT    2

char gJobName[THREAD_COUNT][20]= {
    "RND",
    "SUPPORT"
};

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

```
    EXEC SQL BEGIN DECLARE SECTION;
    int         sEmpNo;
    varchar      sEName[20 + 1];
    char        sJob[20];
    long        sSalary;
    EXEC SQL END DECLARE SECTION;
    int         sRecordCount = 0;
    pthread_t    thread_id[THREAD_COUNT];
    thread_param param[THREAD_COUNT];
    int         i;

    printf("Connect SUNDB ...\n");
    if(Connect(NULL, "DSN=SUNDB", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }

    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }
```

- Create client thread

```
for( i = 0; i < THREAD_COUNT; i ++ )
    {
```

```
        param[i].mNo = i;
        param[i].mJobName = gJobName[i];
        if( pthread_create(&thread_id[i],
                           NULL,
                           clientThread,
                           &param[i]) != 0 )
        {
            printf( "Can't create thread %d!\n", i );
        }
        else
        {
            printf( "Create thread %d!\n", i );
        }
    }

    for( i = 0; i < THREAD_COUNT; i ++ )
    {
        if( pthread_join(thread_id[i],
                         NULL) != 0 )
        {
            printf( "Error when waiting for thread %d to terminate!\n",
i );
        }
        else
        {
```

```
            printf( "Stopped thread %d!\n", i );
        }
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP_TEMP
        ORDER BY empno;


    EXEC SQL OPEN EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }


    printf(" EMPNO    ENAME               JOB      SALARY\n");
    printf("====== ==================== ========== ========\n");
    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
```

```
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sRecordCount ++;

    printf("%6d %20s %10s %8ld\n",
           sEmpNo, sEName.arr, sJob, sSalary);
}

printf("====== ==================== ========== ========\n");
printf("Record Count = %d\n", sRecordCount);
printf("====== ==================== ========== ========\n");

EXEC SQL CLOSE EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
```

```
    }

    if(DropEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("SUCCESS\n");
    printf("##############################\n");

    return 0;

fail_exit:

    printf("FAILURE\n");
    printf("##############################\n\n");
    EXEC SQL ROLLBACK WORK RELEASE;
```

```
    return 0;
}


int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char
*sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
    struct sqlca sqlca;
```

- Log on SUNDB

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB连接

```
if( aCtx != NULL )
    {
```

```
        EXEC SQL CONTEXT USE :aCtx;
        EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    }
    else
    {
        EXEC SQL CONTEXT USE DEFAULT;
        EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    }

    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}

int Disconnect(sql_context aCtx)
{
```

```
    struct sqlca sqlca;
```

- 解除DB连接

```
if( aCtx != NULL )
    {
        EXEC SQL CONTEXT USE :aCtx;
        EXEC SQL DISCONNECT;
    }
    else
    {
        EXEC SQL CONTEXT USE DEFAULT;
        EXEC SQL DISCONNECT;
    }
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
```

```
}
```

- Create table

```
int CreateEmpTempTable()
{
   EXEC SQL DROP TABLE IF EXISTS EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        CREATE TABLE EMP_TEMP (
            EMPNO NUMBER(4) CONSTRAINT PK_EMP_TEMP PRIMARY KEY,
            ENAME VARCHAR2(10),
            JOB VARCHAR2(9),
            SAL NUMBER(7,2),
            DEPTNO NUMBER(2) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
```

```
    {
        goto fail_exit;
    }
    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
```

```
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

void *clientThread(void *args)
{
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT   my_context;
    char          job_name[20 + 1];
    EXEC SQL END DECLARE SECTION;
    int           state = 0;
    thread_param *param = (thread_param *)args;

    EXEC SQL CONTEXT ALLOCATE :my_context;
    state = 1;
```

```
EXEC SQL CONTEXT USE :my_context;

if(Connect(my_context, "DSN=SUNDB", "test", "test") != SUCCESS)

{

    goto fail_exit;

}

state = 2;


strcpy( job_name, param->mJobName );


EXEC SQL

    INSERT INTO EMP_TEMP

    SELECT *

    FROM   EMP

    WHERE  JOB = :job_name;

if(sqlca.sqlcode != 0)

{

    goto fail_exit;

}


EXEC SQL COMMIT WORK;

if(sqlca.sqlcode != 0)

{

    goto fail_exit;

}

state = 1;
```

```
  if(Disconnect(my_context) != SUCCESS)
  {
      goto fail_exit;
  }
  state = 0;

  EXEC SQL CONTEXT FREE :my_context;
  pthread_exit(0);

  return NULL;

fail_exit:

  PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
  switch(state)
  {
      case 2:
          (void)Disconnect(my_context);
      case 1:
          EXEC SQL CONTEXT FREE :my_context;
          break;
      default:
          break;
  }
```

```
    pthread_exit(0);

    return NULL;
}
```

SQL CURSOR

可在embedded SQL程序中使用cursor变量用于查询Cursor变量是需要使用PL/SQL在sundb服务器上进行定义和打开的游标的句柄

使用SQL_CURSOR类型在embedded SQL声明cursor变量下面是生命cursor变量的例子

```
EXEC SQL BEGIN DECLARE SECTION;
sql_cursor   emp_cursor;
SQL_CURSOR   sal_cursor;
sql_cursor * cur_ptr;
EXEC SQL END DECLARE SECTION;
cur_ptr = &emp_cursor;
```

Cursor变量可声明为sql_cursor或SQL_CURSOR进行使用Cursor变量在embedded SQL中和其他host变量一样都遵循C的范围规则可以作为函数的参数进行传递也可定义cursor变量或对cursor变量的指针的返回函数以下是cursor变量作为函数的参数和返回的例子

```
SQL_CURSOR * alloc_cursor( sql_cursor * emp_cursor )
{
EXEC SQL BEGIN DECLARE SECTION;
```

```
    sql_cursor * ret_cursor;
EXEC SQL END DECLARE SECTION;
    ret_cursor = emp_cursor;


....


    return ret_cursor;
}
```

Cursor变量需要在使用前进行分配在声明cursor变量后可使用**ALLOCATE**命令执行下面是对上述例子的alloc_cursor函数中使用的ret_cursor进行分配的例子

```
EXEC SQL ALLOCATE :ret_cursor;
```

Cursor变量的分配是在run-time过程中分配内存所以需要使用**FREE**命令进行释放Cursor变量分配时生成的内存在cursor关闭时不会进行释放而是在明确使用CLOSE命令进行或连接断开时释放

```
EXEC SQL FREE :ret_cursor;
```

需要在Sundb数据库服务器上开启cursor变量不能使用embedded SQL的OPEN命令来打开cursor变量需要调用开启cursor的PL/SQL存储过程或函数来开启cursor变量或定义可以开启cursor的anonymous block来开启cursor变量

以下是调用PL/SQL的存储过程开启cursor变量的例子

```
CREATE OR REPLACE PROCEDURE empProc( curs IN OUT SYS_REFCURSOR,
```

```
                                    dept IN INTEGER )
AS
BEGIN
   OPEN curs FOR SELECT ename FROM emp
                        WHERE deptno = dept ORDER BY ename;
END;
/
```

如上所示定义empProc存储过程并在embedded SQL调用empProc存储过程开启cursor并FETCH

下一个例子

```
EXEC SQL BEGIN DECLARE SECTION;
SQL_CURSOR empCursor;
int  dept;
char ename[128];
EXEC SQL END DECLARE SECTION;

EXEC SQL ALLOCATE :empCursor;

EXEC SQL CALL empProc( :empCursor,  :dept );

while( 1 )
{
    EXEC SQL FETCH :empCursor INTO :ename;
    if( sqlca.sqlcode == 100 ) break;
```

```
    printf( "%s\n", ename );
}
```

以下是在embedded SQL中定义存储过程并开启cursor变量的例子

```
EXEC SQL BEGIN DECLARE SECTION;
SQL_CURSOR empCursor;
int  dept;
char ename[128];
EXEC SQL END DECLARE SECTION;

EXEC SQL ALLOCATE :empCursor;
EXEC SQL EXECUTE

    BEGIN
        OPEN :empCursor FOR SELECT ename FROM emp
                                WHERE dept_no = :dept ORDER BY ename;
    END;
END-EXEC;

while( 1 )
{
    EXEC SQL FETCH :empCursor INTO :ename;
    if( sqlca.sqlcode == 100 ) break;
    printf( "%s\n", ename );
}
```

如下所示,cursor变量可使用CLOSE命令关闭

```
EXEC SQL CLOSE :empCursor;
```

关闭Cursor变量只是关闭cursor变量而不会返还内存Cursor变量的内存返还可使用FREE命令进行

> Caution:
>
> Cursor变量分配是在内部进行client的statement分配所以与context有关所以cursor变量分配后开启cursor后的FETCH, CLOSE, FREE都只能在同一个context上执行例如default context上分配的cursor变量在其他named context上使用将发生错误

Host Structure

SUNDB的embedded SQL中C的结构体可以作为主机变量使用普通的scalar变量表示单个column相反使用结构体可以简单的表示多个column集合

使用host structure与按顺序列出其member变量的结果相同因此只能在SELECT INTOFETCH INTO的INTO子句和INSERT语句的VALUES子句中使用不能在WHERE子句或UPDATE SET子句等中使用

为了使用host structure首先在declare section中定义结构体声明该结构体变量后与host variable一样使用即可结构体的定义方法与C struct的定义方法相同可通过 typedef定义类型后使用

以下为使用host structure的示例

```
/*
```

```
 * sample4.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1
#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                            \
        printf("\n");                                            \
        printf(aMsg);                                            \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                    \
               SQLSTATE,                                         \
               sqlca.sqlerrm.sqlerrmc );                          \
    }

EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int         mEmpNo;
```

```
    varchar      mEName[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;


int Connect(char *aHostInfo, char *aUserID, char *sPassword);


int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    rsRecord     sRecord;
    EXEC SQL END DECLARE SECTION;
    int  sRecordCount = 0;


    printf("Connect SUNDB ...\n");
    if(Connect("DSN=SUNDB", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
```

```
    FROM   EMP
    ORDER BY EMPNO;


EXEC SQL OPEN EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}


printf(" EMPNO    ENAME               JOB      SALARY\n");
printf("====== ==================== ========== ========\n");
while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sRecord;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
```

```
        }

        sRecordCount ++;

        printf("%6d %20s %10s %8ld\n",
              sRecord.mEmpNo, sRecord.mEName.arr, sRecord.mJob,
sRecord.mSalary);
    }

    printf("====== ==================== ========== ========\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("====== ==================== ========== ========\n");

    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
```

```
    }


    printf("\n\nSUCCESS\n");
    printf("############################\n");


    return 0;


  fail_exit:


    printf("\n\nFAILURE\n");
    printf("############################\n\n");


    EXEC SQL ROLLBACK WORK RELEASE;


    return 0;
}


int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on SUNDB

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB连接

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

为了使用主机变量声明结构体时有约束事项声明结构体时无法使用重叠的结构体如下结构体声

明内有其他结构体时不能作为主机变量使用

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    struct person {
        int         mEmpNo;
        varchar      mEName[20 + 1];
    } person;
    char        mJob[20 + 1];
    long        mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;
```

Indicator Variables

Scalar Indicators

host变量中可以有与其结合的indicator变量indicator变量判断当前host变量是否为空Indicator变量也可以声明后使用但只能声明为C的整数型类型(short, int, long, long long)如下使用indicator

```
:hostvar INDICATOR :hostind
:hostvar :hostind (可省略INDICATOR keyword)
```

Indicator变量值有以下意义

Value	Meaning
-1	NULL
>= 0	input主机变量值

Table 4-9 Input indicator值

Value	Meaning
-1	NULL
0	Value均存储在host变量
>0	由于host变量的buffer size不足而值无法全部存储到host变量时的数据库数据的长度

Table 4-10 Output indicator值

以下为使用indicator的示例

```
/*
 * sample5.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;
```

```
#define  SUCCESS  0
#define  FAILURE  -1
#define  PRINT_SQL_ERROR(aMsg)                                          \
    {                                                                  \
        printf("\n");                                                   \
        printf(aMsg);                                                   \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                           \
               SQLSTATE,                                               \
               sqlca.sqlerrm.sqlerrmc );                                \
    }

int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int         sEmpNo;
    varchar     sEName[20 + 1];
    char        sJob[20];
    long        sSalary;
    int         sDeptNo;
```

```
    int         sDeptNoInd;
    EXEC SQL END DECLARE SECTION;
    int         sRecordCount = 0;
    int         state = 0;


    printf("Connect SUNDB ...\n");
    if(Connect("DSN=SUNDB", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }


    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }
    state = 1;
```

- Update Dept NULL where deptno = 10

```
EXEC SQL
        UPDATE EMP_TEMP
        SET    DEPTNO = NULL
        WHERE  DEPTNO = 10;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
```

```
        goto fail_exit;
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal, deptno
        FROM   EMP_TEMP
        ORDER BY empno;


    EXEC SQL OPEN EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }


    printf(" EMPNO        ENAME          JOB      SALARY  DEPTNO\n");
    printf("====== ==================== ========== ======== ======\n");
    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary, :sDeptNo :sDeptNoInd;
        if(sqlca.sqlcode == SQL_NO_DATA)
```

```
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sRecordCount ++;

    if(sDeptNoInd == -1)
    {
        printf("%6d %20s %10s %8ld (null)\n",
               sEmpNo, sEName.arr, sJob, sSalary);
    }
    else
    {
        printf("%6d %20s %10s %8ld %4d\n",
               sEmpNo, sEName.arr, sJob, sSalary, sDeptNo);
    }
}

printf("====== ==================== ========== ======== ======\n");
printf("Record Count = %d\n", sRecordCount);
```

```
printf("====== ==================== ========== ======== ======\n");


EXEC SQL CLOSE EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}


state = 0;
if(DropEmpTempTable() != SUCCESS)
{
    goto fail_exit;
}


EXEC SQL COMMIT WORK RELEASE;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}


printf("\n\nSUCCESS\n");
printf("############################\n");
```

```
    return 0;


  fail_exit:


    printf("\n\nFAILURE\n");
    printf("############################\n\n");


    switch( state )
    {
        case 1:
            (void)DropEmpTempTable();
            break;
        default:
            break;
    }


    EXEC SQL ROLLBACK WORK RELEASE;


    return 0;
}


int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
```

```
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on SUNDB

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB连接

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");
```

```
    return FAILURE;
}
```

- Create table

```
int CreateEmpTempTable()
{
    EXEC SQL DROP TABLE IF EXISTS EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        CREATE TABLE EMP_TEMP (
            EMPNO NUMBER(4) CONSTRAINT PK_EMP_TEMP PRIMARY KEY,
            ENAME VARCHAR2(10),
            JOB VARCHAR2(9),
            SAL NUMBER(7,2),
            DEPTNO NUMBER(2) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

```
    EXEC SQL
        INSERT INTO EMP_TEMP
        SELECT * FROM EMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

Structure Indicators

Host变量为scalar变量时声明indicator变量并结合使用但host变量为structure时不能对结构体的构成变量各自声明indicator如此host变量为结构体时indicator也要声明为结构体后使用

声明indicator结构体时遵守以下事项

- Indicator与host变量结构体有相同的成员数量
- Indicator结构体的成员应均为整数型数据类型

例如如下声明结构体时结构体变量为4个因此Indicator结构体成员也应为4个

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int         mEmpNo;
    varchar      mEName[20 + 1];
    char        mJob[20 + 1];
    long        mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;
```

即如下声明

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecordInd
{
```

```
    int         mEmpNoInd;
    int         mENameInd;
    int         mJobInd;
    int         mSalaryInd;
} rsRecordInd;
EXEC SQL END DECLARE SECTION;
```

Indicator结构体和与其结合的主机变量结构体按照顺序1：1对应即如上声明结构体时其拥有如下结合关系

```
:rsRecordVar INDICATOR :rsRecordIndVar
```

Host variable	对应的indicator
rsRecordVar.mEmpNo	rsRecordIndVar.mEmpNoInd
rsRecordVar.mEName	rsRecordIndVar.mENameInd
rsRecordVar.mJob	rsRecordIndVar.mJobInd
rsRecordVar.mSalary	rsRecordIndVar.mSalaryInd

Embedded SQL

Host Variable

Host variable用作应用程序与SUNDB之间传递数据的媒介应用程序向SUNDB传递数据时叫做input host variable从SUNDB中获取数据时叫做output host variable两种变量的声明方法没有区

别使用的SQL语句决定其角色

SELECT或FETCH语句的INTO子句的host variable是从SUNDB接收数据的output host variable其余的均为input host variable执行SQL语句之前需设置input host variable值

Host Indicators

Host variable直接使用C语言的变量因此无法表示NULL此时可使用indicator变量一个indicator变量对应一个host variable

Indicator变量值有以下意义

Value	Meaning
-1	NULL
>= 0	input主机变量值

Table 4-11 Input indicator值

Value	Meaning
-1	NULL
0	所有值均已存储在host变量
>0	由于host变量的buffer size不足而导致值无法存储在host变量时的DB data长度

Table 4-12 output indicator值

Insert NULL

在列（column）中如下插入空值

```
EXEC SQL INSERT INTO EMP ( EMPNO, DEPTNO ) VALUES ( :empno, NULL );
```

如像上述通过hard coding创建应用程序将大幅降低灵活性以下为使用indicator变量的示例

```
deptno_ind = -1;
EXEC SQL INSERT INTO EMP ( EMPNO, DEPTNO ) VALUES
( :empno, :deptno :deptno_ind );
```

Indicator变量的deptno_ind的值为-1时不管host变量的deptno值识别为空值

Fetch NULL

SUNDB接收数据时可使用Indicator变量判断是否为空值

参考以下示例

```
EXEC SQL DECLARE CUR_1 CURSOR FOR
    SELECT  EMPNO, DEPTNO
    FROM    EMP
    WHERE   EMPNO = :emp_number;

EXEC SQL OPEN CUR_1;

while( 1 )
```

```
{
    EXEC SQL FETCH CUR_1 INTO :empno, :deptno :deptno_ind;
    if( sqlca.sqlcode == SQL_NO_DATA )
    {
        break;
    }

    if( deptno_ind == -1 )
    {
        printf( "empno : %d, deptno : (null)\n", empno );
    }
    else
    {
        printf( "empno : %d, deptno : %d\n", empno, deptno );
    }
}

EXEC SQL CLOSE CUR_1;
```

FETCH后deptno_ind为-1时判断为deptno为空

Basic SQL statments

Embedded SQL中可以使用SUNDB提供的所有SQL语句SQL语句相关详细内容参考**SQL Manual**Embedded SQL中使用SQL语句时在EXEC SQL关键字后面加SQL语句本章节说明Data Definition Language (DDL)和Data Manipulation Language (DML)相关内容下一个章节说明query

等反复获取数据的语句

执行SQL语句后通过SQLCA检查判断SQL语句的成功与否详细内容参考**Handling Runtime Errors**

DDL Statement

DDL statement为生成删除变更SUNDB的tableviewindex等对象的SQL语句在Embedded SQL应用程序执行DDL语句时在EXEC SQL关键字后面加SQL语句

```
EXEC SQL DROP TABLE IF EXISTS EMP;


EXEC SQL
    CREATE TABLE EMP (
        EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
        ENAME VARCHAR2(10),
        JOB VARCHAR2(9),
        SAL NUMBER(7,2),
        DEPTNO NUMBER(2) );
```

在DDL语句不能使用host variable因此以下为错误的用法

```
strcpy( table_name, "T1" );
EXEC SQL CREATE TABLE :table_name ( C1 INTEGER );
```

如果创建应用程序的过程中无法确定DDL语句而如上使用可变的DDL语句时可如下使用dynamic SQL语句

```
strcpy( table_name, "T1" );
sprintf( sql_stmt, "CREATE TABLE %s ( C1 INTEGER )", table_name );
EXEC SQL EXECUTE IMMEDIATE :sql_stmt;
```

详细内容参考**Embedded Dynamic SQL**

Select Into Statement

使用query获取SUNDB的数据大部分情况无法知道对应结果的数量因此为了使用query需要声明cursor并经过openfetchclose等复杂的过程 其过程的具体执行方式参考**Cursors**

但是特殊情况下也可以已知结果的数量比如已知主键并查找与主键拥有相同key的记录时可预测最多有一条记录或没有数据像这样返回的结果数小于1条时可使用的语句为select into语句可如下执行

```
EXEC SQL
    SELECT ename, job, sal
    INTO :emp_name, :job_title, :salary
    FROM emp
    WHERE empno = :emp_number;
```

> Note:
>
> 使用host array时可以通过Select Into语句获取多条结果Host array的使用方法参考
>
> **Host Arrays**

Insert Statement

Insert语句用于在表插入row可使用host variable决定column值也可使用indicator插入空值

以下为Insert语句的使用示例

```
EXEC SQL
    INSERT INTO EMP ( empno, ename, job, sal )
    VALUES ( :emp_number, :emp_name, :job_name :job_ind, :saraly );
EXEC SQL
    INSERT INTO DEPT ( deptno, dname, loc )
    VALUES ( 1, :dept_name, NULL );
```

> Note:
>
> 使用host structure时不个别使用host variable而以structure为单位插入

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int         mEmpNo;
    varchar      mEName[20 + 1];
    char        mJob[20 + 1];
    long        mSalary;
} rsRecord;

rsRecord  sInsertRec;
```

```
EXEC SQL END DECLARE SECTION;


sInsertRec.mEmpNo = 3000;

strcpy( sInsertRec.mEName.arr, "John" );

sInsertRec.mEName.len = strlen( sInsertRec.mEName.arr );

strcpy( sInsertRec.mJob , "RND" );

sInsertRec.mSalary = 3500;


EXEC SQL INSERT INTO EMP ( empno, ename, job, sal ) VALUES

( :sInsertRec );
```

> Note:
>
> 使用host array可以一次性插入多条row详细内容参考**Host Arrays**

Update Statement

Update语句用于变更表的特定row的column值可使用host variable决定column值也可使用indicator插入空值

以下为update语句的示例

```
EXEC SQL

    UPDATE emp

    SET sal = :salary, deptno = :dept_number :deptno_ind

    WHERE empno = :emp_number;
```

> Note:
>
> 使用host array可以一次性变更多条row详细内容参考**Host Arrays**

Delete Statement

通过delete语句删除表的特定row

以下为delete语句的示例

```
EXEC SQL
    DELETE FROM emp
    WHERE empno = :emp_number;
```

> Note:
>
> 使用host array可以一次性删除多条row详细内容参考**Host Arrays**

PSM Statement

PSM语句在服务器内部创建并使用过程（procedure）或函数（function）

PSM相关详细内容参考**PSM Manual**

默认从EXEC SQL EXECUTE开始以END-EXEC; 结束但创建procedure或function时使用EXEC SQL

而不是EXEC SQL EXECUTE

以下为创建并调用过程的语句示例

```
EXEC SQL
```

```
    CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
      IS
        V1 INTEGER;
      BEGIN


        SELECT COUNT(*)
          INTO V1
          FROM T1
          WHERE T1.I1 >= A1 AND T1.I1 <= A2;


        DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
      END;
END-EXEC;


EXEC SQL CALL PROC1( 2, 4 );
```

以下为创建并调用函数的语句示例

```
EXEC SQL BEGIN DECLARE SECTION;
    int         sV1 = 0;
EXEC SQL END DECLARE SECTION;


EXEC SQL
    CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
      RETURN INTEGER
      IS
```

```
        V1 INTEGER;
      BEGIN


        SELECT COUNT(*)
          INTO V1
          FROM T1
          WHERE T1.I1 >= A1 AND T1.I1 <= A2;


        RETURN V1;
      END;
END-EXEC;


EXEC SQL CALL FUNC1( 2, 4 ) INTO :sV1;
```

以下为anonymous block语句的示例

```
EXEC SQL EXECUTE
    DECLARE
      V1 INTEGER := 0;
      FUNCTION FUNC1( A1 INTEGER )
        RETURN INTEGER
        IS
        BEGIN
          RETURN A1 * 10;
        END;
    BEGIN
```

```
      V1 := FUNC1( 10 );
      DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    END;
END-EXEC;
```

```
EXEC SQL EXECUTE
    DECLARE
      PROCEDURE PROC1( A1 INTEGER )
      IS
      BEGIN
        DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
      END;
    BEGIN
      PROC1( 100 );
    END;
END-EXEC;
```

Cursor

应用程序通过执行查询从SUNDB中获取数据通常情况下执行查询时无法准确知道其查询结果的数量因此使用cursorCursor为表示查询结果集中当前row的位置的标识符可通过以下运算操作cursor

Declare Cursor

声明游标声明游标时需要叙述游标名与与其对应的语句声明的游标名将用于后续其他游标操作命令

以下为游标声明语句的示例

```
EXEC SQL
    DECLARE RECORD_CUR1 CURSOR FOR
    SELECT   empno, ename, dept
    FROM     SEMP
    ORDER BY empno;
```

游标名是由预编译器识别并使用的标识符与实际C程序的变量无关Cursor的scope是文件因此即使在其他文件定义相同名称的cursor也会视为不同的cursor即declare/ open/ fetch/ close cursor的语句应位于相同的source文件中

Open Cursor

打开游标

```
EXEC SQL OPEN <cursor_name>;
Example)
EXEC SQL OPEN EMP_CURSOR;
```

打开游标即为准备好获取执行已声明游标的查询的结果集但这并非实际获取结果实际获取结果应执行**Fetch cursor**

直到关闭当前游标执行语句时使用的主机变量不影响结果集

```
EXEC SQL DECLARE EMP_CURSOR CURSOR FOR
        SELECT    empno, ename, dept
        FROM      EMP
        WHERE     empno < :sNo
        ORDER BY  empno;


    sNo = 100;
    EXEC SQL OPEN EMP_CURSOR;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }


    while( 1 )
    {
        sNo = 10;
        EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;


        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
```

```
        {
            goto fail_exit;
        }
        ...
    }
```

以上使用示例中游标打开为sNo = 100时即使在过程中将sNo更改为其他值直到关闭此游标之前结果集不会发生变化

Fetch Cursor

获取游标位置的row

```
EXEC SQL FETCH <cursor_name> INTO <host_variable_list>;
Example)
EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;
```

执行FETCH运算前应提前声明并打开对应游标首次执行FETCH时游标移动至结果集的第一行并将其指定为current row 之后将current row插入到INTO子句的主机变量后返回之后反复执行FETCH时游标将下一条row更新为current row, 反复执行将current row返回为INTO子句的主机变量的操作执行FETCH而不再有结果时 在sqlca.sqlcode返回SQL_NO_DATA 因此应用程序可通过其代码值判断检索是否结束

Close Cursor

关闭游标

```
EXEC SQL CLOSE <cursor_name>;
```

```
Example)
EXEC SQL CLOSE EMP_CURSOR;
```

关闭游标时对应游标应为打开的状态关闭游标后该游标无法再执行FETCH关闭游标后为了再次使用重新打开游标时其已成为新打开的游标因此可能与之前的结果集不同

重复使用cursor

可重复使用已声明的cursor名本章节说明为了重复使用相同cursor名的先后关系

Cursor语句先后关系

- DECLARE CURSOR

  可在执行cursor CLOSECOMMITROLLBACK语句后执行

- OPEN

  可在FETCH所有cursor后或执行CLOSE语句后执行

- FETCH

  可在执行cursor OPEN语句后执行

- CLOSE

  可在执行cursor OPEN语句或FETCH语句后执行

Cursor语句和Host Variable

Cursor语句可放在文件中的任何位置但根据用于DECLARE语句的 input host variable为全域变量或局域变量cursor语句的适用方法也有所不同

- 用于DECLARE语句的input host variable为局域变量时OPEN语句必须位于与DECLARE语句

相同的函数内

- 用于DECLARE语句的input host variable为全域变量时OPEN语句可位于与DECLARE语句不同的函数
- DECLARE语句中没有input host variable时OPEN语句可位于与DECLARE语句不同的函数

由于在内部存储cursor declare语句中使用的host variable的指针并在cursor open语句使用指针而两个语句位于不同的函数并使用的host variable为局域变量时open语句参考的是无效的指针因此推荐将declare语句和open语句放在相同的函数中

Cursor使用示例

以下为使用DDLDMLcursor的示例

```
/*
 * overview.gc
 *
 * Connect / Disconnect
 * DDL(Create/Drop table)
 * Basic DML(Insert, Delete, Update)
 * Standing Cursor
 */
EXEC SQL INCLUDE SQLCA;


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

```
#define  SUCCESS  0
#define  FAILURE  -1
#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                            \
        printf("\n");                                            \
        printf(aMsg);                                            \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                    \
               SQLSTATE,                                         \
               sqlca.sqlerrm.sqlerrmc );                          \
    }

EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsEmpRecord
{
    int      mEmpNo;
    char     mEName[20];
    char     mDept[10];
} rsEmpRecord;

rsEmpRecord gRecord[10] = {
    { 1, "Park", "RND" },
    { 2, "Kim", "CEO" },
    { 3, "Choi", "SALES" },
```

```
    { 4, "Lee", "CTO" },
    { 5, "Lyu", "RND" },
    { 6, "Ohn", "SUPPORT" },
    { 7, "Cheon", "RND" },
    { 8, "Sohn", "SALES" },
    { 9, "Smith", "WAIT" },
    { 10, "mycomman", "WAIT" }
};
EXEC SQL END DECLARE SECTION;


int CreateEmpTable();
int DropEmpTable();
int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int Disconnect();
int PrintRecord();


int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int         sEmpNo;
    char        sDept[10 + 1];
    EXEC SQL END DECLARE SECTION;
    int  sState = 0;


    printf("Connect SUNDB ...\n");
```

```
if(Connect("DSN=SUNDB", "test", "test") != SUCCESS)
{
    goto fail_exit;
}
sState = 1;

printf("Create SEMP table ...\n");
if(CreateEmpTable() != SUCCESS)
{
    goto fail_exit;
}
sState = 2;

printf("Insert record ...\n");
EXEC SQL
    INSERT INTO SEMP(empno, ename, dept)
    VALUES(:gRecord);
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf("====== ==================== ==========\n");
```

```
printf("%d Record Inserted\n", sqlca.sqlerrd[2]);
printf("====== ==================== ==========\n");

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf("Current record\n");
PrintRecord();

sEmpNo = 9;
printf("Delete record WHERE empno == %d\n", sEmpNo);
EXEC SQL
    DELETE FROM SEMP
    WHERE  empno = :sEmpNo;

if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}
```

```
printf("After Delete record\n");
PrintRecord();


EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}


strcpy( sDept, "RND" );
printf("Update record WHERE dept == 'WAIT'\n");
EXEC SQL
    UPDATE SEMP
    SET    dept = :sDept
    WHERE  dept = 'WAIT';
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}


printf("After Update record\n");
PrintRecord();
```

```
EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

sState = 1;
printf("Drop SEMP table ...\n");
if(DropEmpTable() != SUCCESS)
{
    goto fail_exit;
}

sState = 0;
printf("Disconnect SUNDB ...\n");
if(Disconnect() != SUCCESS)
{
    goto fail_exit;
}

printf("SUCCESS\n");
printf("##############################\n");

return 0;
```

```
  fail_exit:

    printf("FAILURE\n");
    printf("############################\n\n");

    EXEC SQL ROLLBACK WORK;
    switch(sState)
    {
        case 2:
            printf("Drop SEMP table ...\n");
            (void)DropEmpTable();
        case 1:
            printf("Disconnect SUNDB ...\n");
            (void)Disconnect();
            break;
        default:
            break;
    }

    return 0;
}
```

- Create table

```
int CreateEmpTable()
```

```
{


EXEC SQL DROP TABLE IF EXISTS SEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL CREATE TABLE SEMP ( empno       INTEGER,
                                 ename       VARCHAR(20),
                                 dept        VARCHAR(10),
                                 PRIMARY KEY (empno) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;
```

```
  fail_exit:


    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");


    EXEC SQL ROLLBACK WORK;


    return FAILURE;
}
```

- Drop table

```
int DropEmpTable()
{
    EXEC SQL DROP TABLE SEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }


    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }


    return SUCCESS;
```

```
  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on SUNDB

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB连接

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;

    if(sqlca.sqlcode != 0)

    {

        goto fail_exit;

    }


    return SUCCESS;


  fail_exit:


    PRINT_SQL_ERROR("[ERROR] Connection Failure!");


    return FAILURE;

}


int Disconnect()

{

    EXEC SQL DISCONNECT;

    if(sqlca.sqlcode != 0)

    {

        goto fail_exit;

    }
```

```
    return SUCCESS;


  fail_exit:


    PRINT_SQL_ERROR("[ERROR] [ERROR] Disconnect Failure!");


    return FAILURE;
}


int PrintRecord()
{
    EXEC SQL BEGIN DECLARE SECTION;
    rsEmpRecord sResultRecord;
    EXEC SQL END DECLARE SECTION;
    int   sRecordCount = 0;


    EXEC SQL
        DECLARE RECORD_CUR1 CURSOR FOR
        SELECT   empno, ename, dept
        FROM     SEMP
        ORDER BY empno;


    EXEC SQL OPEN RECORD_CUR1;
    if(sqlca.sqlcode != 0)
    {
```

```
    goto fail_exit;
}

printf(" EMPNO    ENAME              DEPT\n");
printf("====== ==================== ==========\n");
while( 1 )
{
    EXEC SQL FETCH RECORD_CUR1 INTO :sResultRecord;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    sRecordCount ++;

    printf("%6d %20s %10s\n",
           sResultRecord.mEmpNo,
           sResultRecord.mEName,
           sResultRecord.mDept);
}
```

```
    printf("====== ==================== ==========\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("====== ==================== ==========\n");

    EXEC SQL CLOSE RECORD_CUR1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    return FAILURE;
}
```

Cursor Property

前一章仅讲述了最基本的游标游标可以拥有多种属性根据不同的属性可执行多种功能SUNDB的Embedded SQL提供ISO/IEC-9075-2 SQL Foundation建议的游标属性和ODBC建议的游标属性定义游标的语句和属性相关详细内容参考**DECLARE cursor_name**

Scrollable Cursor

SCROLL是ISO type的游标属性决定游标的scroll支持与否支持scroll的游标可以在FETCH语句中接收位置选项并FETCH根据此选项的位置的row不支持scroll的游标只能按顺序FETCH结果集

以下为声明scroll游标时使用SCROLL选项的示例

```
EXEC SQL DECLARE cur_scroll SCROLL CURSOR FOR SELECT c1, c2 FROM t1;
```

使用NO SCROLL选项时会声明不支持scroll的游标省略scroll选项时默认为NO SCROLL

Scroll游标可以在FETCH时指定位置信息该位置信息叫做fetch orientation如下表

Fetch orientation	Description
NEXT	fetch当前位置的下一个row
PRIOR	fetch当前位置的上一个row
FIRST	fetch结果集的第一个row
LAST	fetch结果集的最后一个row
CURRENT	fetch当前位置的row
ABSOLUTE <position>	• 在结果集fetch属于position位置的row • Position值为负数时从AFTER THE LAST ROW开始fetch之前位置的row
RELATIVE <position>	fetch距离当前位置有与position相同距离的row

Table 4-13 Fetch orientation

Fetch orientation为ABSOLUTERELATIVE时追加要求<position>值此<position>值可以为整数型常数或整数型类型的host variable以下为使用fetch orientation的示例

```
EXEC SQL FETCH NEXT;

EXEC SQL FETCH PRIOR;

EXEC SQL FETCH FIRST;

EXEC SQL FETCH LAST;

EXEC SQL FETCH CURRENT;

EXEC SQL FETCH ABSOLUTE 100;

EXEC SQL FETCH RELATIVE :position;
```

Sensitive Cursor

Sensitivity是ISO type的游标属性运用游标的过程中结果集发生变化时决定是否可查看其变更内容Sensitivity有以下三种选项

Option	Description
SENSITIVE	可以查看其他事务变更或删除的内容
INSENSITIVE	无法查看打开游标后变更或删除的内容
ASENSITIVE	根据<query>的内容决定SENSITIVE/ INSENSITIVE

Table 4-14 Sensitivity

省略sensitive选项时默认值为INSENSITIVE如下声明使用sensitive选项的游标

```
EXEC SQL DECLARE <cur_name> SENSITIVE CURSOR FOR SELECT c1, c2 FROM t1;

EXEC SQL DECLARE <cur_name> INSENSITIVE CURSOR FOR SELECT c1, c2 FROM t1;

EXEC SQL DECLARE <cur_name> ASENSITIVE CURSOR FOR SELECT c1, c2 FROM t1;
```

Holdable Cursor

Holdability是ISO type的游标属性决定提交打开当前游标的事务后是否继续维持当前游标

Holdability有以下两种选项

Option	Description
WITH HOLD	• 事务提交后仍维持游标 • 不能与FOR UPDATE语句一起使用 • 不能与INSERT INTO ... RETURNING语句一起使用 • 不能与UPDATE ... RETURNING语句一起使用 • 不能与DELETE FROM ... RETURNING语句一起使用 • 不能用于包括table commit action为ON COMMIT DELETE ROWS的global temporary table的查询
WITHOUT HOLD	COMMIT/ROLLBACK事务后关闭游标
Rollback与Cursor	• 回滚事务时关闭事务包含的游标 • 回滚到保存点时关闭保存点以后生成的游标

Table 4-15 Holdability

省略holdable option时默认值取决于<cursor updatability>

- 未指定FOR READ ONLY或<cursor updatability>时为WITH HOLD
- 与FOR UPDATE语句一起使用时为WITHOUT HOLD

如下声明使用holdable option的游标

```
EXEC SQL DECLARE <cur_name> CURSOR WITH HOLD FOR SELECT c1, c2 FROM t1;

EXEC SQL DECLARE <cur_name> CURSOR WITHOUT HOLD FOR SELECT c1, c2 FROM t1
FOR UPDATE;
```

Static Cursor

Static游标是ODBC type的游标属性与ISO type的INSENSITIVE SCROLL游标相同

```
EXEC SQL DECLARE cur_static STATIC CURSOR FOR SELECT c1, c2 FROM t1;
```

Static cursor的fetch方法参考 **Scrollable Cursor**

Keyset Driven Cursor

Keyset driven游标是ODBC type的游标属性与ISO type的ASENSITIVE SCROLL游标相同

```
EXEC SQL DECLARE cur_static KEYSET CURSOR FOR SELECT c1, c2 FROM t1;
```

Keyset driven游标也有scroll功能相关fetch方法参考**Scrollable Cursor**

Positioned DML

Positioned DML可使用CURRENT OF <cursor_name>语句对最后fetch的row执行delete或update操作为了执行positioned DML游标应为打开的状态并至少执行一次fetch后游标指向对应row

以下为执行positioned DML的示例

```
EXEC SQL DECLARE emp_cursor CURSOR FOR
     SELECT ename, sal FROM emp WHERE job = 'SALES'
     FOR UPDATE;
...

EXEC SQL OPEN emp_cursor;
EXEC SQL WHENEVER NOT FOUND GOTO ...

while( 1 )
{
    EXEC SQL FETCH emp_cursor INTO :emp_name, :salary;
    ...
    EXEC SQL UPDATE emp SET sal = :new_salary
         WHERE CURRENT OF emp_cursor;
}
```

Options

本章节说明预编译embedded SQL源代码时可使用的选项

Precompiled Header Files

编写embedded SQL程序时将多个源代码共同引用的内容记录在单独的头文件并使源代码包含该头文件这样更有效率在C语言中使用#include语句支持其功能但通过C语言的#include包含头文件时不会预编译对象文件而由C语言编译器解析因此文件内的声明部分等不会被预编译器转换

这样创建头文件时如果要使预编译器转换此头文件并插入到源代码则需要使用EXEC SQL INCLUDE语句其语句如下

```
EXEC SQL INCLUDE <filename>;
```

此语句预编译指定文件并将其插入到源代码中

指定Header File路径

一般情况下查找头文件时优先搜索当前源代码所在的目录但是普遍单独收集头文件即使没有收集通常也会由于各种原因位于不同的路径中

此时可如下单独指定搜索头文件的目录路径其语句形式如下

```
EXEC SQL OPTION( INCLUDE = <directory path> );
```

可以列出并指定多个此选项通过EXEC SQL INCLUDE查找头文件时按照此选项指定的顺序搜索目录

> Note:
>
> 此选项也可以使用预编译器的Command-line option指定相关内容参考**Precompiler Options--include-path, -I**

Host Array

目前为止仅说明了主机变量拥有单一值的scalar变量本章节说明批量处理主机变量的方法

使用Host Array可以简化源代码并能提高性能但使用方面存在一些制约条件因此需注意

Host Array的声明

Host array的声明与scalar variable声明相似以数组形式声明变量即可以下为声明数组大小为10的host array的示例

```
EXEC SQL BEGIN DECLARE SECTION;
    int    empno[10];
    char   ename[10][20];
    double salary[10];
EXEC SQL END DECLARE SECTION;
```

声明host array时有以下制约条件

- 不允许二维以上的数组
- 但字符串系列(char, varchar)和binary系列(binary, varbinary)由表示array size的数组与表示

data size的数组构成因此可使用二维数组

- 不允许pointer的数组

Host Array的使用

访问Host Array

在SQL语句中使用SQL语句的方法与使用scalar host variable的用法相同

以下为host array的简单示例

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值

```
...
```

- 插入emp_numberemp_namedept_number值

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

上述示例与以下代码执行相同操作

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值

```
...
```

- 插入emp_numberemp_namedept_number

```
for( i = 0; i < 20; i ++ )
{
    EXEC SQL INSERT INTO emp (empno, ename, deptno)
        VALUES (:emp_number[i], :emp_name[i], :dept_number[i]);
}
```

使用多个host variable array时各host array中仅对拥有最小array size的进行运算以上示例中所有host variable的大小为20因此插入20条row但如下dept_number的数组大小指定为10时最终插入10条row

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[10];
EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值

```
...
```

- 插入emp_numberemp_namedept_number

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

Host Indicator Array的使用

Host变量为数组时对应的Indicator变量也应为数组而且indicator变量的数组大小和host变量的数组大小应相同在上述示例中增加indicator变量时如下所示

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
int    emp_number_ind[20];
char   emp_name[20][10];
int    emp_name_ind[20];
int    dept_number[20];
int      dept_number_ind[20];
EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值
- 设置emp_number_indemp_name_inddept_number_ind的各indicator值

```
...
```

- 插入emp_numberemp_namedept_number

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number :emp_number_ind,
            :emp_name :emp_name_ind,
            :dept_number :dept_number_ind);
```

制约条件

- 使用多个主机变量时不能混合使用scalar变量和array变量
- Select语句的WHERE子句中不能使用host array
- Updatedelete语句的CURRENT OF子句中不能使用host array

INTO子句中的Array

使用SELECT INTO语句或游标可以从SUNDB获取rowEmbedded SQL中的这两种方法均使用INTO子句并且在INTO子句中使用host array可获取多条row

SELECT INTO中的Array

已知要获取的row数时可使用**Select Into statement**简单实现在**Select Into statement**中说明了没有结果或只有一条结果的情况但在这里使用host array可获取多条row

使用方法与使用scalar变量的方法相同但以array声明主机变量就可以编写使用array的select into语句

```
EXEC SQL BEGIN DECLARE SECTION;
char   emp_name[50][20];
```

```
int    emp_number[50];
float  salary[50];
EXEC SQL END DECLARE SECTION;


EXEC SQL SELECT ENAME, EMPNO, SAL
    INTO :emp_name, :emp_number, :salary
    FROM EMP
    WHERE SAL > 1000;
```

在以上示例中只需将host变量声明为数组就会编写一个通过SELECT INTO语句获取50条row的语句 由于此语句拥有独立的执行单元因此仅获取符合查询条件的前50行

> Caution:
>
> 即使满足条件的行数超过50条通过SELECT INTO语句不能获取后面的第51行之后的row如果要连续fetch row应使用游标

使用Cursor时的Array

无法知道当前查询的结果集数量时应使用游标使用游标的方法参考**Cursors**声明游标后在FETCH语句的INTO子句中使用array主机变量时可一次性获取多个row

```
EXEC SQL BEGIN DECLARE SECTION;
    int   emp_number[50];
    char  emp_name[50][20];
    char  dept_name[50][20];
EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL DECLARE EMP_CURSOR CURSOR FOR
        SELECT    empno, ename, dept
        FROM      EMP
        WHERE     empno < :sNo
        ORDER BY  empno;


    sNo = 100;
    EXEC SQL OPEN EMP_CURSOR;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }


    while( 1 )
    {
        EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;


        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
```

```
        }
        ...
    }
```

通过在FETCH语句中使用host array可以一次性获取50条row

sqlca.sqlerrd[2]

使用array时获取与声明的数组大小相同的的row但也会有由于没有row而无法获取与数组大小相同的row的情况例如声明的数组大小为50但实际结果集的row数为30条时只能获取30条row为了处理这种情况SUNDB的embedded SQL在sqlca.sqlerrd[2]提供当前执行的语句处理了几条row的信息

在sqlca.sqlerrd[2]返回INSERTUPDATEDELETESELECT INTOFETCH语句实际处理的row数

```
EXEC SQL BEGIN DECLARE SECTION;
    int   emp_number[50];
    char  emp_name[50][20];
    char  dept_name[50][20];
EXEC SQL END DECLARE SECTION;


EXEC SQL DECLARE EMP_CURSOR CURSOR FOR
        SELECT    empno, ename, dept
        FROM      EMP
        WHERE     empno < :sNo
        ORDER BY  empno;
```

```
    sNo = 100;
    EXEC SQL OPEN EMP_CURSOR;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    while( 1 )
    {
        EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;

        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }

        for( i = 0; i < sqlca.sqlerrd[2]; i ++ )
        {
            printf( "%d %s %s\n", emp_number[i], emp_name[i],
dept_name[i] );
        }
```

```
        ...
    }
```

sqlca的详细内容参考**Handling Runtime Errors**

Insert语句中的Array

INSERT语句中以array声明主机变量时可实现array insert

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值

```
...
```

- 插入emp_numberemp_namedept_number

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

上述示例与以下代码执行相同的操作

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
```

```
char   emp_name[20][10];

int    dept_number[20];

EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值

```
...
```

- 插入emp_numberemp_namedept_number

```
for( i = 0; i < 20; i ++ )

{

    EXEC SQL INSERT INTO emp (empno, ename, deptno)

        VALUES (:emp_number[i], :emp_name[i], :dept_number[i]);

}
```

> Note:
>
> * 使用Array Insert时主机变量应均为array变量或均为scalar变量
>
> * 通过查看sqlca.sqlerrd[2]可查看实际插入的row数

Atomic Insert

Atomic insert为array insert的特殊形式有以下两个特点

- Insert过程中只要有一条失败则所有的插入操作将失败
- 应用程序向SUNDB只传送一次命令因此性能优越

Atomic insert使用如下ATOMIC关键字

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值

```
...
```

- 插入emp_numberemp_namedept_number值

```
EXEC SQL ATOMIC INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

Update语句中的Array

以下为在update语句中使用array的示例

```
EXEC SQL BEGIN DECLARE SECTION;
char  job_title [10][20];
float commission[10];
EXEC SQL END DECLARE SECTION;


...
```

```
EXEC SQL UPDATE emp SET comm = :commission
    WHERE job = :job_title;
```

上述使用示例与以下功能相同

```
EXEC SQL BEGIN DECLARE SECTION;
char  job_title [10][20];
float commission[10];
EXEC SQL END DECLARE SECTION;

...

for( i = 0; i < 10; i ++ )
{
    EXEC SQL UPDATE emp SET comm = :commission[i]
        WHERE job = :job_title[i];
}
```

Note:

* 使用array update时主机变量应均为array变量或均为scalar变量

* 通过查看sqlca.sqlerrd[2]查看实际更新的row数

* UPDATE语句的CURRENT OF子句中无法使用array

Delete语句中的Array

在delete语句中可如下使用array

```
EXEC SQL BEGIN DECLARE SECTION;
char job_title[10][20];
EXEC SQL BEGIN DECLARE SECTION;


...
EXEC SQL DELETE FROM emp
    WHERE job = :job_title;
```

上述示例与以下执行相同的功能

```
EXEC SQL BEGIN DECLARE SECTION;
char job_title[10][20];
EXEC SQL BEGIN DECLARE SECTION;


...
for( i = 0; i < 10; i ++ )
{
    EXEC SQL DELETE FROM emp
        WHERE job = :job_title[i];
}
```

> Note:

> * 使用array delete时主机变量应均为array变量或均为scalar变量
>
> * 通过查看sqlca.sqlerrd[2]可查看删除的row数
>
> * DELETE语句的CURRENT OF子句中无法使用array

使用FOR语句

执行SQL时如果要指定处理的数组大小则使用FOR语句以下语句中可以使用FOR语句

- SELECT INTO
- FETCH
- INSERT
- UPDATE
- DELETE

For语句的用法如下

```
EXEC SQL FOR :host_variable <sql_stmt>

EXEC SQL FOR <integer_constant> <sql_stmt>
```

以下为for语句的使用示例

```
EXEC SQL BEGIN DECLARE SECTION;

int    emp_number[20];

char   emp_name[20][10];

int    dept_number[20];

int      record_cnt;
```

```
EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值

```
...
```

- 插入emp_numberemp_namedept_number

```
record_cnt = 10;
EXEC SQL FOR :record_cnt INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

以上示例中host array的大小为20但通过FOR语句指定执行与record_cnt数量相同的数量因此实际仅插入了10条数据

> Note:
>
> FOR语句不能在UPDATE/DELETE CURRENT OF语句中使用

结构体Array

一般的scalar变量可通过数组一次性处理多条row但一个变量只能表示一个column一个主机变量要访问多个column时使用**Host Structure**即可如此将host变量声明为结构体并将其结构体用作array则可一次性处理拥有多个column的多条row

以下情况下可使用结构体数组

- Output host variable array: SELECT INTOFETCH INTO语句

- Input host variable array: INSERT语句的VALUES项目

制约条件

以下情况不能使用结构体数组

- WHERE子句或FROM子句
- UPDATE语句的SET子句

声明结构体数组

结构体声明一般与C语言的结构体声明方式相同可以直接声明结构体变量或通过typedef声明type后声明host变量

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int          mEmpNo;
    varchar      mEName[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
rsRecord     sRecord[10];

struct {
    int          mEmpNo;
    varchar      mEName[20 + 1];
```

```
    char        mJob[20 + 1];
    long        mSalary;
} sResultRecord[10];
EXEC SQL END DECLARE SECTION;
```

声明使用主机变量的结构体时不能使用重叠的结构体如下结构体声明内存在其他结构体时不能作为主机变量使用

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    struct person {
        int         mEmpNo;
        varchar     mEName[20 + 1];
    } person;
    char        mJob[20 + 1];
    long        mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;
```

结构体数组的Indicators

Structure Indicators中已说明如果主机变量为结构体则Indicator变量也应为结构体因此主机变量为结构体数组时indicator变量也应为对应的结构体数组

Indicator结构体数组声明遵守以下事项

- Indicator的成员数量应与要结合的host变量结构体的成员数量相同
- Indicator结构体的成员应均为整数型数据类型
- Indicator结构体数组大小应与host变量结构体数组大小相同如果indicator结构体数组的大小小则执行SQL语句时需使用FOR子句限制数组执行次数

Structure与Scalar变量的混用

Host structure传送至SUNDB时按照顺序列出其structure member以下为混用host structure与scalar变量的使用示例

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct rsEmp
    {
        int         mEmpNo;
        varchar      mEName[20 + 1];
    } rsEmp;

    rsEmp     sEmp[5];
    char      sJob[5][20 + 1];
    long      sSalary[5];
EXEC SQL END DECLARE SECTION;

EXEC SQL
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM   EMP
```

```
    ORDER BY EMPNO;
EXEC SQL OPEN EMP_CUR;


EXEC SQL
    FETCH EMP_CUR
    INTO  :sEmp, :sJob, :sSalary;
```

以上FETCH语句中同时使用rsEmp结构体和sJobsSalary变量其在内部被解析为sEmp.mEmpNosEmp.mENamesJobsSalary四个变量用这种方法可混合使用host structure和host scalar variable

以下为混合使用host structure arraystructure array indicatorstructure和scalar变量的示例

```
/*
 * fetch_struct_array.gc
 *  : structure array fetch
 *  : structure indicators
 *  : mix structure, scalar variable
 *  : sqlca.sqlerrd[2]
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


EXEC SQL INCLUDE SQLCA;
```

```
#define  SUCCESS  0
#define  FAILURE  -1

#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                             \
        printf("\n");                                             \
        printf(aMsg);                                             \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                     \
               SQLSTATE,                                          \
               sqlca.sqlerrm.sqlerrmc );                          \
    }

EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsEmp
{
    int         mEmpNo;
    varchar     mEName[20 + 1];
} rsEmp;
EXEC SQL END DECLARE SECTION;

int Connect(char *aHostInfo, char *aUserID, char *sPassword);

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

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    rsEmp     sEmp[5];
    char      sJob[5][20 + 1];
    long      sSalary[5];
    EXEC SQL END DECLARE SECTION;
    int  sRecordCount = 0;
    int  i;

    printf("Connect SUNDB ...\n");
    if(Connect("DSN=SUNDB", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP
        ORDER BY EMPNO;

    EXEC SQL OPEN EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
```

```
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf(" EMPNO    ENAME                JOB      SALARY\n");
printf("====== ==================== ========== ========\n");

while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmp, :sJob, :sSalary;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sRecordCount += sqlca.sqlerrd[2];

    for( i = 0; i < sqlca.sqlerrd[2]; i ++ )
```

```
        {
            printf("%6d %20s %10s %8ld\n",
                  sEmp[i].mEmpNo, sEmp[i].mEName.arr, sJob[i],
sSalary[i]);
        }
    }
    printf("====== ==================== ========== ========\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("====== ==================== ========== ========\n");

    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("\n\nSUCCESS\n");
```

```
    printf("############################\n");


    return 0;


  fail_exit:


    printf("\n\nFAILURE\n");
    printf("############################\n\n");


    EXEC SQL ROLLBACK WORK RELEASE;


    return 0;
}


int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on SUNDB

```
strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
```

```
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB连接

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

Handling Runtime Errors

概要

应用程序在执行过程中有可能达不到预期的执行结果因此编写embedded SQL应用程序时需要应对这种异常情况本章节说明检测执行SQL后返回的执行结果的方法

Runtime Error的检测

SQLCA

Embedded SQL中发生的所有错误信息都存储于SQL Communication Area (SQLCA)领域因此应用程序查看SQLCA的内容时可查看当前操作的成功与否以及报错时的错误类型

SQLCA为存储errorwarningSQL语句执行状态等的资料结构SQLCA仅有结合其资料结构的最后一个SQL语句的执行结果不存储SQL语句执行的历史记录相关内容执行Embedded SQL时将丢失现有的SQLCA的内容因此为了处理异常执行SQL后要立即查看SQLCA并采取必要的措施

SQLCA的使用

通过如下语句使用SQLCA

```
EXEC SQL INCLUDE SQLCA;
```

上述语句在预编译过程中置换为以下语句

```
#include "sqlca.h"
```

需在使用第一个embedded SQL语句之前提前使用此语句通常建议位于源代码的上端

SUNDB的embedded SQL应用程序中sqlca一般为全局变量因此在single threa程序中不需要额外的声明即可使用sqlca但是在多线程程序同时访问sqlca时发生并发性错误因此需要单独声明详细内容参考**Multithread Applications**

SQLCA的结构

以下为sqlca的结构体

```
struct sqlca
{
    char    sqlcaid[8];    ❶ 初始化为字符串SQLCA
    int     sqlabc;        ❷ sqlca结构体的大小
```

- 最近执行的语句中产生的error code为0时成功正数时warning负数时error

```
int     sqlcode;
```

- 存储属于sqlcode的错误信息
- .sqlerrml是sqlerrmc的长度
- .sqlerrmc以string的形式存储错误信息

```
struct
    {
        unsigned short sqlerrml;
        char           sqlerrmc[SQLERRMC_LEN];
```

```
    } sqlerrm;


    char    sqlerrp[8];    ❸ unused
    int     sqlerrd[6];
```

- 0: empty
- 1: empty
- 2: INSERTUPDATEDELETE后处理的row的数量
- 3: empty
- 4: empty
- 5: empty

```
char    sqlwarn[8];
```

- 0: 产生任何一个warning时均为'W'
- 1: SELECTFETCH中结果string被truncate时为'W'
- 2: unused
- 3: unused
- 4: unused
- 5: unused
- 6: unused
- 7: unused

```
char    sqlext[8];     ❹ unused
char    sqlstate[8];   ❺ SQLSTATE
unsigned short  *rowstatus;     ❻ fetched row status array
```

```
};
```

下面详细介绍各个构成要素

SQLCODE

如下定义SQLCODE

```
#define SQLCODE         (sqlca.sqlcode)
```

SQLCODE返回embedded SQL的执行结果代码SQLCODE最初是在ISO / IEC-9075中提出的但在SQL-92中被deprecate但是很多程序仍在使用因此为了兼容性SUNDB提供SQLCODE以下为执行结果代码

SQLCODE	结果
0	成功
> 0	发生Warning
SQL_NO_DATA	没有结果
发生< 0	Error

Table 4-16 SQLCODE执行结果

可通过以下方法查看sqlcode可使用sqlca.sqlcode或SQLCODE

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
```

```
    SELECT empno, ename, job, sal
    FROM   EMP;


EXEC SQL OPEN EMP_CUR;
if(SQLCODE != 0)
{
    goto fail_exit;
}


while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmpNo, :sEName, :sJob, :sSalary;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }


    ...
}
```

SQLSTATE

如下定义SQLSTATE

```
#define SQLSTATE        (sqlca.sqlstate)
```

在ISO/IEC-9075中SQLCODE被deprecate后替代使用SQLSTATESQLSTATE由5个字符（数字英文大小写字母）组成前两位为Class后三位为Subclass以下为SQLSTATE的执行结果

SQLSTATE	结果
00000	成功
01xxx	产生Warning
02000	没有结果
除外的所有State	产生Error

Table 4-17 SQLSTATE执行结果

查看SQLSTATE代替SQLCODE修改代码为如下可使用sqlca.sqlstate或SQLSTATE

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;


    EXEC SQL OPEN EMP_CUR;
    if(strcmp(SQLSTATE, "00000") != 0)
```

```
{
    goto fail_exit;
}

while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmpNo, :sEName, :sJob, :sSalary;
    if(strcmp(sqlca.sqlstate, "02000") == 0)
    {
        break;
    }
    else if(strcmp(sqlca.sqlstate, "00000") != 0)
    {
        goto fail_exit;
    }

    ...
}
```

处理的row数

执行updatedelete语句或使用array的insertselect intofetch语句时仅显示已处理的行数该信息存储在sqlca.sqlerrd[2]中应用程序可以通过参考对应字段的值来查看已处理的行数使用sqlca.sqlerrd[2]的示例如下

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct rsEmp
    {
        int         mEmpNo;
        varchar      mEName[20 + 1];
    } rsEmp;
    rsEmp     sEmp[5];
    char      sJob[5][20 + 1];
    long      sSalary[5];
EXEC SQL END DECLARE SECTION;


    EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;


    EXEC SQL OPEN EMP_CUR;
    if(SQLCODE != 0)
    {
        goto fail_exit;
    }


    while( 1 )
    {
        EXEC SQL
```

```
        FETCH EMP_CUR
        INTO  :sEmpNo, :sEName, :sJob, :sSalary;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    sRecordCount += sqlca.sqlerrd[2];
    ...
}
```

仅限于fetch语句sqlca.sqlerrd[2]的值可以处理为累积的row个数相关的内容请参考gpec(Precompiler)的参数 **--cumulative**

已处理的row的状态

sqlca.rowstatus表示当前已处理的row的状态使用scroll sensitive cursor或使用where CURRENT OF更新/删除row时可以更改row的状态使用array执行SQL语句时拥有与array大小相同的row status

row数	Row状态
1条	*sqlca.rowstatus

row数	Row状态
Array size n	sqlca.rowstats[0] sqlca.rowstats[1] sqlca.rowstats[2] ... sqlca.rowstats[n-1]

Table 4-18 Row状态

Row status	Description
SQL_ROW_SUCCESS	Row状态正常
SQL_ROW_DELETED	Row被删除
SQL_ROW_UPDATED	Row被更新
SQL_ROW_NOROW	Row不存在
SQL_ROW_ADDED	已增加row
SQL_ROW_ERROR	Row状态不正常

Table 4-19 Row状态值

以下为参考row status的示例

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct rsEmp
    {
```

```
        int         mEmpNo;
        varchar      mEName[20 + 1];
    } rsEmp;
    rsEmp     sEmp[5];
    char      sJob[5][20 + 1];
    long      sSalary[5];
EXEC SQL END DECLARE SECTION;


    EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;


    EXEC SQL OPEN EMP_CUR;
    if(SQLCODE != 0)
    {
        goto fail_exit;
    }


    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
        if(sqlca.sqlcode == SQL_NO_DATA)
```

```
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }

        sRecordCount += sqlca.sqlerrd[2];

        for( i = 0; i < sqlca.sqlerrd[2]; i ++ )
        {
            if( sqlca.rowstatus[i] == SQL_ROW_SUCCESS )
            {
                printf("%6d %20s %10s %8ld\n",
                        sEmp[i].mEmpNo, sEmp[i].mEName.arr, sJob[i],
sSalary[i]);
            }
        }

        ...
    }
```

Error Message Text

执行embedded SQL后执行结果发生error或warning时可以以文本形式传送相关信息错误信息存

储于sqlca.sqlerrm其中sqlca.sqlerrm.sqlerrml为文本的长度实际信息存储于sqlca.sqlerrm.sqlerrmc当应用程序中出现异常时可输出错误信息文本后将信息传输给用户以下为使用error message text的示例

```
EXEC SQL INSERT INTO EMP VALUES ( :sEmp );
if( SQLCODE != 0 )
{
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",
           SQLCODE,
           SQLSTATE,
           sqlca.sqlerrm.sqlerrmc );
}
```

Warning Flags

sqlca.sqlwarn用作执行embedded SQL后发生warning时标记warning的flag由8个char array构成发生warning时留下'W'标记

Warning flag	Description
sqlca.sqlwarn[0]	至少发生一个Warning时记录'W'
sqlca.sqlwarn[1]	Select IntoFetch的结果字符串被truncate时记录'W'
sqlca.sqlwarn[2]	reserved
sqlca.sqlwarn[3]	reserved
sqlca.sqlwarn[4]	reserved

Warning flag	Description
sqlca.sqlwarn[5]	reserved
sqlca.sqlwarn[6]	reserved
sqlca.sqlwarn[7]	reserved

Table 4-20 Warning flag

Handling Implicit Error

执行embedded SQL语句后必须查看SQLCA并处理执行结果但执行embedded SQL语句后处理相同的异常时可使用WHENEVER指示符自动处理其异常

WHENEVER语句的使用

以下为WHENEVER语句的语法

```
EXEC SQL WHENEVER conditions actions;
```

WHENEVER Condition

以下为WHENEVER语句的condition

```
<conditions> ::=
      SQLERROR
    | SQLWARNING
    | NOT FOUND
```

```
    | SQLSTATE <sqlstate class value>[<sqlstate subclass value>]

    ;


<sqlstate_char> ::= [0-9A-Z];

<sqlstate class value> ::= <sqlstate_char><sqlstate_char>;

<sqlstate subclass value> ::=

<sqlstate_char><sqlstate_char><sqlstate_char>;
```

Condition	Description
SQLERROR	执行embedded SQL的过程中发生了error
SQLWARNING	执行embedded SQL的过程中发生了warning
NOT FOUND	没有结果row
SQLSTATE <sqlstate>	执行embedded SQL的过程中发生了SQLSTATE <sqlstate>

Table 4-21 Conditions of WHENEVER statement

WHENEVER Action

Action说明了满足上述condition时实际执行的操作其语法为如下

```
<actions> ::=

      CONTINUE

    | GOTO <label>

    | STOP

    | DO <c statements>
```

```
;
```

Action	Description
CONTINUE	不做任何操作即忽略指定的condition
GOTO <label>	以<label>分开程序流
STOP	结束程序的执行
DO <c statements>	执行<c statements>

Table 4-22 Actions of WHENEVER statement

WHENEVER语句的适用范围

WHENEVER语句定义conditions的actions对一个condition只能定义一个actions因此使用WHENEVER语句定义下一个condition之前对所有的Embedded SQL语句使用相同的actions

WHENEVER语句按照各个conditions单独管理因此特定时间点可使用的WHENEVER语句的最大数量为4个对同一个condition适用新的WHENEVER语句时将取消原有的action后续开始适用新的action以下为相关示例

```
EXEC SQL WHENEVER SQLERROR STOP;
EXEC SQL INSERT INTO emp VALUES ( :emp_number, :emp_name, :salary ); ❶
...
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL UPDATE emp SET sal = sal * 1.1 WHERE sal < :sal_bound; ❷
...
```

```
EXEC SQL WHENEVER SQLERROR GOTO exit_label;
EXEC SQL WHENEVER NOT FOUND DO break;
EXEC SQL DECLARE EMP_CURSOR FOR
        SELECT empno, ename, sal
        FROM   emp;

EXEC SQL OPEN EMP_CURSOR; ❸

EXEC SQL WHENEVER SQLERROR GOTO close_label;
while( 1 )
{
    EXEC SQL FETCH EMP_CURSOR
             INTO   :emp_number, :emp_name, :salary; ❹

    printf( "emp number : %d, emp name : %s, salary : %lf\n",
            emp_number, emp_name, salary );
}

close_label:
EXEC SQL WHENEVER SQLERROR DO sql_error();
EXEC SQL CLOSE EMP_CURSOR; ❺
...
exit_label:
...
```

Line 1中SQLERROR时将STOP因此如果在执行❶语句的过程中报错则结束应用程序Line 4中SQLERROR时将CONTINUE因此SQLERROR的action变更为不执行任何操作因此在执行❷语句的过程中报错也将继续执行下一个操作Line 7中SQLERROR的Action设置为转移到exit_label因此执行❸语句时报错则转移到exit_labelLine 8中当NOT FOUND时将执行break并且在Line 15中将SQLERROR的action重新指定为close_label因此对❹语句将适用2种条件即FETCH过程中如果报错则转移到close_label没有结果时执行break语句Line 26中SQLERROR的action指定为执行sql_error()因此执行❺语句的过程中报错则调用sql_error()函数

使用WHENEVER语句时的注意事项

使用WHENEVER语句时应准确掌握其运行原理并注意使用使用WHENEVER语句时需注意以下几点

- 源代码中的WHENEVER语句位置：WHENEVER语句只是向预编译器告知自动插入exception handling代码的指示符而不是逻辑执行的代码因此使用WHENEVER语句时需在源代码上适用的embedded SQL语句的前面填写而且为了不影响后续出现的其他embedded SQL的执行必须使用CONTINUE action进行初始化
- 使用breakcontinue关键字：在action语句中使用DO breakDO continue时必须确认loop的领域参考如下示例

```
EXEC SQL WHENEVER SQLERROR GOTO fail_exit;
EXEC SQL WHENEVER NOT FOUND DO break; ❶


EXEC SQL DECLARE EMP_CURSOR FOR
        SELECT empno, ename, sal
        FROM   emp;
```

```
EXEC SQL OPEN EMP_CURSOR;
while( 1 )
{
    EXEC SQL FETCH EMP_CURSOR
             INTO   :emp_number, :emp_name, :salary;  ❷
    printf( "emp number : %d, emp name : %s, salary : %lf\n",
            emp_number, emp_name, salary );
}

EXEC SQL CLOSE EMP_CURSOR;

EXEC SQL
    SELECT MAX(sal)
    INTO   :max_salary
    FROM   emp;          ❸
...
```

❶中对NOT FOUND的action定义了DO break; 因此之后的SQL执行中发生NOT FOUND时执行break; ❷中在执行FETCH的过程中发生NOT FOUND时将执行break并退出while loop但是❸中发生NOT FOUND时即使要执行break也因为由于不是loop而无法执行break这种情况在创建源程序后制作应用程序的过程中已出现编译错误

- 防止无限loop

Action使用转移时有可能发生无限循环参考以下示例

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
...
EXEC SQL INSERT INTO emp VALUES ( :emp_number, :emp_name, :salary );
...
sql_error:
    EXEC SQL ROLLBACK WORK RELEASE;
```

执行SQL的过程中如果发生错误则转移到sql_error label此时如果在嵌入式SQL语句的执行过程中发生error则转移到sql_error但在处理此错误的过程中执行另一个嵌入式SQL并且此处重复出现错误时应用程序将陷入无限循环 在这种情况下建议如下初始化错误处理部分

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
...
EXEC SQL INSERT INTO emp VALUES ( :emp_number, :emp_name, :salary );
...
sql_error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK WORK RELEASE;
```

- 转移label的scope

Action中转移时对应的label必须存在于可访问的位置参考以下示例

```
func1()
{
```

```
    EXEC SQL WHENEVER SQLERROR GOTO labelA;
    EXEC SQL DELETE FROM emp WHERE deptno = :dept_number;
    ...
labelA:
...
}

func2()
{

    EXEC SQL INSERT INTO emp (job) VALUES (:job_title);
    ...
}
```

func1()中对SQLERROR指定了labelA转移func1()有labelA因此没有问题但func2()中没有labelA因此将出现编译报错此时应在func2()也要创建同样的labelA或如下初始化action

```
func1()
{

    EXEC SQL WHENEVER SQLERROR GOTO labelA;
    EXEC SQL DELETE FROM emp WHERE deptno = :dept_number;
    ...
labelA:
...
}
```

```
func2()
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL INSERT INTO emp (job) VALUES (:job_title);
    ...
}
```

4.3 Advanced Topic

Embedded Dynamic SQL

概要

大部分embedded SQL应用程序对SUNDB有具体的操作以向特定表插入更新删除查询row为目的并使用叫做SQL的数据库语言描述其操作方法为此直接在嵌入式SQL源代码输入SQL则预编译器将解析SQL语句并在已知指定SQL内容和input/ output host variable的情况下转换为可在SUNDB执行的API调用

但是有些应用程序在编写程序时无法提前知道SQL语句例如GUI工具等应用程序的用户为了选择并查询运算类型表名称条件等而使用static SQL时需要提前生成用户可选择的所有组合的SQL但这不符合实际即使可行也是效率非常低的操作此时如果可以通过用户选择的选项生成并执行SQL则编写程序会变得更加灵活这种未在源代码定义SQL而在执行时间点可变型的SQL叫做dynmic SQLSUNDB提供嵌入式dynamic SQL功能

Embedded dynamic SQL应用程序相比静态SQL提供更灵活的使用方法但编写源代码会更加复杂即使执行同一个语句相比静态SQL执行速度慢

因此编写应用程序时考虑到这种特征需慎重选择静态SQL或动态SQL

本章节说明编写embedded dynamic SQL程序的方法

Dynamic SQL的类型

Dynamic SQL根据使用方法如下分类

Method	Description	是否支持
Method 1	Non-query，无host variable	O
Method 2	Non-query，有已知其数量和类型的host variable	O
Method 3	Query，有已知其数量和类型的host variable	O
Method 4	Query，不知道host variable的存在与否数量及类型	X

Table 4-23 Dynamic SQL类型

SUNDB目前支持到method 3

Method 1

是最简单的动态SQL可在non-query且没有主机变量的情况使用通常在DDL或没有主机变量的DML中使用

- EXECUTE IMMEDIATE

Method 1为non-query且没有主机变量因此可以立即执行SQL语句SQL的立即执行语法为如下

```
EXEC SQL EXECUTE IMMEDIATE { :host_variable | <string_literal> };
<string_literal> ::=
      ' <sql_statement> '
    | " <sql_statement> "
    | <sql_statement>
    ;
```

如下使用method 1

```
sprintf(sSqlStmt, "CREATE TABLE EMP_RND (\n"
          "EMPNO NUMBER(4) CONSTRAINT PK_EMP_RND PRIMARY KEY,\n"
          "ENAME VARCHAR2(10),\n"
          "JOB VARCHAR2(9),\n"
          "SAL NUMBER(7,2),\n"
          "DEPTNO NUMBER(2) )\n" );
   EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
   if(sqlca.sqlcode != 0)
   {
       goto fail_exit;
   }

   sprintf(sSqlStmt, "INSERT INTO EMP_RND\n"
          "SELECT *\n"
          "FROM   EMP\n"
          "WHERE  JOB = 'RND'\n" );
   EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
   if(sqlca.sqlcode != 0)
   {
       goto fail_exit;
   }
```

Method 2

Method 2在non-query且有input host variable的情况使用此时应提前知道input host variable数量和数据类型通过准备(prepare)和执行(execute)阶段实现

- Prepare

在准备阶段分析SQL statement并向statement赋予名称以下为prepare的语句

```
EXEC SQL PREPARE <statement_name> FROM { :host_variable |
<string_literal> };
<string_literal> ::=
      ' <sql_statement> '
    | " <sql_statement> "
    | <sql_statement>
    ;
```

<statement_name>为告诉预编译器的标识符不是主机变量因此没有单独的类型也不需要声明

- Execute

Execute阶段执行已分析的statement以下为execute语句的语法

```
EXEC SQL EXECUTE <statement_name> [ USING <host_variable_list> ];
<host_variable_list> ::= <host_variable_entry> [ , <host_variable_list> ];
<host_variable_entry> ::= :host_variable
[ [ INDICATOR ] :host_indicator ];
```

有input host variable的情况使用USING子句没有主机变量时省略USING子句按USING语句中出现的主机变量的顺序依次绑定到已prepare的SQL语句的主机变量

Method 2中反复执行同一个SQL语句时仅执行一次prepare后可反复执行execute已prepare的statement有效至在当前源代码中通过相同的statement name准备或断开其他SQL语句为止

Method 3

Method 3为从method 2扩展至可支持query的形式一般要通过query prepare过程解析statement并query因此需要可操作游标所以增加declareopenfetchclose游标的语句

- Prepare

Prepare阶段与method 2相同解析SQL statement并给statement赋予名称以下为prepare的语句

```
EXEC SQL PREPARE <statement_name> FROM { :host_variable |
<string_literal> };
<string_literal> ::=
      ' <sql_statement> '
    | " <sql_statement> "
    | <sql_statement>
    ;
```

<statement_name>为告诉预编译器的标识符不是主机变量因此没有单独的类型也不需要声明

- Declare cursor

Declare语句对已prepare的statement声明游标声明游标时可使用与standing cursor相同的cursor

property相关语句如下

```
EXEC SQL <dynamic declare cursor>;


<dynamic declare cursor> ::=
    DECLARE <cursor_name> <cursor properties> { FOR | IS }
<statement_name>
    ;


<cursor properties> ::=
      [ <cursor sensitivity> ] [ <cursor scrollability>] ] CURSOR
[ <cursor holdability> ]
    | [ <odbc cursor type] CURSOR [ <cursor holdability> ]
    ;


<cursor sensitivity> ::=
      INSENSITIVE
    | SENSITIVE
    | ASENSITIVE
    ;


<cursor scrollability> ::=
      NO SCROLL
    | SCROLL
    ;
```

```
<cursor holdability> ::=
      WITH HOLD
    | WITHOUT HOLD
    ;

<odbc cursor type> ::=
      STATIC
    | KEYSET
    ;
```

<cursor properties>与standing cursor的含义及使用名称相同

<statement_name>为PREPARE语句指定的名称<cursor name>和<statement name>是告诉预编译器的标识符不是主机变量因此没有单独的类型也不需要声明

单个statement只能声明单个游标Statement中已经有声明的游标的状态下再声明新的游标将关闭原来的游标

- Open cursor

打开游标与standing cursor的**Open**基本相同但是在dynamic cursor的open上有重要的区别

Dynamic cursor在运行过程中可随时变更SQL语句因此根据声明的语句的SQL决定主机变量因此打开dynamic cursor时在open时使用USING子句传递主机变量Standing cursor在open时需要游标处于关闭状态但是dynamic cursor会在关闭游标后open

Dynamic cursor的open语句为如下

```
EXEC SQL <dynamic cursor open>;


<dynamic cursor open> ::=

    OPEN <cursor_name> [ USING <host_variable_list> ]

    ;


<host_variable_list> ::= <host_variable_entry> [ , <host_variable_list> ];

<host_variable_entry> ::= :host_variable

[ [ INDICATOR ] :host_indicator ];
```

- Fetch cursor

从游标fetchDynamic cursor的fetch与**Stangding cursor的Fetch**相同

- Close cursor

关闭游标Dynamic cursor的close与**Stangding cursor的Close**相同

Example Program

以下为使用dynamic method 123的示例程序

```
/*

 * dyn2.gc

 *  : dynamic method 1

 *  : dynamic method 2

 *  : dynamic method 3
```

```
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1
#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                            \
        printf("\n");                                            \
        printf(aMsg);                                            \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
              sqlca.sqlcode,                                     \
              SQLSTATE,                                          \
              sqlca.sqlerrm.sqlerrmc );                           \
    }

EXEC SQL BEGIN DECLARE SECTION;
typedef struct Record
{
    int         mEmpNo;
    varchar      mEName[20 + 1];
    char        mJob[20];
```

```
    char        mSalary[10];
} Record;
EXEC SQL END DECLARE SECTION;


int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();
int UpdateSalary(char *aJob, int aBound, double aRatio);


int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    EXEC SQL END DECLARE SECTION;
    printf("Connect SUNDB ...\n");
    if(Connect("DSN=SUNDB", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }


    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }
```

- Print RND employee increate 20% salary where salary < 2000

```
printf("print RND employee increate 20%% salary where salary < 2000\n");
UpdateSalary( "RND", 2000, 1.2 );
printf("\n\n");
```

- Print SUPPORT employee increate 10% salary where salary < 3000

```
printf("print SUPPORT employee increate 10%% salary where salary <
3000\n");
UpdateSalary( "SUPPORT", 3000, 1.1 );
printf("\n\n");

if(DropEmpTempTable() != SUCCESS)
{
     goto fail_exit;
}

printf("Disconnect SUNDB ...\n");
EXEC SQL COMMIT WORK RELEASE;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf("SUCCESS\n");
printf("##############################\n");
```

```
return 0;


  fail_exit:


    printf("\n\nFAILURE\n");
    printf("############################\n\n");
    EXEC SQL ROLLBACK WORK RELEASE;


    return 0;
}


int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on SUNDB

```
strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
```

```
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB连接

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

- Create table

```
int CreateEmpTempTable()
{
    EXEC SQL BEGIN DECLARE SECTION;
    char   sSqlStmt[8192];
```

```
EXEC SQL END DECLARE SECTION;

sprintf(sSqlStmt, "DROP TABLE IF EXISTS EMP_RND" );

EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;

if(sqlca.sqlcode != 0)

{

    goto fail_exit;

}


sprintf(sSqlStmt, "CREATE TABLE EMP_RND (\n"

        "EMPNO NUMBER(4) CONSTRAINT PK_EMP_RND PRIMARY KEY,\n"

        "ENAME VARCHAR2(10),\n"

        "JOB VARCHAR2(9),\n"

        "SAL NUMBER(7,2),\n"

        "DEPTNO NUMBER(2) )\n" );

EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;

if(sqlca.sqlcode != 0)

{

    goto fail_exit;

}


sprintf(sSqlStmt, "INSERT INTO EMP_RND\n"

        "SELECT *\n"

        "FROM   EMP\n"

        "WHERE  JOB = 'RND'\n" );

EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
```

```
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf(sSqlStmt, "DROP TABLE IF EXISTS EMP_SUPPORT" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf(sSqlStmt, "CREATE TABLE EMP_SUPPORT (\n"
        "EMPNO NUMBER(4) CONSTRAINT PK_EMP_SUPPORT PRIMARY KEY,\n"
        "ENAME VARCHAR2(10),\n"
        "JOB VARCHAR2(9),\n"
        "SAL NUMBER(7,2),\n"
        "DEPTNO NUMBER(2) )\n" );
EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf(sSqlStmt, "INSERT INTO EMP_SUPPORT\n"
```

```
            "SELECT *\n"
            "FROM   EMP\n"
            "WHERE  JOB = 'SUPPORT'\n" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_RND;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL DROP TABLE EMP_SUPPORT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
```

```
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

int UpdateSalary(char *aJob, int aBound, double aRatio)
{
    EXEC SQL BEGIN DECLARE SECTION;
    Record      sRecord;
    char        sSelectSql[128];
    char        sUpdateSql[128];
    int         sBound = aBound;
    double      sRatio = aRatio;
    EXEC SQL END DECLARE SECTION;

    int  sRecordCount = 0;
    int  i;
    int  sIsOpenCur = 0;

    sprintf( sSelectSql, "SELECT EMPNO, ENAME, JOB, SAL FROM EMP_%s WHERE
sal < :v1 FOR UPDATE", aJob );
    sprintf( sUpdateSql, "UPDATE EMP_%s SET sal = sal * :v1 WHERE CURRENT
OF DYN_CUR", aJob);
```

```
EXEC SQL PREPARE SELECT_STMT FROM :sSelectSql;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

EXEC SQL DECLARE DYN_CUR KEYSET CURSOR FOR SELECT_STMT;
EXEC SQL OPEN DYN_CUR USING :sBound;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}
sIsOpenCur = 1;

EXEC SQL PREPARE UPDATE_STMT FROM :sUpdateSql;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

while( 1 )
{
    EXEC SQL
        FETCH NEXT DYN_CUR
```

```
        INTO  :sRecord;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL EXECUTE UPDATE_STMT USING :sRatio;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
}

sIsOpenCur = 0;
EXEC SQL CLOSE DYN_CUR;
if(sqlca.sqlcode != 0)
{
    goto fail_exit;
}

sprintf( sSelectSql, "SELECT EMPNO, ENAME, JOB, SAL FROM EMP_%s ORDER
```

```
BY SAL DESC", aJob );
    EXEC SQL PREPARE SELECT_STMT FROM :sSelectSql;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL OPEN DYN_CUR;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    sIsOpenCur = 1;

    printf("%s salary list\n", aJob);
    sRecordCount = 0;
    printf(" EMPNO    ENAME                JOB       SALARY\n");
    printf("====== ==================== ========== ==========\n");
    while( 1 )
    {
        EXEC SQL
            FETCH DYN_CUR
            INTO  :sRecord;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
```

```
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    for(i = 0; i < sqlca.sqlerrd[2]; i ++)
    {
        sRecordCount ++;
        printf("%6d %20s %10s %10s\n",
               sRecord.mEmpNo,
               sRecord.mEName.arr,
               sRecord.mJob,
               sRecord.mSalary);
    }
}

printf("====== ==================== ========== ==========\n");
printf("Record Count = %d\n", sRecordCount);
printf("====== ==================== ========== ==========\n");

sIsOpenCur = 0;
EXEC SQL CLOSE DYN_CUR;
```

```
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    if(sIsOpenCur == 1)
    {
        EXEC SQL CLOSE DYN_CUR;
    }
    return FAILURE;
}
```

Multi-thread Application

Multi-threaded application为一个进程内有多个执行单位的应用程序Multi-threaded application像同时执行多个应用程序一样可并行处理多个操作由于是一个process因此可共享同一个地址领域

共享同一个地址领域是指共享全局（global）变量和静态（static）变量因此编写应用程序时需

考虑到访问这样的变量时各个线程对此的并发性控制

SUNDB为此提供run-time contextrun-time context在 Direct Attach (D/A)模式与Client/Server (C/S)模式稍有差异下一章说明编写run-time context与multi-threaded application的guideline

Run-time Context

SUNDB的embedded SQL中run-time context用于管理应用程序的SUNDB的connectionRun-time context与connection有1:1的对应关系也可将run-time context视为connection本身

SUNDB的结构在D/A模式下仅允许一个线程有一个connection要使用多个connection时应用程序应编写为多线程结构CS模式是通过network与服务器连接因此一个线程内可以有多个连接

Direct Attach (D/A)模式

D/A模式下1个线程只能有一个连接因此编写拥有n个connection的应用程序时需通过n个线程运行

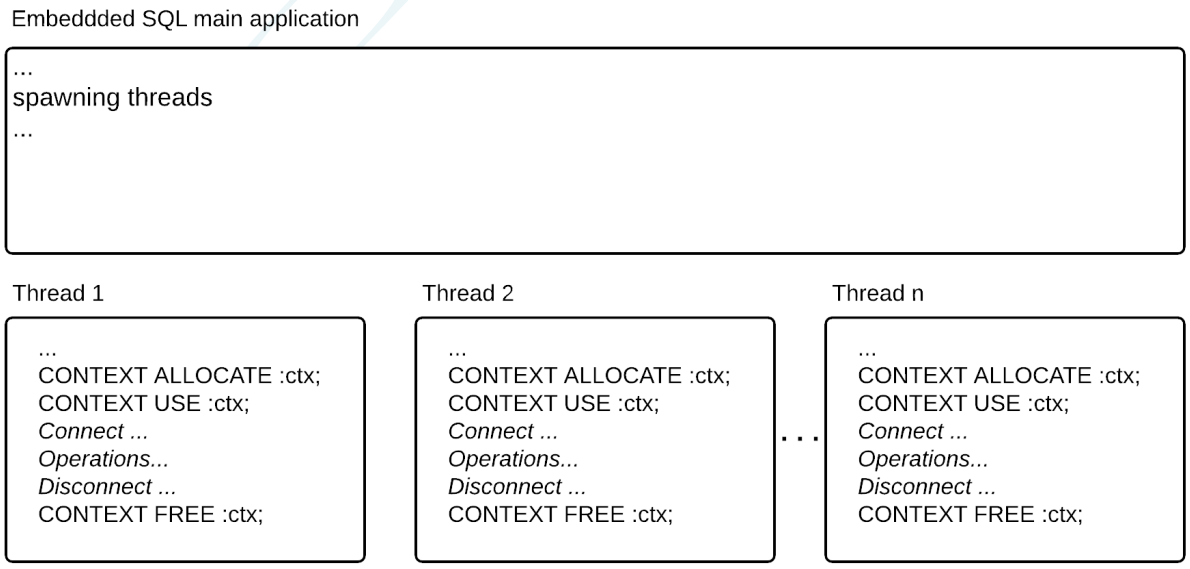


Figure 4-2 D/A模式下每个线程有自己的连接时

Client/Server (C/S)模式

C/S模式下对connection没有额外的约束一个线程可以有多个连接多个线程可以共享一个连接而且n个线程也可以共享m个连接

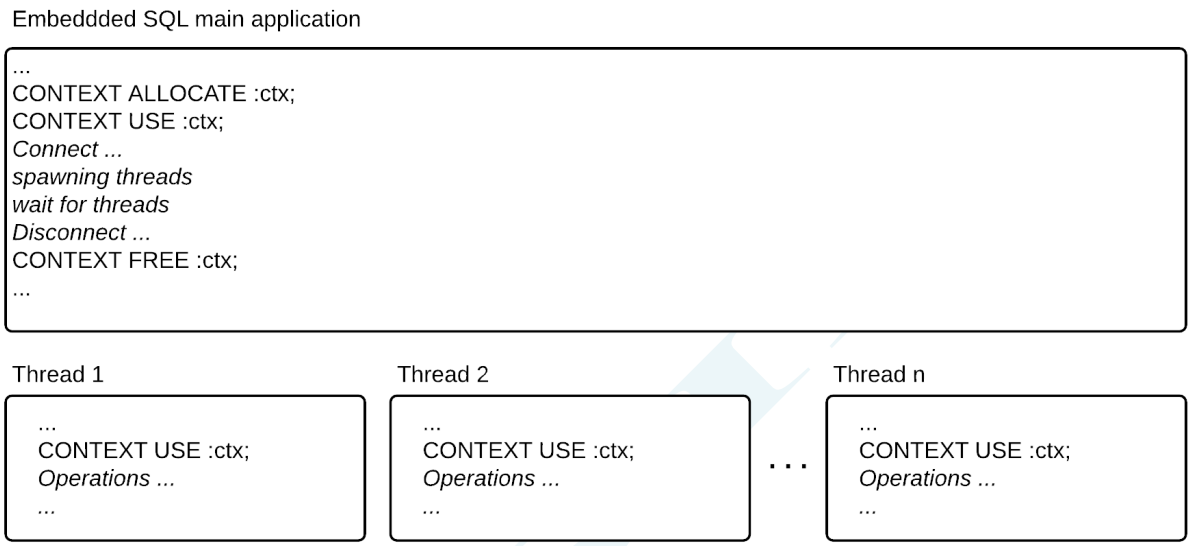


Figure 4-3 多个线程共享1个连接时

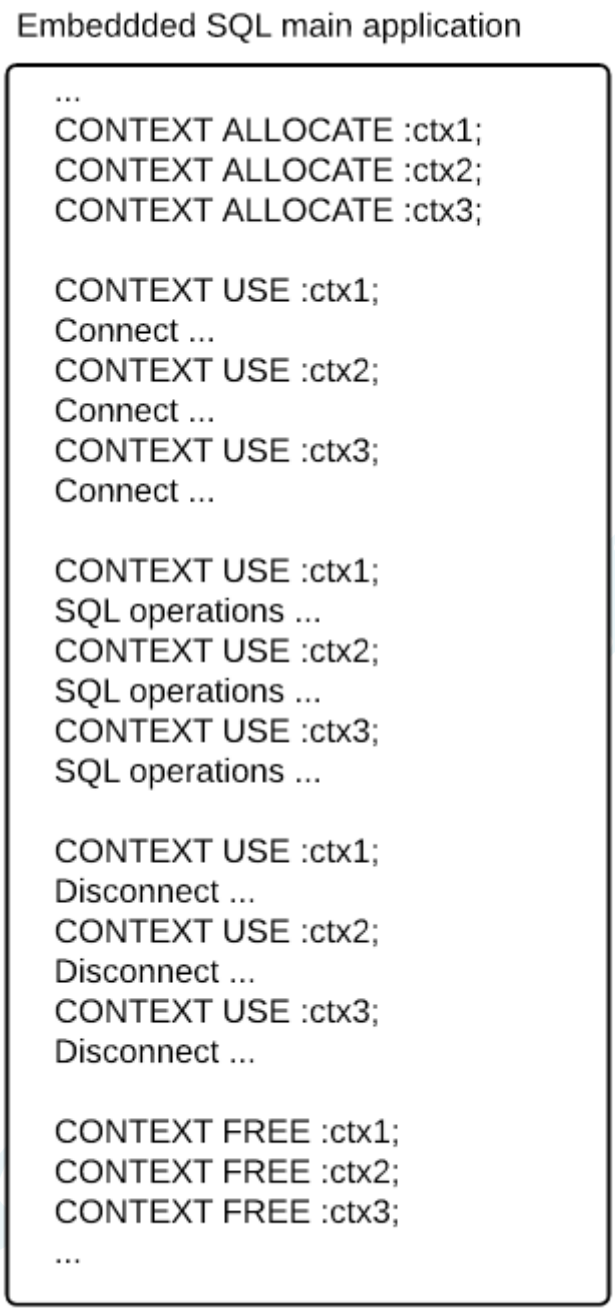


Figure 4-4 1个线程有多个连接时

Guidelines

编写多线程应用程序时需要注意以下几点

- Thread-safe的声明SQLCA变量建议在每个线程内以stack变量声明其示例参考下面的

example program

- 多线程在一个进程内拥有同一个地址领域因此共享static变量或global变量因此使用这些变量时需考虑应用程序的并发性控制
- 一个run-time context不能在多个线程同时使用即使用run-time context时需考虑并发性控制

Example Program

以下为多线程应用的使用示例程序

```
/*
 * thread1.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1

#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                            \
        printf("\n");                                            \
```

```
        printf(aMsg);                                                \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                        \
               SQLSTATE,                                             \
               sqlca.sqlerrm.sqlerrmc );                              \
    }


int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char
*sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();
void *clientThread(void *args);


typedef struct thread_param
{
    int    mNo;
    char  *mJobName;
} thread_param;


#define  THREAD_COUNT    2


char gJobName[THREAD_COUNT][20]= {
    "RND",
    "SUPPORT"
};
```

```
int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int         sEmpNo;
    varchar     sEName[20 + 1];
    char        sJob[20];
    long        sSalary;
    EXEC SQL END DECLARE SECTION;
    int         sRecordCount = 0;
    pthread_t    thread_id[THREAD_COUNT];
    thread_param param[THREAD_COUNT];
    int         i;

    printf("Connect SUNDB ...\n");
    if(Connect(NULL, "DSN=SUNDB", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }

    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }
```

- Create client thread

```
for( i = 0; i < THREAD_COUNT; i ++ )
    {
        param[i].mNo = i;
        param[i].mJobName = gJobName[i];
        if( pthread_create(&thread_id[i],
                           NULL,
                           clientThread,
                           &param[i]) != 0 )
        {
            printf( "Can't create thread %d!\n", i );
        }
        else
        {
            printf( "Create thread %d!\n", i );
        }
    }

    for( i = 0; i < THREAD_COUNT; i ++ )
    {
        if( pthread_join(thread_id[i],
                         NULL) != 0 )
        {
            printf( "Error when waiting for thread %d to terminate!\n",
```

```
i );
        }
        else
        {
            printf( "Stopped thread %d!\n", i );
        }
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP_TEMP
        ORDER BY empno;


    EXEC SQL OPEN EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }


    printf(" EMPNO    ENAME                JOB      SALARY\n");
    printf("====== ==================== ========== ========\n");
    while( 1 )
```

```
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmpNo, :sEName, :sJob, :sSalary;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sRecordCount ++;

    printf("%6d %20s %10s %8ld\n",
           sEmpNo, sEName.arr, sJob, sSalary);
}

printf("====== ==================== ========== ========\n");
printf("Record Count = %d\n", sRecordCount);
printf("====== ==================== ========== ========\n");

EXEC SQL CLOSE EMP_CUR;
```

```
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    if(DropEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("SUCCESS\n");
    printf("##############################\n");

    return 0;

fail_exit:
```

```
    printf("FAILURE\n");
    printf("############################\n\n");
    EXEC SQL ROLLBACK WORK RELEASE;

    return 0;
}

int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char
*sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
    struct sqlca sqlca;
```

- Log on SUNDB

```
strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB连接

```
if( aCtx != NULL )
    {
        EXEC SQL CONTEXT USE :aCtx;
        EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    }
    else
    {
        EXEC SQL CONTEXT USE DEFAULT;
        EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    }

    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
```

```
}
```

- DB连接解除

```
int Disconnect(sql_context aCtx)
{
    struct sqlca sqlca;
    if( aCtx != NULL )
    {
        EXEC SQL CONTEXT USE :aCtx;
        EXEC SQL DISCONNECT;
    }
    else
    {
        EXEC SQL CONTEXT USE DEFAULT;
        EXEC SQL DISCONNECT;
    }
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
```

```
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");


    return FAILURE;
}
```

- Create table

```
int CreateEmpTempTable()
{
    EXEC SQL DROP TABLE IF EXISTS EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        CREATE TABLE EMP_TEMP (
            EMPNO NUMBER(4) CONSTRAINT PK_EMP_TEMP PRIMARY KEY,
            ENAME VARCHAR2(10),
            JOB VARCHAR2(9),
            SAL NUMBER(7,2),
            DEPTNO NUMBER(2) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

```
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

```
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

void *clientThread(void *args)
{
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT   my_context;
    char          job_name[20 + 1];
    EXEC SQL END DECLARE SECTION;
    int           state = 0;
    thread_param *param = (thread_param *)args;
```

```
EXEC SQL CONTEXT ALLOCATE :my_context;

state = 1;


EXEC SQL CONTEXT USE :my_context;

if(Connect(my_context, "DSN=SUNDB", "test", "test") != SUCCESS)

{

    goto fail_exit;

}

state = 2;


strcpy( job_name, param->mJobName );


EXEC SQL

    INSERT INTO EMP_TEMP

    SELECT *

    FROM   EMP

    WHERE  JOB = :job_name;

if(sqlca.sqlcode != 0)

{

    goto fail_exit;

}


EXEC SQL COMMIT WORK;

if(sqlca.sqlcode != 0)

{
```

```
        goto fail_exit;
    }
    state = 1;

    if(Disconnect(my_context) != SUCCESS)
    {
        goto fail_exit;
    }
    state = 0;

    EXEC SQL CONTEXT FREE :my_context;
    pthread_exit(0);

    return NULL;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    switch(state)
    {
        case 2:
            (void)Disconnect(my_context);
        case 1:
            EXEC SQL CONTEXT FREE :my_context;
            break;
```

```
        default:
            break;
    }

    pthread_exit(0);

    return NULL;
}
```

C++ Application

Output Filename的扩展名

SUNDB的预编译器(gpec)预编译embedded SQL源代码后将生成C代码因此输出文件的扩展名默认为 .c一般情况下C++源代码根据编译器的种类拥有多种形式的文件扩展名

通过gpec的输出文件指定选项-o指定预编译后的文件的扩展名例如如下将testfile.gc转换为testfile.cpp

```
gpec $(GPEC_OPT) testfile.gc -o testfile.cpp
```

SQLCA_STORAGE_CLASS

在C++程序以global变量声明相同的symbol时symbol会发生冲突SUNDB的embedded SQL默认拥有sqlca变量因此与多个C++文件链接时此变量会发生冲突

为了避免此问题除一个文件外其他所有文件中应定义SQLCA_STORAGE_CLASS macro假设通过链接三个文件创建应用程序时在三个文件中的两个文件中如下定义SQLCA_STORAGE_CLASS macro

```
#define SQLCA_STORAGE_CLASS extern

EXEC SQL INCLUDE SQLCA;
```

定义SQLCA_STORAGE_CLASS macro必须位于以下语句之前

```
EXEC SQL INCLUDE SQLCA;
```

XA

xa_open string定义

xa_open字符串包含连接Resource Manager (RM)的信息详细内容参考**SQLDriverConnect**

以下为xa_open字符串的使用示例

```
DSN=SUNDB;UID=test;PWD=test;CONN_NAME=XA_CONN
```

预编译器中的XA使用

预编译器中使用XA时可选择使用以下中的一个

- 使用default connection
- 使用named connection

使用Default Connection

使用仅通过连接信息连接xa_open字符串的connection

以下为使用default connection的xa_open字符串的示例

```
DSN=SUNDB;UID=test;PWD=test
```

使用使用default connection的embedded SQL时以如下使用

```
EXEC SQL
        UPDATE Deposit
        Set InterestRates = :value :value_ind
        WHERE AccountNumber = :account_number;
```

> Caution:
>
> 要以default context使用XAxa open前default context中应没有任何connection
>
> 不能使用没有CONN_NAME的xa_open string生成多个XA connection生成多个没有
>
> CONN_NAME的connection时最先生成的connection与default context进行匹配

使用named connection

与连接信息同时在xa_open字符串中使用连接名

以下为使用named connection的xa_open字符串的示例

```
DSN=SUNDB;UID=test;PWD=test;CONN_NAME=XA_CONN
```

使用named connection的embedded SQL中应如下描述connection名称

```
EXEC SQL AT XA_CONN
        UPDATE Deposit
        Set InterestRates = :value :value_ind
        WHERE AccountNumber = :account_number;
```

Example Program

• SUNDB sample - XA

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

• Include SUNDB ODBC header

```
#include <sundb.h>


EXEC SQL INCLUDE SQLCA;
#define  SUCCESS  0
#define  FAILURE  -1


#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                            \
        printf("\n");                                            \
        printf(aMsg);                                            \
```

```
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                       \
               SQLSTATE,                                            \
               sqlca.sqlerrm.sqlerrmc );                             \
    }
```

• User-specific definitions

```
#define  BUF_LEN 101
#define SUNDB_SQL_THROW( aLabel )              \
    goto aLabel;
#define SUNDB_SQL_TRY( aExpression )           \
    do                                        \
    {                                         \
        if( !(SQL_SUCCEEDED( aExpression ) ) )  \
        {                                     \
            goto SUNDB_FINISH_LABEL;           \
        }                                     \
    } while( 0 )
#define SUNDB_FINISH                           \
    goto SUNDB_FINISH_LABEL;                   \
    SUNDB_FINISH_LABEL:
```

• Print diagnostic record to console

```
void PrintDiagnosticRecord( SQLSMALLINT aHandleType, SQLHANDLE aHandle )
```

```
{
    SQLCHAR       sSQLState[6];
    SQLINTEGER    sNaiveError;
    SQLSMALLINT   sTextLength;
    SQLCHAR       sMessageText[SQL_MAX_MESSAGE_LENGTH];
    SQLSMALLINT   sRecNumber = 1;
    SQLRETURN     sReturn;
```

• SQLGetDiagRec returns the current values which includes an error, warning.

```
while( 1 )
    {
        sReturn = SQLGetDiagRec( aHandleType,
                                 aHandle,
                                 sRecNumber,
                                 sSQLState,
                                 &sNaiveError,
                                 sMessageText,
                                 100,
                                 &sTextLength );
        if( sReturn == SQL_NO_DATA )
        {
            break;
        }
        SUNDB_SQL_TRY( sReturn );
        printf("\n=============================================\n" );
```

```
        printf("SQL_DIAG_SQLSTATE     : %s\n", sSQLState );
        printf("SQL_DIAG_NATIVE       : %d\n", sNaiveError );
        printf("SQL_DIAG_MESSAGE_TEXT : %s\n", sMessageText );
        printf("============================================\n" );
        sRecNumber++;
    }
    return;
    SUNDB_FINISH;
    printf("SQLGetDiagRec failure.\n" );
    return;
}
```

• Create table

```
int testCreateTable()
{
    EXEC SQL AT XA_CONN
        DROP TABLE IF EXISTS DEPOSIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL AT XA_CONN
        CREATE TABLE DEPOSIT (
            NAME         VARCHAR(30),
            BALANCE      INTEGER,
```

```
            ACCOUNTNUMBER VARCHAR(100),
            ACCOUNTDAY    DATE,
            INTERESTRATES NUMBER(10, 5),
            PHONENUMBER   VARCHAR(30) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL AT XA_CONN COMMIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL AT XA_CONN ROLLBACK;
    return FAILURE;
}
```

• Drop table

```
int testDropTable()
{
    EXEC SQL AT XA_CONN
        DROP TABLE DEPOSIT;
```

```
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL AT XA_CONN COMMIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL AT XA_CONN ROLLBACK;
    return FAILURE;
}
```

• Insert function

```
int testInsert( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char            sName[BUF_LEN];
    int             sNameInd                    = 0;
    int             sBalance                    = 0;
    int             sBalanceInd                 = 0;
    char            sAccountNumber[BUF_LEN];
```

```
    int             sAccountNumberInd          = 0;
    DATE            sAccountDay;
    int             sAccountDayInd             = 0;
    double          sInterestRates             = 0;
    int             sInterestRatesInd          = 0;
    char            sPhoneNumber[BUF_LEN];
    int             sPhoneNumberInd            = 0;
    EXEC SQL END DECLARE SECTION;
    sNameInd             = snprintf( (char*)sName, BUF_LEN, "csii" );
    sBalance             = 30000000;
    sAccountNumberInd     = snprintf( (char*)sAccountNumber, BUF_LEN,
"9999-99-9999" );
    sAccountDay.year      = 2009;
    sAccountDay.month     = 1;
    sAccountDay.day       = 1;
    sAccountDay.hour      = 0;
    sAccountDay.minute    = 0;
    sAccountDay.second    = 0;
    sAccountDay.fraction  = 0;
    sInterestRates       = (double)5.0;
    sPhoneNumberInd       = snprintf( (char*)sPhoneNumber, BUF_LEN, "010-
9999-9999" );
    EXEC SQL AT XA_CONN
        INSERT INTO DEPOSIT
        VALUES ( :sName :sNameInd,
```

```
                :sBalance :sBalanceInd,
                :sAccountNumber :sAccountNumberInd,
                :sAccountDay :sAccountDayInd,
                :sInterestRates :sInterestRatesInd,
                :sPhoneNumber :sPhoneNumberInd );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

• The number of rows affected by INSERT statement

```
printf("\n%d row created.\n\n", sqlca.sqlerrd[2] );
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    return FAILURE;
}
```

• Update function

```
int testUpdate( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char     sCondition[BUF_LEN];
    int      sConditionInd          = 0;
```

```
    double   sValue                = 0;
    int      sValueInd             = 0;
    EXEC SQL END DECLARE SECTION;
    sValue       = (SQLREAL)6.0;
    sConditionInd = snprintf( (char*)sCondition,
                              BUF_LEN,
                              "9999-99-9999" );
    EXEC SQL AT XA_CONN
        UPDATE Deposit
        Set InterestRates = :sValue :sValueInd
        WHERE AccountNumber = :sCondition :sConditionInd;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

• The number of rows affected by UPDATE statement

```
printf("\n%d row updated.\n\n", sqlca.sqlerrd[2] );
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    return FAILURE;
}
```

• Select function

```
int testSelect( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char            sName[BUF_LEN];
    int             sNameInd                    = 0;
    int             sBalance                    = 0;
    int             sBalanceInd                 = 0;
    char            sAccountNumber[BUF_LEN];
    int             sAccountNumberInd           = 0;
    DATE            sAccountDay;
    int             sAccountDayInd              = 0;
    double          sInterestRates              = 0;
    int             sInterestRatesInd           = 0;
    char            sPhoneNumber[BUF_LEN];
    int             sPhoneNumberInd             = 0;
    EXEC SQL END DECLARE SECTION;
    int             sCount                      = 0;
    int             sIsOpen                     = 0;
    EXEC SQL DECLARE CUR1 CURSOR FOR
        SELECT NAME, BALANCE, ACCOUNTNUMBER, ACCOUNTDAY, INTERESTRATES,
PHONENUMBER
        FROM DEPOSIT;
    EXEC SQL AT XA_CONN
        OPEN CUR1;
    if( sqlca.sqlcode != 0 )
```

```
{
    goto fail_exit;
}
sIsOpen = 1;
printf( "========================================\n" );
while( 1 )
{
    EXEC SQL AT XA_CONN
        FETCH CUR1 INTO
        :sName :sNameInd,
        :sBalance :sBalanceInd,
        :sAccountNumber :sAccountNumberInd,
        :sAccountDay :sAccountDayInd,
        :sInterestRates :sInterestRatesInd,
        :sPhoneNumber :sPhoneNumberInd;
    if( sqlca.sqlcode == SQL_NO_DATA )
    {
        break;
    }
    else if( sqlca.sqlcode != 0 )
    {
        goto fail_exit;
    }
    printf( "NAME         : " );
    if( sNameInd == -1 )
```

```
{
    printf( "(null)" );
}
else
{
    printf( "%s", sName );
}
printf( "\n" );
printf( "BALANCE      : " );
if( sBalanceInd == -1 )
{
    printf( "(null)" );
}
else
{
    printf( "%d", sBalance );
}
printf( "\n" );
printf( "ACCOUNTNUMBER : " );
if( sAccountNumberInd == -1 )
{
    printf( "(null)" );
}
else
{
```

```
            printf( "%s", sAccountNumber );
        }
        printf( "\n" );
        printf( "ACCOUNTDAY    : " );
        if( sAccountDayInd == -1 )
        {
            printf( "(null)" );
        }
        else
        {
            printf( "%4d-%02d-%02d", sAccountDay.year, sAccountDay.month,
sAccountDay.day);
        }
        printf( "\n" );
        printf( "INTERESTRATES : " );
        if( sInterestRatesInd == -1 )
        {
            printf( "(null)" );
        }
        else
        {
            printf( "%lf", sInterestRates );
        }
        printf( "\n" );
        printf( "PHONENUMBER   : " );
```

```
        if( sPhoneNumberInd == -1 )
        {
            printf( "(null)" );
        }
        else
        {
            printf( "%s", sPhoneNumber );
        }
        printf( "\n" );
        printf( "----------------------------------------\n" );
        sCount ++;
    }
    printf( "========================================\n" );
    printf( "\n%d rows selected.\n\n", sCount );
    sIsOpen = 0;
    EXEC SQL AT XA_CONN
        CLOSE CUR1;
    if( sqlca.sqlcode != 0 )
    {
        goto fail_exit;
    }
    return SUCCESS;
fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    if( sIsOpen == 1 )
```

```
    {
        EXEC SQL AT XA_CONN
            CLOSE CUR1;
    }
    return FAILURE;
}
```

• Delete function

```
int testDelete( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char     sCondition[BUF_LEN];
    int      sConditionInd          = 0;
    EXEC SQL END DECLARE SECTION;
    sConditionInd = snprintf( (char*)sCondition,
                              BUF_LEN,
                              "9999-99-9999" );
    EXEC SQL AT XA_CONN
        DELETE FROM DEPOSIT WHERE AccountNumber
= :sCondition :sConditionInd;
    if( sqlca.sqlcode != 0 )
    {
        goto fail_exit;
    }
```

• The number of rows affected by DELETE statement

```
printf("\n%d row deleted.\n\n", sqlca.sqlerrd[2] );
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    return FAILURE;
}
```

• Start function

```
int main( int aArgc, char** aArgv )
{
    SQLHENV       sEnv    = NULL;
    SQLINTEGER    sState  = 0;
    xa_switch_t * sXaSwitch;
    XID           sXid;
    sXaSwitch = SQLGetXaSwitch();
```

• If a user calls SQLAllocEnv() which is included in SUNDB ODBC

```
SUNDB_SQL_TRY( SQLAllocHandle( SQL_HANDLE_ENV,
                               NULL,
                               &sEnv ) );
    sState = 1;
```

• SQLSetEnvAttr sets attributes which controls aspects of environments.

```
SUNDB_SQL_TRY( SQLSetEnvAttr( sEnv,
                              SQL_ATTR_ODBC_VERSION,
                              (SQLPOINTER)SQL_OV_ODBC3,
                              0 ) );


if( (sXaSwitch->xa_open_entry)( "DSN=SUNDB;UID=test;PWD=test;CONN_NAME=XA_
CONN", 0, TMNOFLAGS ) != XA_OK )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
    sState = 2;
    sXid.formatID = 0;
    sXid.gtrid_length = 2;
    sXid.bqual_length = 1;
    memcpy( sXid.data, "100", sXid.gtrid_length + sXid.bqual_length );
```

• Create table

```
if( testCreateTable() != SUCCESS )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
    sState = 3;
    if( (sXaSwitch->xa_start_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
```

```
    }
    sState = 4;
```

• Insert row

```
if( testInsert() != SUCCESS )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
```

• Update row

```
if( testUpdate() != SUCCESS )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
```

• Select row

```
if( testSelect() != SUCCESS )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
```

• Delete row

```
if( testDelete() != SUCCESS )
```

```
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
    sState = 3;
    if( (sXaSwitch->xa_end_entry)( &sXid, 0, TMSUCCESS ) != XA_OK )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
    if( (sXaSwitch->xa_prepare_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
    if( (sXaSwitch->xa_commit_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
    sState = 2;
```

• Drop table

```
if( testDropTable() != SUCCESS )
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
    sState = 1;
    if( (sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS ) != XA_OK )
```

```
    {
        SUNDB_SQL_THROW( SUNDB_FINISH_LABEL );
    }
```

• SQLFreeHandleEnv releases resources related to the environment.

```
sState = 0;
    SUNDB_SQL_TRY( SQLFreeHandle( SQL_HANDLE_ENV,
                                  sEnv ) );
    sEnv = NULL;
    return EXIT_SUCCESS;
    SUNDB_FINISH;
    if( sEnv != NULL)
    {
        PrintDiagnosticRecord( SQL_HANDLE_ENV, sEnv );
    }
    switch( sState )
    {
        case 4:
            (void)(sXaSwitch->xa_end_entry)( &sXid, 0, TMSUCCESS );
            (void)(sXaSwitch->xa_prepare_entry)( &sXid, 0, TMNOFLAGS );
            (void)(sXaSwitch->xa_commit_entry)( &sXid, 0, TMNOFLAGS );
        case 3:
            (void)testDropTable();
        case 2:
            (void)(sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS );
```

```
        case 1:
            (void)SQLFreeHandle( SQL_HANDLE_ENV, sEnv );
            sEnv = NULL;
        default:
            break;
    }
    return EXIT_FAILURE;
}
```

4.4 Embedded SQL Reference

本章节只说明可在SUNDB的embedded SQL应用程序中使用的SQL语句在源代码中处理embedded SQL语句时必须遵循以下语法

```
<statement> ::= EXEC SQL <exec sql statement>;
<exec sql statement> ::=
      <embedded SQL statement>
    | <embedded get groupid statement>
    | <embedded specific statement>
    ;

<embedded SQL statement> ::=
    [ AT <db_name> ] [ ATOMIC ] [ FOR <iteration_count> ] <sql statement>
    ;
<embedded get groupid statement> ::=
    [ AT <db_name> ]  <get groupid statement> <sql statement>

<embedded specific statement> ::=
      <autocommit statement>
    | <declare section statement>
    | <include statement>
    | <exception statement>
    | <context statement>
    | <option statement>
```

```
    | <allocate statement>
    | <free statement>
    ;

<autocommit statement> ::= [ AT <db_name> ] AUTOCOMMIT { ON | OFF };
<declare section statement> ::= { BEGIN | END } DECLARE SECTION;
<include statement> ::= INCLUDE { SQLCA | <identifier> };
<exception statement> ::= WHENEVER <exception_condition>
<exception_action>;
<context statement> ::= CONTEXT <context action>;
<context action> ::=
      ALLOCATE :context_name
    | FREE :context_name
    | USE :context_name
    | USE DEFAULT
    ;
<option statement> ::= OPTION ( <option> );
<get groupid statement> :: GET GROUPID INTO :groupid;
<allocate statement> ::= [AT <db_name>] ALLOCATE :cursor_variable;
<free statement> ::= [AT <db_name>] FREE :cursor_variable;
```

EXEC SQL ALLOCATE

功能

Embedded SQL中分配Cursor变量

语句

```
EXEC SQL [ AT <db_name> ] ALLOCATE :hostvar


<db_name> ::=

      <identifier>

   | :hostvar

   ;
```

说明

分配run-time cursor变量的内存如要分配run-time cursor变量需要先在DECLARE SECTION声明SQL_CURSOR类型的变量后对该变量进行分配该语句不仅分配内存还会在client上分配statement所以与context有关联即只能在分配cursor变量的context上才能使用

使用示例

```
{

    EXEC SQL BEGIN DECLARE SECTION;

    SQL_CURSOR empCursor;
```

```
    EXEC SQL END DECLARE SECTION;


    ...
    EXEC SQL AT CONN1 ALLOCATE :empCursor;
    ...
}
```

EXEC SQL AT

功能

指定要应用于embedded SQL的连接名

语句

```
EXEC SQL [ AT <db_name> ] ...


<db_name> ::=
      <identifier>
    | :hostvar
    ;
```

说明

Embedded SQL应用程序进行连接时可指定连接名其名称用于使用特定connection执行embedded SQL的情况

使用示例

```
{
    ...
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd
USING :conn_str;


    EXEC SQL AT :conn_name
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';


    ...
}
```

参考

相关内容参考 **Connection**

EXEC SQL ATOMIC INSERT

功能

执行atomic array insert

语句

```
EXEC SQL ATOMIC <insert_statement>;
```

说明

在embedded SQL应用程序执行atomic array insertAtomic array insert是可用一次命令插入多条row的语句所有的row插入成功则返回成功只要失败一条则所有row的插入均失败

用一次命令插入row因此相比逐条插入性能更加优越

使用示例

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    int    emp_number[20];
    char   emp_name[20][10];
    int    dept_number[20];
    EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值

```
...
```

- 插入emp_numberemp_namedept_number

```
EXEC SQL ATOMIC INSERT INTO emp (empno, ename, deptno)
                          VALUES (:emp_number, :emp_name, :dept_number);
}
```

参考

相关内容参考 **Atomic Insert**

EXEC SQL AUTOCOMMIT

功能

变更autocommit设置

语句

```
EXEC SQL AUTOCOMMIT { ON | OFF };
```

说明

变更autocommit设置

Flag	说明
ON	执行语句后自动提交
OFF	执行commit语句之前不提交

使用示例

```
{
    ...
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd
USING :conn_str;


    EXEC SQL AUTOCOMMIT ON;
```

```
    EXEC SQL AT :conn_name
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';

    ...
}
```

参考

相关内容参考**Auto commit**

EXEC SQL BEGIN DECLARE SECTION

功能

作为预编译器的指示符指定host variable的声明领域

语句

```
EXEC SQL BEGIN DECLARE SECTION;
```

说明

作为指定host variable声明领域的预编译器指示符总是与EXEC SQL END DECLARE SECTION一起使用预编译器遇到此语句时将declare section判断为开始并将之后的变量声明识别为host variable处理

使用示例

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    int    emp_number[20];
    char   emp_name[20][10];
    int    dept_number[20];
    EXEC SQL END DECLARE SECTION;


    ...
```

```
}
```

参考

相关内容参考如下

- **声明Host Variable**
- **EXEC SQL END DECLARE SECTION**

EXEC SQL COMMIT RELEASE

功能

事务结束后断开连接

语句

```
EXEC SQL [ AT <db_name> ] COMMIT [ WORK ] RELEASE;
```

说明

事务结束后断开当前连接

```
EXEC SQL AT :conn_name COMMIT RELEASE;
```

上述语句与以下相同

```
EXEC SQL AT :conn_name COMMIT;

EXEC SQL AT :conn_name DISCONNECT;
```

使用示例

```
{
    ...
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd
USING :conn_str;
```

```
    EXEC SQL AT :conn_name
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';

    EXEC SQL AT :conn_name COMMIT RELEASE;
    ...
}
```

参考

相关内容参考 **RELEASE option**

EXEC SQL CONNECT

功能

连接SUNDB

语句

```
EXEC SQL [ AT <db_name> ] CONNECT <user_name> IDENTIFIED BY <password>
[ AT <db_name> ] [ USING <conn_string> ]


<db_name> ::=
      <identifier>
    | :hostvar
    ;
<user_name> ::=
      <identifier>
    | :hostvar
    ;
<password> ::=
      <identifier>
    | :hostvar
    ;
<conn_string> ::= :hostvar;
```

说明

设置SUNDB连接

使用示例

```
{
    ...
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;


    ...
}
```

参考

相关内容参考 **连接数据库**

EXEC SQL CONTEXT ALLOCATE

功能

分配run-time context内存

语句

```
EXEC SQL CONTEXT ALLOCATE :context;
```

说明

分配run-time context内存分配run-time context时在declare section中声明SQL_CONTEXT类型的变量后对其变量进行分配此语句仅分配内存为了使用需指定USE并执行connect

使用示例

```
{
    ...
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT ctxt;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL CONTEXT ALLOCATE :ctxt;

    EXEC SQL CONTEXT USE :ctxt;
```

```
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;


    EXEC SQL
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';


    EXEC SQL DISCONNECT;
    EXEC SQL CONTEXT FREE :ctxt;


    ...
}
```

参考

相关内容参考**SQL_CONTEXT**

EXEC SQL CONTEXT FREE

功能

释放run-time context内存

语句

```
EXEC SQL CONTEXT FREE :context;
```

说明

释放run-time context内存为了释放run-time context需断开使其不再使用connection否则报错

使用示例

```
{
    ...
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT ctxt;
    EXEC SQL END DECLARE SECTION;


    EXEC SQL CONTEXT ALLOCATE :ctxt;


    EXEC SQL CONTEXT USE :ctxt;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;
```

```
    EXEC SQL
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';

    EXEC SQL DISCONNECT;
    EXEC SQL CONTEXT FREE :ctxt;

    ...
}
```

参考

相关内容参考 **SQL_CONTEXT**

EXEC SQL CONTEXT USE

功能

告知使用run-time context

语句

```
EXEC SQL CONTEXT USE { :context | DEFAULT };
```

说明

作为向预编译器告知使用run-time context的指示符指定当前开始要使用的run-time contextUSE语句中通过SQL_CONTEXT变量可使用户声明并分配的run-time context使用USE DEFAULT时使用应用程序默认拥有的default context

使用示例

```
{
    ...
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT ctxt;
    double      max_sal;
    EXEC SQL END DECLARE SECTION;


    EXEC SQL CONTEXT ALLOCATE :ctxt;
```

```
    EXEC SQL CONTEXT USE :ctxt;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';

    EXEC SQL COMMIT RELEASE;
    EXEC SQL CONTEXT FREE :ctxt;

    EXEC SQL CONTEXT USE DEFAULT;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL
        SELECT MAX(sal)
        INTO   max_sal
        FROM   EMP
        WHERE  JOB = 'SALES';

    EXEC SQL DISCONNECT;
    ...
}
```

参考

相关内容参考**SQL_CONTEXT**

EXEC SQL DISCONNECT

功能

断开SUNDB的连接

语句

```
EXEC SQL [ AT <db_name> ] DISCONNECT [ ALL ]
```

说明

断开与SUNDB的连接可通过AT子句断开特定连接不使用AT子句则断开当前使用中的连接使用DISCONNECT ALL时断开当前应用程序使用中的所有连接

使用示例

```
{
    ...
    EXEC SQL AT :conn_name DISCONNECT;

    ...
}
```

参考

相关内容参考 **断开数据库连接**

EXEC SQL END DECLARE SECTION

功能

作为预编译器指示符指定host variable的声明领域

语句

```
EXEC SQL END DECLARE SECTION;
```

说明

作为指定host variable声明领域的预编译器指示符总是与EXEC SQL BEGIN DECLARE SECTION一起使用预编译器分析declare section的过程中接收到此语句则判断为declare section已结束

使用示例

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    int    emp_number[20];
    char   emp_name[20][10];
    int    dept_number[20];
    EXEC SQL END DECLARE SECTION;

    ...
}
```

参考

相关内容参考如下

- **声明Host Variable**
- **EXEC SQL BEGIN DECLARE SECTION**

EXEC SQL FOR

功能

在array operation指定array数量

语句

```
EXEC SQL FOR { :array_count | integer_constant } <sql statement>;
```

说明

SQL语句的主机变量为array时指定array count的预编译器指示符如有FOR子句则忽略实际host array的array count执行FOR子句指定的array数量

表示array count的常数或变量应为整数型

使用示例

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
int    record_cnt;
EXEC SQL END DECLARE SECTION;
```

- 设置emp_numberemp_namedept_number值

```
...
```

- 插入emp_numberemp_namedept_number

```
record_cnt = 10;
EXEC SQL FOR :record_cnt INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

参考

相关内容参考**使用FOR语句**

EXEC SQL FREE

功能

在embedded SQL中释放cursor变量

语句

```
EXEC SQL [ AT <db_name> ] FREE :hostvar


<db_name> ::=

      <identifier>

    | :hostvar

    ;
```

说明

释放run-time cursor变量的内存没有明确释放的cursor变量会在context断开连接或关闭程序时释放Cursor变量与分配的context有关所以与FREE命令也有关联

使用示例

```
{

    EXEC SQL BEGIN DECLARE SECTION;

    SQL_CURSOR empCursor;

    EXEC SQL END DECLARE SECTION;
```

```
    ...
    EXEC SQL AT CONN1 FREE :empCursor;
    ...
}
```

EXEC SQL GET GROUPID INTO

功能

获取SQL statement的group ID

语句

```
EXEC SQL [ AT <db_name> ] GET GROUPID INTO :groupid { delete_stmt |
insert_stmt | select_stmt | update_stmt };
```

说明

在使用global connection的集群环境中获取SQL statement的group ID主机变量：groupid只能使用signed numeric type可获取group ID的SQL statement仅限于deleteinsertselectupdatetable中应设置shard key

获取group ID的SQL statement在内部上以未执行SQLExecute的SQLPrepare的状态cache

> Caution:
>
> 可能返回无效的group ID值 -1

使用示例

```
{
EXEC SQL BEGIN DECLARE SECTION;
```

```
int  groupid[10];
int  emp_no[10];
char emp_name[10][20];
int  dept_no[10];
EXEC SQL END DECLARE SECTION;
```

- 设置emp_noemp_namedept_no值

```
...
```

- 获取group ID值

```
EXEC SQL GET GROUPID INTO :groupid
        INSERT INTO emp (empno, ename, deptno) VALUES
(:emp_no, :emp_name, :dept_no);
```

- INSERT emp_noemp_namedept_no

```
EXEC SQL INSERT INTO emp (empno, ename, deptno) VALUES
(:emp_no, :emp_name, :dept_no);
}
```

EXEC SQL INCLUDE

功能

包含embedded SQL头文件

语句

```
EXEC SQL INCLUDE <Header file name>;
```

说明

包含embedded SQL头文件预编译器不解析以C语言的#include语句包含的头文件因此头文件中存在declare section等预编译器需了解的内容时也无法识别因此使用预编译器需要了解的embedded SQL语句时需要使用EXEC SQL INCLUDE语句

使用示例

```
EXEC SQL INCLUDE decl.h;
```

参考

相关内容参考**Precompiled Header Files**

EXEC SQL INCLUDE SQLCA

功能

包含*sqlca.h* 头文件

语句

```
EXEC SQL INCLUDE SQLCA;
```

说明

是EXEC SQL INCLUDE语句的特殊形态包含SUNDB提供的sqlca.h头文件Embedded SQL应用程序的run-time exception handling必须需要此头文件

使用示例

```
EXEC SQL INCLUDE SQLCA;
```

参考

相关内容参考 **Runtime Error的检测**

EXEC SQL OPTION

功能

预编译embedded SQL源代码时适用选项

语句

```
EXEC SQL OPTION ( <option_desc> );

<option_desc> ::=

     INCLUDE = <directory path>

   ;
```

说明

指定预编译Embedded SQL源代码的过程中适用的选项当前版本仅支持INCLUDE路径指定此选项

描述了要在EXEC SQL INCLUDE中预编译的头文件所在的目录

使用示例

```
EXEC SQL OPTION ( INCLUDE = include );
```

参考

相关内容参考 **指定Header File路径**

EXEC SQL ROLLBACK RELEASE

功能

回滚事务并断开当前连接

语句

```
EXEC SQL [ AT <db_name> ] ROLLBACK [ WORK ] RELEASE;
```

说明

回滚事务并断开当前连接

```
EXEC SQL AT :conn_name ROLLBACK RELEASE;
```

与以下语句功能相同

```
EXEC SQL AT :conn_name ROLLBACK;

EXEC SQL AT :conn_name DISCONNECT;
```

使用示例

```
{
    ...
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd
USING :conn_str;
```

```
    EXEC SQL AT :conn_name
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';

    EXEC SQL AT :conn_name ROLLBACK RELEASE;
    ...
}
```

参考

相关内容参考**RELEASE option**

EXEC SQL WHENEVER

功能

在embedded SQL应用程序中执行run-time exception handling

语句

```
EXEC SQL WHENEVER <conditions> <actions>;
<conditions> ::=
      SQLERROR
    | SQLWARNING
    | NOT FOUND
    | SQLSTATE <sqlstate class value>[<sqlstate subclass value>]
    ;
<sqlstate_char> ::= [0-9A-Z];
<sqlstate class value> ::= <sqlstate_char> <sqlstate_char>;
<sqlstate subclass value> ::= <sqlstate_char> <sqlstate_char>
<sqlstate_char>;
<actions> ::=
      CONTINUE
    | GOTO <label>
    | STOP
    | DO <c statements>
    ;
```

说明

Embedded SQL应用程序中自动化处理run-time exception handling有4个条件每个条件可以指定一个action此action也可根据所需重新定义详细内容参考**Handling Implicit Error**

使用示例

```
EXEC SQL WHENEVER SQLERROR STOP;

EXEC SQL WHENEVER SQLERROR CONTINUE;

EXEC SQL WHENEVER SQLERROR GOTO exit_label;

EXEC SQL WHENEVER NOT FOUND DO break;

EXEC SQL WHENEVER SQLERROR GOTO close_label;

EXEC SQL WHENEVER SQLERROR DO sql_error();

EXEC SQL WHENEVER SQLWARNING CONTINUE;

EXEC SQL WHENEVER SQLSTATE HY000 DO sql_error();
```

参考

相关内容参考 **Implicit Error Handling**

5.PDO

5.1 PDO概要

SUNDB PDO (PDO_SUNDB)驱动程序为了使PDO能够访问SUNDB数据库而提供PDO接口以PECL package提供

PDO_SUNDB基于SUNDB CLI驱动程序编写

5.2 安装/构成

要求事项

需要在与PHP相同的系统安装SUNDB的客户端安装包并设置$SUNDB_HOME环境变量

安装

使用PECL安装

没有PDO_SUNDB安装包文件时生成安装包文件

```
% pecl pickle

Attempting to process the second package file

Package PDO_SUNDB-3.2.0.tgz done
```

有PDO_SUNDB安装包文件时使用pecl设置安装包

```
% sudo -E pecl install PDO_SUNDB-3.2.0.tgz
```

通过source build安装

6. PDO_SUNDB解压

```
% tar xvzf PDO_SUNDB-3.2.0.tgz

% cd PDO_SUNDB-3.2.0
```

7. 生成build环境

使用phpize命令创建PDO_SUNDB扩展模块的build环境

如果无法找到phpize命令则需安装合适版本的PHP devel安装包

```
% whereis phpize

phpize: /usr/bin/phpize /usr/share/man/man1/phpize.1.gz
```

```
% /usr/bin/phpize

Configuring for:
```

```
PHP Api Version:         20121113

Zend Module Api No:      20121212

Zend Extension Api No:   220121212
```

8. Build

在--with-pdo-sundb选项添加$SUNDB_HOME路径在--with-php-config选项添加php-config路径并执行configure后build

```
% whereis php-config

php-config: /usr/bin/php-config /usr/share/man/man1/php-config.1.gz
```

```
% ./configure --with-php-config=/usr/bin/php-config --with-pdo-

sundb=/home/sundb/sundb_home/
```

```
% make
```

9. 安装

```
% sudo -E make install
```

PHP安装文件修改

使用CentOS 6.0以上版本或Fedora 15以上版本时在 /etc/php.d 目录生成pdo_sundb.ini文件后

如下添加extension属性

```
extension=pdo_sundb.so
```

使用其他操作系统时在php.ini文件设置合适的extension_dir后如下添加extension属性

```
extension=pdo_sundb.so
```

重启网络服务器

重新启动网络服务器

查看PDO_SUNDB安装

使用phpinfo()查看是否激活PDO_SUNDB

```
% php -i | grep PDO

PDO

PDO support => enabled

PDO drivers => sundb, sqlite

PDO Driver for SUNDB => enabled

PDO Driver for SQLite 3.x => enabled
```

5.3 使用

数据源名称（DSN)

PDO_SUNDB的数据源名称（DSN）构成如下

属性	说明
DSN prefix	sundb
DSN	ODBC 数据源名称 (DSN)
HOST	服务器IP地址
PORT	服务器端口号
UID	用户ID
PWD	用户密码

5.4 示例

- Connecting to SUNDB

```
print "\n[Connecting to SUNDB]\n";

$db = new PDO('sundb:HOST=192.168.0.16;PORT=22581', 'test', 'test');
```

```
//$db = new PDO('sundb:DSN=SUNDB;UID=test;PWD=test');
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

- Set test data

```
$result = $db->exec('DROP TABLE IF EXISTS t1');
$result = $db->exec('CREATE TABLE t1 ( id INTEGER GENERATED BY DEFAULT AS
IDENTITY, name VARCHAR(32) )');
$result = $db->exec("INSERT INTO t1(name) VALUES ('Carol')");
$result = $db->exec("INSERT INTO t1(name) VALUES ('Ted')");
$result = $db->exec("INSERT INTO t1(name) VALUES (null)");
$result = $db->exec("INSERT INTO t1(name) VALUES ('William')");
$result = $db->exec("INSERT INTO t1(name) VALUES ('Chelsea')");
$result = $db->exec("INSERT INTO t1(name) VALUES ('Colin')");

$result = $db->exec('DROP TABLE IF EXISTS t2');
$result = $db->exec('CREATE TABLE t2 ( id INTEGER, name VARCHAR(32) )');
$result = $db->exec("INSERT INTO t2 VALUES (1, 'John')");
$result = $db->exec("INSERT INTO t2 VALUES (1, 'John')");
$result = $db->exec("INSERT INTO t2 VALUES (1, 'John')");
$result = $db->exec("INSERT INTO t2 VALUES (2, 'Smith')");
$result = $db->exec("INSERT INTO t2 VALUES (2, 'Smith')");

$result = $db->exec('DROP TABLE IF EXISTS images');
$result = $db->exec('CREATE TABLE images ( id INTEGER, contenttype
VARCHAR(256), imagedata LONG VARBINARY )');
```

```
$result = $db->exec('DROP PROCEDURE IF EXISTS sp_out_string');
$result = $db->exec("CREATE OR REPLACE PROCEDURE sp_out_string( v1 OUT
VARCHAR(32) ) ".
                    "IS ".
                    "BEGIN ".
                    "  v1 := 'WORLD'; ".
                    "END; ");

$result = $db->exec('DROP PROCEDURE IF EXISTS sp_inout_string');
$result = $db->exec("CREATE OR REPLACE PROCEDURE sp_inout_string( v1 INOUT
VARCHAR(32) ) ".
                    "IS ".
                    "BEGIN ".
                    "  v1 := V1 || ' WORLD'; ".
                    "END; ");
```

- Quoting a normal string

```
print "\n[Quoting a normal string]\n";
$string = 'Nice';

print "Unquoted string: $string\n";
print "Quoted string  : " . $db->quote($string) . "\n";
```

- Quoting a dangerous string

```
print "\n[Quoting a dangerous string]\n";

$string = 'Naughty \' string';


print "Unquoted string: $string\n";

print "Quoted string  :" . $db->quote($string) . "\n";
```

- Quoting a complex string

```
print "\n[Quoting a complex string]\n";

$string = "Co'mpl''ex \"st'\"ring";


print "Unquoted string: $string\n";

print "Quoted string  : " . $db->quote($string) . "\n";
```

- Handling an error

```
print "\n[Error Handling]\n";

try

{

    //connect as appropriate as above

    $db->query('invalid query'); //invalid query!

}

catch(PDOException $ex)

{

    print "ERROR:".$ex->getMessage()."\n";

}
```

- Retrieving column metadata

```
print "\n[Retrieving column metadata]\n";
$stmt = $db->query('SELECT * FROM t1');
foreach(range(0, $stmt->columnCount() - 1) as $column_index)
{
    $meta = $stmt->getColumnMeta($column_index);
    var_dump($meta);
}
```

- Running simple select statements

```
print "\n[Running Simple Select Statements]\n";

print "\n#1\n";
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row['ID'].' '.$row['NAME']."\n";
}

print "\n#2\n";
$stmt = $db->query('SELECT * FROM t1');

while($row = $stmt->fetch(PDO::FETCH_ASSOC))
{
    print $row['ID'].' '.$row['NAME']."\n";
```

```
}
```

- Fetching rows using different fetch styles

```
print "\n[Fetching rows using different fetch styles]\n";
$stmt = $db->prepare("SELECT id, name FROM t1");
$stmt->execute();

print("PDO::FETCH_ASSOC: ");
print("Return next row as an array indexed by column name\n");
$result = $stmt->fetch(PDO::FETCH_ASSOC);
print_r($result);
print("\n");

print("PDO::FETCH_BOTH: ");
print("Return next row as an array indexed by both column name and
number\n");
$result = $stmt->fetch(PDO::FETCH_BOTH);
print_r($result);
print("\n");

print("PDO::FETCH_LAZY: ");
print("Return next row as an anonymous object with column names as
properties\n");
$result = $stmt->fetch(PDO::FETCH_LAZY);
print_r($result);
```

```
print("\n");


print("PDO::FETCH_OBJ: ");
print("Return next row as an anonymous object with column names as
properties\n");
$result = $stmt->fetch(PDO::FETCH_OBJ);
print $result->NAME;
print("\n");
```

- Fetching rows with a scrollable cursor

```
print "\n[Fetching rows using different fetch styles]\n";
print "\n[Fetching rows with a scrollable cursor]\n";
$sql = 'SELECT id, name FROM t1 ORDER BY id';


print "Reading forwards:\n";
try
{
    $stmt = $db->prepare($sql, array(PDO::ATTR_CURSOR =>
PDO::CURSOR_SCROLL));
    $stmt->execute();
    while( $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_NEXT) )
    {
        $data = $row[0] . "\t" . $row[1] . "\n";
        print $data;
    }
```

```
    $stmt = null;
}
catch (PDOException $e)
{
    print $e->getMessage();
}

print "Reading backwards:\n";
try
{
    $stmt = $db->prepare($sql, array(PDO::ATTR_CURSOR =>
PDO::CURSOR_SCROLL));
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_LAST);
    do
    {
        $data = $row[0] . "\t" . $row[1] . "\n";
        print $data;
    } while( $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_PRIOR) );

    $stmt = null;
}
catch (PDOException $e)
{
    print $e->getMessage();
```

```
}
```

- Constructing an order

```
print "\n[Construction order]\n";
class Person
{
    private $NAME;

    public function __construct()
    {
        $this->tell();
    }

    public function tell()
    {
        if (isset($this->NAME))
        {
            print "I am {$this->NAME}.\n";
        }
        else
        {
            print "I don't have a name yet.\n";
        }
    }
}
```

```
$stmt = $db->query("SELECT name FROM t1");
$stmt->setFetchMode(PDO::FETCH_CLASS, 'Person');
$person = $stmt->fetch();
$person->tell();
$stmt->setFetchMode(PDO::FETCH_CLASS|PDO::FETCH_PROPS_LATE, 'Person');
$person = $stmt->fetch();
$person->tell();
```

- Running simple INSERT, UPDATE, or DELETE statements

```
print "\n[Running Simple INSERT, UPDATE, or DELETE statements]\n";
$affected_rows = $db->exec("INSERT INTO t1(name) VALUES ('John'),
('Marry')");
print $affected_rows." were affected\n";
```

- Getting the last insert id

```
print "\n[Getting the Last Insert Id]\n";
$affected_rows = $db->exec("INSERT INTO t1(name) VALUES ('Smith')");
$insertId = $db->lastInsertId();
print "last insert id : ".$insertId."\n";
```

- Running statements with parameters

```
print "\n[Running Statements With Parameters]\n";


print "\n#1 array\n";
```

```
$id = 1;
$name = 'John';

$stmt = $db->prepare("SELECT * FROM t2 WHERE id=? AND name=?");
$stmt->execute(array($id, $name));
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);

var_dump($rows);

print "\n#2 bindValue\n";
$id = 2;
$name = 'Smith';

$stmt = $db->prepare("SELECT * FROM t2 WHERE id=? AND name=?");
$stmt->bindValue(1, $id, PDO::PARAM_INT);
$stmt->bindValue(2, $name, PDO::PARAM_STR);
$stmt->execute();
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);

var_dump($rows);
```

- Named placeholders

```
print "\n[Named Placeholders]\n";

print "\n#1 array\n";
```

```
$id = 1;
$name = 'John';

$stmt = $db->prepare("SELECT * FROM t2 WHERE id=:id AND name=:name");
$stmt->execute(array(':name' => $name, ':id' => $id));
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);

var_dump($rows);

print "\n#2 bindValue\n";
$id = 2;
$name = 'Smith';

$stmt = $db->prepare("SELECT * FROM t2 WHERE id=:id AND name=:name");
$stmt->bindValue(':id', $id, PDO::PARAM_INT);
$stmt->bindValue(':name', $name, PDO::PARAM_STR);
$stmt->execute();
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);

var_dump($rows);
```

- Executing prepared statements in a loop

```
print "\n[Executing prepared statements in a loop]\n";
$values = array('bob', 'alice', 'lisa', 'john');
$name = '';
```

```
$stmt = $db->prepare("INSERT INTO t1(name) VALUES(:name)");

$stmt->bindParam(':name', $name, PDO::PARAM_STR);

foreach($values as $name)

{

   $stmt->execute();

}


foreach($db->query('SELECT * FROM t1') as $row)

{

    print $row[0].' '.$row[1]."\n";

}
```

- Calling a stored procedure with an output parameter

```
print "\n[Calling a stored procedure with an output parameter]\n";

$stmt = $db->prepare("CALL sp_out_string(?)");

$stmt->bindParam(1, $value, PDO::PARAM_STR, 32);

$stmt->execute();


print "procedure returned $value\n";
```

- Calling a stored procedure with an input/output parameter

```
print "\n[Calling a stored procedure with an input/output parameter]\n";

$stmt = $db->prepare("CALL sp_inout_string(?)");

$value = 'HELLO';
```

```
$stmt->bindParam(1, $value, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT, 32);
$stmt->execute();


print "procedure returned $value\n";
```

- Transactions

```
print "\n[Transactions]\n";


print "\n#1 commit\n";
try
{
    $db->beginTransaction();


    $db->exec("INSERT INTO t1(name) VALUES('kim')");


    $name = 'lee';
    $stmt = $db->prepare("INSERT INTO t1(name) VALUES(?)");
    $stmt->execute(array($name));


    $id = 3;
    $name = 'park';
    $stmt = $db->prepare("INSERT INTO t2 VALUES(?, ?)");
    $stmt->execute(array($id, $name));


    $db->commit();
```

```
}
catch(PDOException $ex)
{
    //Something went wrong rollback!
    $db->rollBack();
    print $ex->getMessage()."\n";
}

print "\nT1\n";
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row[0].' '.$row[1]."\n";
}

print "\nT2\n";
foreach($db->query('SELECT * FROM t2') as $row)
{
    print $row[0].' '.$row[1]."\n";
}

print "\n#2 rollback\n";
try
{
    $db->beginTransaction();
```

```
    $db->exec("INSERT INTO t1(name) VALUES('KIM')");


    $name = 'LEE';
    $stmt = $db->prepare("INSERT INTO t1(name) VALUES(?)");
    $stmt->execute(array($name));


    $id = 3;
    $name = 'PARK';
    $stmt = $db->prepare("INSERT INTO invalid VALUES(?, ?)");
    $stmt->execute(array($id, $name));


    $db->commit();
}
catch(PDOException $ex)
{
    //Something went wrong rollback!
    $db->rollBack();
    print $ex->getMessage()."\n";
}


print "\nT1\n";
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
```

```
print "\nT2\n";
foreach($db->query('SELECT * FROM t2') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
```

- Inserting an image into a database

```
print "\n[Inserting an image into a database]\n";
try
{
    $stmt = $db->prepare("INSERT INTO images (id, contenttype, imagedata)
VALUES (?, ?, ?)");

    $id = 1;
    $type = "image/png";
    $fp = fopen("logo.png", "rb");

    $stmt->bindParam(1, $id, PDO::PARAM_INT);
    $stmt->bindParam(2, $type, PDO::PARAM_STR, 256);
    $stmt->bindParam(3, $fp, PDO::PARAM_LOB);

    $db->beginTransaction();
    $stmt->execute();
    $db->commit();
```

```
}
catch(PDOException $ex)
{
    $db->rollBack();
    print $ex->getMessage()."\n";
}

foreach($db->query('SELECT id, contenttype, lengthb(imagedata) FROM
images') as $row)
{
    print $row[0].' '.$row[1].' '.$row[2]."\n";
}
```

- Displaying an image from a database

```
print "\n[Displaying an image from a database]\n";
try
{
    $stmt = $db->prepare("SELECT contenttype, imagedata FROM images WHERE
id = ?");

    $id = 1;
    $stmt->execute(array($id));
    $stmt->bindColumn(1, $type, PDO::PARAM_STR, 256);
    $stmt->bindColumn(2, $lob, PDO::PARAM_LOB);
    $stmt->fetch(PDO::FETCH_BOUND);
```

```
    print 'Content-Type: '.$type."\n";
    print 'size: '.file_put_contents($id.".png", $lob)."\n";
}
catch(PDOException $ex)
{
    print $ex->getMessage()."\n";
}
```

6.PyDBC

6.1 SUNDB PyDBC

概要

PyDBC可使用遵守**Python Database API Specification v2.0(PEP 249)**的API编程访问SUNDB数据库的Python

PyDBC需要Python标准库连接并操作SUNDB数据库的内部运算调用ODBC API因此需要ODBC库

PyDBC默认使用ODBC库$SUNDB_HOME/lib/中的gdlcs其用户可修改变更setup.py

PyDBC的内部运算使用ODBC驱动程序因此与**ODBC构成要素概要**相同有应用程序链接到驱动程序管理器的体系结构和应用程序链接到SUNDB ODBC驱动程序库的体系结构

版本体系

如下执行pysundb.so文件可查看 SUNDB PyDBC版本信息

```
shell>python
>>> import pysundb
>>> print pysundb.version
3.2.0
```

目前SUNDB PyDBC驱动程序版本是对应SUNDB版本的3.2.0其驱动程序遵循标准Python database API 2.0PyDBC驱动程序支持的Python版本为2.73.43.53.6PyDBC驱动程序库的安装需符合各个Python版本

安装

安装PyDBC需要创建sourcePyDBC链接位于SUNDB_HOME/lib的gdlcs库并包含位于SUNDB_HOME/include的sundb.h头部文件因此需要在合适的位置设置环境变量SUNDB_HOME

安装PyDBCPython和SUNDB库的bit需相同因此SUNDB搭建为32 bit时Python也应使用32 bit安装PyDBC

在Linux安装

Linux需要gcc编译如下build

```
shell> sudo python setup.py install
```

> Note:
>
> 不支持HP-UX与AIX平台

在Windows安装

在Window如下build

```
shell> python setup.py install
```

编译PyDBC需要使用符合python版本的Microsoft Visual C++编译器

详细内容参考**https://wiki.python.org/moin/WindowsCompilers**

- 创建Python 2.4或2.5版本需要Visual Studio 2003.NET 编译器该编译器只有付费版本
- 创建Python 2.62.73.03.2版本需要Visual C++ 2008 编译器该编译器的免费版本为Visual C++ 2008 Express
- 创建Python 3.33.4版本需要Visual C++ 2010编译器该编译器的免费版本为Visual C++ 2010 Express
- 创建Python 3.53.6版本需要 Visual C++ 2014 或 VC 2017 编译器
- 创建Python 3.7版本需要Visual C++ 2017编译器

使用示例

获取connection class

PyDBC在内部调用ODBC库因此为了获取connection需要搭建**数据源构成**

- 为了获取connection如下编写

```
import pysundb

cnxn = pysundb.connect( 'DSN=SUNDB;UID=test;PWD=test' )
```

通过调用PyDBC的模块pysundb的内置函数connect获取connection

> Note:

> 为了使用DSN需要提前在词典编写**数据源构成**

ODBC库在DSN未指定CHARSET时将其设置为Console Character SetPyDBC在内部默认encodingODBC使用的character set SUNDB服务器使用的character set和ODBC库使用的character set不同时会出现数据转换此时性能会低下例如在Windows默认使用CP949 character set未设置任何时PyDBC在内部使用CP949(UHC)进行encodingODBC库也以UHC处理character set

变更client的character set的方法为如下

- 变更**31.2 数据源构成**的CHARSET参数
- 在连接字符串添加CHARSET参数
- 在pysundb模块的connect method使用attrs_before 关键字

上述三种饭发均变更ODBC的连接参数SQL_ATTR_CHARACTER_SET并设置PyDBC库的encoding

使用cursor和row class

可如下使用cursor和row class

```
cursor = cnxn.cursor()
cursor.execute( "SELECT NAME, ADDRESS FROM EMP" )


rows = cursor.fetchall()


for row in rows:
    print row.A, row.B
```

```
cursor.close()

cnxn.close()
```

6.2 API Reference

pysundb 模块

pysundb object遵循 **Python Database API Specification v2.0**

详细内容参考 **Python DB API 모듈**（Python DB API模块）

属性

- version
  - pysundb模块的版本遵从SUNDB数据库的版本版本为major.minor.patch形式的字符串
- apilevel
  - 指DB API level 2.0值为“2.0”字符串常数
- lowercase
  - 控制结果值的row对象中column名称的小写与否默认值为false有利于数据库column的大小写不一致的情况
- threadsafety
  - 常数为1thread共享模块但不共享连接
- paramstyle
  - 表示参数其值为表示问号的字符串常数“qmark”

connect

与数据库创建新的连接

```
connect( *connectionstring, **kwargs )
```

输入ODBC连接字符串与关键字关键字如下

关键字	说明	默认值
attrs_before	指定Connection之前需设置的属性值为dictionary类型	-
autocommit	设置是否auto commitFalse时需调用connection.commit才可反映到数据库	False
readonly	True时, connection设置为readonly	False
timeout	设置Connection的timeout设置为SQL_ATTR_LOGIN_TIMEOUT	-

- attrs_before
  - 指定连接前设置的选项该选项使用**SQLSetConnectAttr**进行设置属性与值为词典类型设置相关详细内容参考**ODBC属性**

```
cnxn =  pysundb.connect( "DSN=SUNDB",
attr_before={ pysundb.SQL_ATTR_MAX_ROWS : 1000 })
```

Date

```
>>> print pysundb.Date(1984,11,23),  type(pysundb.Date(1984,11,23))
1984-11-23 <type 'datetime.date'>
```

生成对应指定值的date对象

Time

```
>>> print pysundb.Time(11,23,23), type(pysundb.Time(11,23,23))
11:23:23 <type 'datetime.time'>
```

生成对应指定值的time对象

Timestamp

```
>>> print pysundb.Timestamp(1984,11,23,11,23,23),
type(pysundb.Timestamp(1984,11,23,11,23,23))
1984-11-23 11:23:23 <type 'datetime.datetime'>
```

生成对应指定值的datetime对象

DATETIME

```
>>> print pysundb.DATETIME(1984,11,23,11,23,23),
type(pysundb.DATETIME(1984,11,23,11,23,23))
1984-11-23 11:23:23 <type 'datetime.datetime'>
```

生成对应指定值的datetime对象与 **Timestamp** 相同

Binary

```
>>> print pysundb.Binary('binary'), type(pysundb.Binary('binary'))
binary <type 'bytearray'>
```

生成对应指定值的bytearray对象与**BINARY** 相同

BINARY

```
>>> print pysundb.BINARY('binary'), type(pysundb.BINARY('binary'))
binary <type 'bytearray'>
```

生成对应指定值的bytearray对象

STRING

```
>>> print pysundb.STRING('str'), type(pysundb.STRING('str'))
str <type 'str'>
```

生成对应指定值的str对象

NUMBER

```
>>> print pysundb.NUMBER(100.001), type(pysundb.NUMBER(100.001))
100.001 <type 'float'>
```

生成对应指定值的float对象

ROWID

```
>>> print pysundb.ROWID('AA'), type(pysundb.ROWID('AA'))
AA <type 'str'>
```

用于描述数据库的row ID column返回str对象

TimeFromTicks

```
>>> print pysundb.TimeFromTicks( 10 )

09:00:10
```

返回设置为参数值的datetime.time对象

DateFromTicks

```
>>> print pysundb.DateFromTicks( 360000 )

1970-01-05
```

返回设置为参数值的datetime.date对象

TimestampFromTicks

```
>>> print pysundb.DateFromTicks( 360000 )

1970-01-05
```

返回设置为参数值的datetime.timestamp对象

setDecimalSeparator

设置从数据库获取的NUMERIC类型的小数点区分字符默认值使用 .

getDecimalSeparator

获取已设置的NUMERIC类型的小数点区分字符

Connection

管理与数据库的连接的对象生成为pysundb模块的connect()函数

属性

- autocommit
  - 可设置Connection的autocommit模式
- searchescape
  - 获取ODBC的Escape字符pysundb使用 '/'
- timeout
  - 使用SQLSetConnectAttr函数设置SQL_ATTR_QUERY_TIMEOUT

函数

- cursor()
  - 返回新的游标对象
- commit()
  - 提交执行过的SQL语句
- rollback()
  - 回滚执行过的SQL语句

- close()
  - 关闭连接autocommit为false时未commit的SQL语句被回滚
- getinfo(info)
  - 使用ODBC的SQLGetInfo函数可获取连接相关属性详细内容参考**SQLGetInfo**

```
dns_name = cnxn.getinfo( pysundb.SQL_DATA_SOURCE_NAME )
```

- execute(sql, [*params])
  - 生成新的游标对象并执行该对象的execute函数后返回游标对象

```
cursor = cnxn.execute( "SELECT COUNT(*) FROM EMP" )
```

详细内容参考Cursor.execute() 函数Python API中没有该函数但为了便利性而提供每次调用该函数时分配游标对象因此不建议在执行一个以上的SQL语句时使用

- set_attr(attr_id, value)
  - 可通过执行**SQLSetConnectAttr**函数设置连接属性
  - 以下为使用set_attr函数调整数据库的事务隔离阶段的示例

```
connection.set_attr( pysundb.SQL_ATTR_TXN_ISOLATION,
pysundb.SQL_TXN_SERIALIZABLE )
```

Cursor

通常游标对象指用于管理fetch操作的数据库游标数据库游标映射于ODBC statement handle(HSTMT)相同的connection生成的游标对象不会互相分离即一个游标在数据库执行的所

有变更事项也会适用于其他游标

> Note:
>
> 游标不管理数据库事务connection提交或回滚事务

属性

Description

只读专用属性包含以元组（tuple）类型最后执行的SELECT语句返回的各个column内容各元组包含以下内容

1. Column name (或 alias)
2. Type code
3. Display size
4. Internal size
5. Precision
6. Scale
7. Nullable

未调用SELECT语句时description为none

rowcount

最后执行的SQL语句更新的row数

arraysize

可使用**fetchmany([size = cursor.arraysize])**函数一次获取的row数默认值为1

connection

为只读属性指该游标对象生成的connection对象

fast_executemany

设置为True时执行**executemany(sql, [*params])**函数时以数组构成参数后通过一次execute处理

设置为False时每个参数单独执行execute

函数

execute(sql, [*params])

通过SQLPrepare与SQLExcute函数执行SQL语句并返回调用此函数的游标

可如下使用选项参数

```
cursor.execute( "SELECT A FROM TEST WHERE B=? AND C=?", x, y )
```

```
cursor.execute( "SELECT A FROM TEST WHERE B=? AND C=?", (x, y) )
```

executemany(sql, [*params])

执行各参数的SQL语句并返回none参数params必须为sequence的sequence类型或sequence generator

```
params = [ ( 1, 'A' ), ( 2, 'B' ) ]
cursor.executemany("INSERT INTO TEST( C1, C2 ) VALUES ( ?, ? )", params)
```

上述示例中执行两次SQL语句即各对执行(1, 'A')与(2, 'B')执行根据游标对象的fast_executemany设置为true或falseexecutemany操作会有所不同

上述示例与以下示例相同

```
params = [ ( 1, 'A' ), ( 2, 'B' ) ]
for p in params:
    cursor.execute( "INSERT INTO TEST( C1, C2 ) VALUES ( ?, ? )", p )
```

fast_executemany设置为true时executemany要通过执行一次execute处理操作为此参数params的item中位于相同索引位置的数据应为相同的数据类型

```
params = [ ( 1, 'A' ), ( '2', 'B' ) ]
cursor.executemany("INSERT INTO TEST( C1, C2 ) VALUES ( ?, ? )", params)
```

上述示例中参数params的两个item中第一个item的数据类型不同如此item之间的位于相同索引位置的数据类型不同时executemany将分别处理SQL语句而非一次性处理

Connection对象的autocommit为true时分割处理SQL语句并分别提交按顺序处理记录的过程中发生错误时只有部分记录commit到数据库部分则未能commit因此使用executemany()时为了查看

所有记录是否已全部commit到数据库建议将autocommit设置为false后执行

fetchone()

返回查询的下一个row没有下一个数据时为none

fetchall()

返回查询中剩余的所有row将把所有row读取至内存因此使用时需谨慎

fetchmany([size = cursor.arraysize])

返回与size或cursor.arraysize相同数量的剩余row下一个数据返回空sequence数据

cursor.arraysize的默认值为1

commit()

提交SQL语句是生成游标对象的connection对象执行的函数因此适用于相同的connection对象生成的所有游标与Connection对象的commit相同

rollback()

回滚SQL语句是生成游标对象的connection对象执行的函数因此适用于相同的connection对象生成的所有游标与Connection对象的rollback相同

skip(count)

通过SQLFetchScroll和SQL_FETCH_NEXT传递与count中设置的次数相同次数的记录

nextset()

SUNDB ODBC不支持SQLMoreResults因此返回false

close()

关闭游标对象

setinputsizes(size_list)

选择性函数接收sequence类型作为参数设置SQLBindParameter的INPUT参数大小

setoutputsize(size)

选择性函数用途与DB API不同分配OUTPUT参数的缓冲区大小

callproc(procname [, params])

调用属于procname的存储procedure参数应为sequence类型并包含输出参数但输入时位于输出参数的数据毫无意义callproc函数变更输入参数数据的INOUTOUT数据并返回为sequence类型

```
create_proc = """CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 OUT
CHAR(10) )
IS
  V1 CHAR(10);
BEGIN
  SELECT T1.I1
    INTO V1
```

```
    FROM T1
    WHERE T1.I1 >= A1 AND T1.I1 <= A1;
  A2 := V1;
END;\
"""

cursor.execute( create_proc )

result = cursor.callproc( 'PROC1', ( 1, 0 ) )
```

callfunc(funcname [, params])

调用属于funcname的函数callfunc()返回函数的数据

```
create_func = """
CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
  RETURN INTEGER
  IS
    V1 INTEGER;
  BEGIN

    SELECT COUNT(*)
      INTO V1
      FROM T1
      WHERE T1.I1 >= A1 AND T1.I1 <= A2;
```

```
    RETURN V1;
  END;\
"""


cursor.execute( create_func )
cursor.commit()


result = cussr.callfunc( 'FUNC1', ( 1,  4) )
```

tables(table=None, catalog=None, schema=None, tableType=None)

返回满足指定条件的数据库的表信息字符'_'与'%'解析为通配符（wildcard）各row拥有如下column信息详细内容参考**SQLTables**

1. table_cat：目录名
2. table_schem：schema名
3. table_name：表名
4. table_type: 可以是 TABLE''VIEW''SYSTEM TABLE''GLOBAL TEMPORARY''LOCAL TEMPORARY''IMMUTABLE TABLE''ALIAS''SYNONYM' 或特定类型的名称
5. remarks: 表明细

```
print cursor.tables( table= 'TEST' ).fetchone()


#print table name
for row in cursor.tables():
```

```
print row.table_name
```

> Note:
>
> 参数为空时返回用户有权限的所有表的信息

columns(table=None, catalog=None, schema=None, tableType=None)

通过**SQLColumns**函数获取指定表的column信息各row包含如下column信息

1. table_cat
2. table_schem
3. table_name
4. column_name
5. data_type
6. type_name
7. column_size
8. buffer_length
9. decimal_digits
10. num_prec_radix
11. nullable
12. remarks
13. column_def
14. sql_data_type
15. sql_datetime_sub

16. char_octet_length
17. ordinal_position
18. is_nullable: SQL_NULLABLE, SQL_NO_NULLS或SQL_NULLS_UNKNOWN.

```
#print column name of table TEST
for r in cursor.columns( table = 'TEST' ):
    print r.column_name
```

statistics(table, catalog=None, schema=None, unique=False, quick=True)

通过**SQLStatistics**函数获取与指定表相关的信息

unique为true时仅返回unique索引false时返回所有索引

quick为true时仅在可立即适用的情况返回CARDINALYTIY与PAGES否则该列返回NULL

1. table_cat
2. table_schem
3. table_name
4. non_unique
5. index_qualifier
6. index_name
7. type
8. ordinal_position
9. column_name
10. asc_or_desc
11. cardinality

12. pages
13. filter_condition

> Note:
>
> 不允许wildcard字符

rowIdColumns(table, catalog=None, schema=None, nullable=True)

用SQL_BEST_ROWID执行**SQLSpecialColumns**返回固有识别row的column的结果集各row拥有如下column信息

1. scope: SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION, 或 SQL_SCOPE_SESSION
2. column_name
3. data_type: ODBC的SQL类型常数
4. type_name
5. column_size
6. buffer_length
7. decimal_digits
8. pseudo_column: SQL_PC_UNKNOWN, SQL_PC_NOT_PSEUDO 或 SQL_PC_PSEUDO

rowVerColumns(table, catalog=None, schema=None, nullable=True)

通过SQL_ROWVER执行**SQLSpecialColumns**返回更新row时自动更新的column的结果集各row

包含如下column信息

1. scope: SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION, 或 SQL_SCOPE_SESSION
2. column_name
3. data_type: ODBC的SQL类型常数
4. type_name
5. column_size
6. buffer_length
7. decimal_digits
8. pseudo_column: SQL_PC_UNKNOWN, SQL_PC_NOT_PSEUDO 或 SQL_PC_PSEUDO

primaryKeys(table, catalog=None, schema=None)

执行**SQLPrimaryKeys**函数返回构成表的主要key的column结果集各row包含如下column信息

1. table_cat
2. table_schem
3. table_name
4. column_name
5. key_seq
6. pk_name

foreignKeys(table=None, catalog=None, schema=None, foreignTable=None, foreignCatalog=None, foreignSchema=None)

执行**SQLForeignKeys**函数生成指定表或参照指定表的主键的其他表的外键的column名称的结果集各row包含如下column信息

1. pktable_cat
2. pktable_schem
3. pktable_name
4. pkcolumn_name
5. fktable_cat
6. fktable_schem
7. fktable_name
8. fkcolumn_name
9. key_seq
10. update_rule
11. delete_rule
12. fk_name
13. pk_name
14. deferrability

procedures(procedure=None, catalog=None, schema=None)

执行**SQLProcedures**生成procedure信息的结果集各row包含如下column信息

1. procedure_cat
2. procedure_schem
3. procedure_name
4. num_input_params
5. num_output_params
6. num_result_sets
7. remarks
8. procedure_type

getTypeInfo(sqlType=None)

执行**SQLGetTypeInfo**函数生成指定的数据类型或SUNDB ODBC支持的所有数据类型相关信息的结果集各row包含如下column

1. type_name
2. data_type
3. column_size
4. literal_prefix
5. literal_suffix
6. create_params
7. nullable
8. case_sensitive
9. searchable
10. unsigned_attribute
11. fixed_prec_scale
12. auto_unique_value

13. local_type_name
14. minimum_scale
15. maximum_scale
16. sql_data_type
17. sql_datetime_sub
18. num_prec_radix
19. interval_precision

Row

Row对象以游标对象的fetch函数返回像DB API指定的元组类型一样处理

```
row = cursor.fetchone()
for column in row:
    print column
```

pysundb添加了如下功能

- 可使用column名称访问数据
- 游标对象关闭后也可以通过row访问cursor.description值
- 可变更row的值

使用column名称访问row不仅方便且可读性高但column名称中包含Python保留字或空白时需通过row.__getattribute__()访问

```
cursor.execute( "select c1 from test")
```

```
print cursor.description

row = cursor.fetchone()

print row.C1
```

```
(('C1', <type 'str'>, 10, 10, 10, 0, True),)

test
```

> Note:
>
> SUNDB数据库的识别符默认为大写但也有指定为小写的情况因此使用column名称访问row时需注意大小写

属性

- cursor_description

生成该row的游标对象的属性description的副本详细内容参考 **Cursor.description**

6.3 Exception

Python异常是在SUNDB ODBC感应到错误时由pysundb触发的如下异常class与 **Python DB API** 相同

- Error
  - DatabaseError
    - DataError
    - OperationalError
    - IntegrityError
    - InternalError
    - ProgrammingError
    - NotSupportedError

发生错误时通常情况下异常类型以数据库提供的SQLSTATE值为基础进行处理

SQLSTATE	Exception
0A000	NotSupportedError
01002	OperationalError
08001	OperationalError
08003	OperationalError
08004	OperationalError
08007	OperationalError

SQLSTATE	Exception
08S01	OperationalError
28000	InterfaceError
40002	IntegrityError
22***	DataError
23***	IntegrityError
24***	ProgrammingError
25***	ProgrammingError
42***	ProgrammingError

6.4 Data Type

向SUNDB传送Python参数

向SUNDB ODBC传送python参数时如下转换数据

Python datatype	Description	ODBC datatype
None	-	SQL_VARCHAR
str	UTF-8	SQL_VARCHAR or SQL_LONGVARCHAR
bytes, bytearray	binary	SQL_VARBINARY or SQL_LONGVARBINARY
bool	bit	SQL_BIT
datetime.date	date	SQL_TYPE_DATE
datetime.time	time	SQL_TYPE_TIME
datetime.datetime	timestamp	SQL_TYPE_TIMESTAMP
int	integer	SQL_BIGINT
float	floating point	SQL_DOUBLE
decimal	numeric	SQL_NUMERIC

Table 6-1 Python 3

Python datatype	Description	ODBC datatype
None	-	SQL_VARCHAR
str	UTF-8	SQL_VARCHAR or SQL_LONGVARCHAR
bytearray	binary	SQL_VARBINARY or SQL_LONGVARBINARY
buffer	binary	SQL_VARBINARY or SQL_LONGVARBINARY
bool	bit	SQL_BIT
datetime.date	date	SQL_TYPE_DATE
datetime.time	time	SQL_TYPE_TIME
datetime.datetime	timestamp	SQL_TYPE_TIMESTAMP
int	integer	32 bit: SQL_INTEGER, 64 bit: SQL_BIGINT
long	bigint	SQL_BIGINT
float	floating point	SQL_DOUBLE
decimal	numeric	SQL_NUMERIC

Table 6-2 Python 2

从SUNDB接收的SQL值

以下为在python接收SUNDB数据库的数据时的数据转换

ODBC datatype	Description	Python datatype
any	NULL	None
SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR	text	text
SQL_BINARY_SQL_VARBINARY, SQL_LONGVARBINARY	binary	bytes
SQL_NUMERIC	decimal, numeric	decimal.Decimal
SQL_BOOLEAN	bit, bool	bool
SQL_SMALLINT, SQL_INTEGER	integers	int
SQL_BIGINT	long	long
SQL_REAL, SQL_FLOAT, SQL_DOUBLE	floating point	float
SQL_TYPE_TIME	time	datetime.time
SQL_TYPE_DATE	date	datetime.date
SQL_TYPE_TIMESTAMP	timestamp	datetime.timestamp
SQL_TYPE_TIME_WITH_TIMEZONE	time with timezone	text
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	timestamp with timezone	text
SQL_C_INTERVAL_***	interval	text

Table 6-3 Python 3

ODBC datatype	Description	Python datatype
any	NULL	None
SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR	text	text
SQL_BINARY_SQL_VARBINARY, SQL_LONGVARBINARY	binary	bytes
SQL_NUMERIC	decimal, numeric	decimal.Decimal
SQL_BOOLEAN	bit, bool	bool
SQL_SMALLINT, SQL_INTEGER	integers	int
SQL_BIGINT	long	long
SQL_REAL, SQL_FLOAT, SQL_DOUBLE	floating point	float
SQL_TYPE_TIME	time	datetime.time
SQL_TYPE_DATE	date	datetime.date
SQL_TYPE_TIMESTAMP	timestamp	datetime.timestamp
SQL_TYPE_TIME_WITH_TIMEZONE	time with timezone	text
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	timestamp with timezone	text
SQL_C_INTERVAL_***	interval	text

Table 6-4 Python 2

Python数据类型的text在python 3中转换为unicode在python 2根据数据库的字符集转换为

unicode或string

DB character set	Python type
UTF-8	str
SQL_ASCII	str
UHC	unicode
GB18030	unicode

Table 6-5 Python 2 text

7.Ruby

7.1 概要

SUNDB ruby驱动程序是使其可在以ruby编写的应用程序中使用SUNDB数据库的驱动程序以RubyGem安装包的形式提供

SUNDB ruby驱动程序基于SUNDB CLI驱动程序编写

7.2 安装

要求事项

需要在与ruby相同的系统上安装SUNDB的客户端安装包应设置$SUNDB_HOME环境变量

安装/卸载

可使用gem安装/卸载$SUNDB ruby驱动程序

卸载

卸载已安装的$SUNDB ryby驱动程序

```
% sudo -E gem uninstall ruby-sundb
```

安装

没有SUNDB ruby驱动程序安装包文件时生成安装包文件

```
% gem build sundb.gemspec

  Successfully built RubyGem

  Name: ruby-sundb

  Version: 3.2.0

  File: ruby-sundb-3.2.0.gem
```

有SUNDB ruby驱动程序安装包文件时使用gem安装安装包

```
% sudo -E gem install ruby-sundb-3.2.0.gem

Building native extensions.  This could take a while...

Successfully installed ruby-sundb-3.2.0

Parsing documentation for ruby-sundb-3.2.0

Installing ri documentation for ruby-sundb-3.2.0

Done installing documentation for ruby-sundb after 0 seconds

1 gem installed
```

7.3 示例

连接

使用DSN连接

```
require 'sundb'


$c = SUNDB.connect("SUNDB")
```

使用DSN与UIDPWD连接

```
require 'sundb'


$c = SUNDB.connect("SUNDB", "test", "test")
```

使用连接字符串连接

```
$c = SUNDB::Database.new.drvconnect("DSN=SUNDB")
```

```
$c =
SUNDB::Database.new.drvconnect("HOST=192.168.0.1;PORT=22581;UID=test;PWD=t
est")
```

生成表

```
$c.run("create table test (id int not null, str varchar(32) not null)")
```

插入数据

```
$q = $c.run("insert into test (id, str) values (1, 'foo')")
$q.run("insert into test (id, str) values (2, 'bar')")

$p = $c.proc("insert into test (id, str) values (?, ?)") {}
$p.call(3, "FOO")
$p[4, "BAR"]
```

检索数据

```
$q = $c.prepare("select id,str from test order by id")

if $q.column(0).name.upcase != "ID" then raise "fetch failed" end
if $q.column(1).name.upcase != "STR" then raise "fetch failed" end

$q.execute
if $q.fetch != [1, "foo"] then raise "fetch: failed" end
if $q.fetch != [2, "bar"] then raise "fetch: failed" end
```

```
if $q.fetch != [3, "FOO"] then raise "fetch: failed" end
if $q.fetch != [4, "BAR"] then raise "fetch: failed" end
if $q.fetch != nil then raise "fetch: failed" end
$q.close

if $q.execute.entries != [[1, "foo"], [2, "bar"], [3, "FOO"], [4, "BAR"]]
then
  raise "fetch: failed"
end
$q.close

if $q.execute.fetch_all != [[1, "foo"], [2, "bar"], [3, "FOO"], [4,
"BAR"]] then
  raise "fetch: failed"
end
$q.close

$q.execute
if $q.fetch_many(2) != [[1, "foo"], [2, "bar"]] then raise "fetch: failed"
end
if $q.fetch_many(3) != [[3, "FOO"], [4, "BAR"]] then raise "fetch: failed"
end
if $q.fetch_many(99) != nil then raise "fetch: failed" end
$q.close
```

```
a = []
$q.execute {|r| a=r.entries}
if a.size != 4 then raise "fetch: failed" end
$q.close

a = []
$q.execute.each {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close

a = []
$q.execute.each_hash {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close

a = []
$q.execute.each_hash(true) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close

a = []
$q.execute.each_hash(:key=>:Symbol) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close
```

```
a = []
$q.execute.each_hash(:key=>:Symbol,:table_names=>true) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close

a = []
$q.execute.each_hash(:key=>:String,:table_names=>false) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close

a = []
$q.execute.each_hash(:key=>:Fixnum,:table_names=>false) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close
```

更新数据

```
$q = $c.run("update test set id=0, str='hoge'")
if $q.nrows != 4 then
  $stderr.print "update row count: expected 4, got ", $q.nrows, "\n"
end
```

删除数据

```
$count = $c.do("delete from test where 1 = 1")
if $count != 4
  $stderr.print "delete row count: expected 4, got ", $count, "\n"
end
$count = $c.do("delete from test where 1 = 1")
if $count != 0
  $stderr.print "delete row count: expected 0, got ", $count, "\n"
end
```

整理已生成的所有statement

```
$c.drop_all
```

删除表

```
$c.do("drop table test")
```

解除访问

```
$c.disconnect
```

7.4 ActiveRecord的使用示例

为举例提前生成如下表

```
CREATE TABLE members
(
  id         INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  name       VARCHAR(255),
  email      VARCHAR(255),
  created_at TIMESTAMP,
  updated_at TIMESTAMP
);
```

连接

```
require 'rubygems'
require 'active_record'
require 'sundb'

ActiveRecord::Base.establish_connection(
    :adapter => "sundb",
    :host => "192.168.0.1",
    :port => "22581",
    :uid => "TEST",
```

```
    :pwd => "test",
    :option => "" )
```

用于连接的属性值如下

属性	说明
:adapter	sundb
:dsn	ODBC 数据源名称 (DSN)
:host	服务器IP地址
:port	服务器端口号
:uid	用户 ID
:pwd	用户 password
:opt	连接字符串

插入数据

```
class Members < ActiveRecord::Base
end


Members.create(:name => "David",  :email => "David@mail.com")
Members.create(:name => "Olivia", :email => "Olivia@mail.com")
Members.create(:name => "Amelia", :email => "Amelia@mail.com")
```

```
Members.create(:name => "Hazel",  :email => "Hazel@mail.com")

Members.create(:name => "Lily",   :email => "Lily@mail.com")
```

检索数据

```
require 'pp'


puts "\nfind(1)"

puts '-'*80

David = Members.find(1)

pp(David)

puts '-'*80


puts "\nwhere(:name => \"Amelia\")"

puts '-'*80

Amelia = Members.where(:name => "Amelia")

pp(Amelia)

puts '-'*80


puts "\nfind_each"

puts '-'*80

Members.find_each do |member|

  pp(member)

end
```

```
puts '-'*80
```

更新数据

```
puts "\nupdate Lily -> Harry"
puts '-'*80
Members.transaction do
 Lily = Members.where(:name => 'Lily')[0]
 Lily.name = 'Harry'
 Lily.save
end

puts "\nfind_each"
Members.find_each do |member|
 pp(member)
end
puts '-'*80
```

删除数据

```
puts "\ndelete Harry"
puts '-'*80
Members.transaction do
 Harry = Members.where(:name => 'Harry')[0]
```

```
 Harry.destroy
end


puts "\nfind_each"
Members.find_each do |member|
 pp(member)
end
puts '-'*80
```

8.Hibernate

8.1 概要

Hibernate是将Java Persistent API (JPA)模型适用于数据库的Object-Replation Mapping (ORM)框架映射DB表与Java对象之间的关系帮助处理持续性逻辑（persistent logic）即使用Hibernate可在不使用SQL的情况下轻松访问并控制DB

8.2 联动

下载

可在**https://sourceforge.net/projects/hibernate/**下载Hibernate解压已下载的文件后lib目录中显示多个jar文件使用该文件可以与Hibernate联动

> Note:
>
> SUNDB的Hibernate基于Hibernate 5.2版本编写

SundbDialect Class 联动

SUNDB提供符合Hibernate ORM版本的SundbDialect.java文件在Hibernate ORM 4版本使用Sundb.java 5版本编译会报错因此需使用符合Hibernate版本的SundbDialect.java文件

尤其是Hibernate ORM 4版本根据补丁版本所提供的api不同因此SUNDB也提供符合其的SundbDialect.java文件例如所使用的Hibernate ORM的版本为4.1.4则使用Sundb.java 4.0.0版本；Hibernate ORM版本为4.1.9则使用SundbDialect.java 4.1.5版本即可

SUNDB在$SUNDB_HOME/app_dev/Hibernate/ 下面的目录提供SundbDialect.java

移植到Hibernate jar文件

联动hibernate需要移植到下载的Hibernate jar文件如下可将SundbDialect.class和SundbDialect$1.class文件包含在Hibernate jar文件

1. 将SundbDialect.java文件编译为javac后生成class文件（各版本生成的class文件不同）
2. 解压Hibernate jar文件
3. 将SundbDialect.class和SundbDialect$1.class文件移动至org/hibernate/dialect目录
4. 再次压缩到Hibernate jar文件
5. 将新生成的jar文件添加到calsspath

```
$ javac -cp hibernate-cor-5.2.11.Final.jar SundbDialect.java


$ jar -xvf hibernate-core-5.2.11.Final.jar
```

```
$ pwd
hibernate-release-5.2.11.Final/lib/required/
$ mv SundbDialect*.class org/hibernate/dialect/


$ jar -cvf hibernate.5.2.11.jar  META-INF/MANIFEST.MF .
```

移植到program source

未在jar文件一直class文件的情况下可直接使用source文件进行联动在用户project生成org.hibernate.dialect package并将SundbDialect.java文件放进此package

8.3 示例

设置

联动hibernate与DB需要 hibernate-configuration XML文件与hibernate-mapping XML文件

Hibernate Configuration 文件

Configuration XML文件是设置用于访问dialect class与DB的联系信息及Hibernate-mapping文件的文件

详细内容参考**Configuration**

以下为hibernate.conf.xml文件的示例

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.driver_class">csii.sundb.jdbc.SundbDriver</prop
erty>
        <property
```

```
name="hibernate.connection.url">jdbc:sundb://192.168.0.21:22581/test</prop
erty>
        <property name="hibernate.connection.username">test</property>
        <property name="hibernate.connection.password">test</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.SundbDialect</property>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto"> create </property>
        <mapping resource="SampleTable.mapping.xml" />
    </session-factory>
</hibernate-configuration>
```

在<session-factory> tag设置DB连接信息后指定要连接hibernate.dialect 属性的DB的Dialect class名称另外使用 <mapping-resource> tag指定 hibernate-mapping 文件

Hibernate Mapping 文件

Mapping XML是包含DB表与Java object映射信息的配置文件

详细内容参考**Mapping**

以下为在上述示例中使用的 SampleTable.mapping.xml 文件的例子

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
```

```
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">


<hibernate-mapping>
    <class name="SampleTable" table="SAMPLETABLE">
        <id column="ID" name="id" type="long">
            <generator class="increment" />
        </id>
        <property column="STUDENT_NAME" name="name" type="string" />
        <property column="BIGINT_VALUE" name="bigintValue" type="long" />
        <property column="INT_VALUE" name="intValue" type="int" />
        <property column="SMALL_VALUE" name="smallValue" type="short" />
        <property column="FLOAT_VALUE" name="floatValue" type="float" />
        <property column="DOUBLE_VALUE" name="doubleValue" type="double"
/>
        <property column="NUMERIC_VALUE" name="numericValue"
type="big_integer" precision="10" scale="5" />
        <property column="DATE_VALUE" name="dateValue" type="date" />
        <property column="TIME_VALUE" name="timeValue" type="time" />
        <property column="TIMESTAMP_VALUE" name="timestampValue"
type="timestamp" />
        <property column="VARBINARY_VALUE" name="binaryValue"
type="byte[]" />
    </class>
</hibernate-mapping>
```

<class> tag中描述Java class与DB表的映射信息<property> tag中指定表的column与Java使用的数据类型

Application 示例

描述映射于DB表的Java class以下为上述说明的 SampleTable.java的例子

```
import java.math.BigInteger;
import java.sql.Date;
import java.sql.Time;
import java.sql.Timestamp;


public class SampleTable {
    private long id;
    private String     name;
    private long       bigintValue;
    private int        intValue;
    private short      smallValue;
    private float      floatValue;
    private double     doubleValue;
    private BigInteger numericValue;
    private Date       dateValue;
    private Time       timeValue;
    private Timestamp  timestampValue;
    private byte[]     binaryValue;
```

```
    private String     longvarcharValue;
    private byte[]     longvarbinaryValue;

    public SampleTable() {
        this.name = null;
    }

    public SampleTable( String name ) {
        this.name = name;
    }

    public long getId() {
        return id;
    }

    public void setId( long id ) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName( String name ) {
        this.name = name;
```

```
    }
....
```

需要编写对应SampleTable表的column的变量的get/set函数

SampleTable表如下处理数据

1. 通过configuraion 文件获取 org.hibernate.SessionFactory 对象
2. 通过SessionFactory 对象获取 org.hibernate.Session 对象
3. 调用Session对象的method进行所需操作

以下为在DB的SampleTable表插入变更查询删除数据的应用程序示例通过

org.hibernate.Configuration Class获取SessionFactory 对象

```
public class HibernateSample
{
    private static SessionFactory mFactory;

    public static void main( String[] args ) {
        try {
            mFactory = new
Configuration().configure( "hibernate.conf.xml" ).buildSessionFactory();
        } catch( Throwable ex ) {
            System.err.println( "Failed to create sessionFactory object." +
ex);
            throw new ExceptionInInitializerError(ex);
        }
```

```
        HibernateSample sHibernateSample = new HibernateSample();
```

- Insert 3 records

```
Long sRecord1 = sHibernateSample.addSample( "RECORD_1" );
Long sRecord2 = sHibernateSample.addSample( "RECORD_2" );
Long sRecord3 = sHibernateSample.addSample( "RECORD_3" );
```

- Update record 1

```
sHibernateSample.updateSample( sRecord1, ....
```

- Fetch table

```
sHibernateSample.fetchSample( 0, 3 );
```

- Delete record 2

```
sHibernateSample.deleteSample( sRecord2 );


sHibernateSample.closeFactory();
    }


public void closeFactory() {
        mFactory.close();
    }
```

```
....
```

以下为在HibernateSample Class插入记录的method示例首先通过SessionFactory获取session对象后获取org.hibernate.Transaction对象使用Transaction class开始transaction并插入记录后提交或回滚

```
public Long addSample( String name ) {
    Session session = mFactory.openSession();
    Transaction tx = null;
    Long id = null;

    try{
```

- Begin transaction

```
tx = (Transaction) session.beginTransaction();
SampleTable sample = new SampleTable( name );
```

- Insert

```
id = (Long) session.save(sample);
```

- Commit transaction

```
tx.commit();

        System.out.println("add success");
```

```
    } catch( HibernateException e) {
        if( tx != null ) {
            tx.rollback();
        }
        System.out.println("add fail");
        e.printStackTrace();
    } finally {
        session.close();
    }
    return id;
}
```

以下为查询SampleTable表的method的示例通过Session Class获取org.hibernate.Query对象fetchSample method从offset查询到count数量的SampleTable

```
public void fetchSample( int offset, int count ) {
    Session session = mFactory.openSession();
    Query query = null;

    try{
        query = session.createQuery( "FROM SampleTable" );

        query.setFirstResult( offset );
        query.setMaxResults( count );
        query.setFetchSize( 20 );
```

```
        @SuppressWarnings( "unchecked" )
        List<SampleTable> samples = (List<SampleTable>) query.list();

        for( Iterator<SampleTable> iter = samples.iterator();
iter.hasNext(); ) {
            SampleTable sample = (SampleTable) iter.next();
            System.out.print( "id: " + sample.getId() );
            ... 省略 ...
        }
    } catch ( HibernateException e ) {
        e.printStackTrace();
    } finally {
        session.close();
    }
}
```

以下为更新SampleTable表的method示例变更为SampleTable Class中编写的set method后调用session Class的update method并更新

```
public void updateSample( long id, ... ) {
    Session session = mFactory.openSession();
    Transaction tx = null;
    try {
        tx = session.beginTransaction();
        SampleTable sample = (SampleTable)session.get(SampleTable.class,
id);
```

```
        sample.setName( name );
        ...
        session.update( sample );
        tx.commit();
        System.out.println( "update success" );
        } catch ( HibernateException e ) {
            if( tx != null ){
                tx.rollback();
            }
            System.out.println("update fail");
            e.printStackTrace();
        } finally {
    session.close();
    }
}
```

以下为删除记录的示例可使用Session Class的delete method删除

```
public void deleteSample( long id ) {
    Session session = mFactory.openSession();
    Transaction tx = null;

    try {
        tx = session.beginTransaction();
        SampleTable sample = (SampleTable) session.get( SampleTable.class,
id );
```

```
        session.delete( sample );
        tx.commit();

        System.out.println( "delete success" );
    } catch ( HibernateException e ) {
        if( tx != null ) {
            tx.rollback();
        }
        System.out.println("delete fail");
        e.printStackTrace();
    } finally {
        session.close();
    }
}
```