编程

SAP Sybase IQ 16.0 SP03

文档 ID： DC02021-01-1603-01
最后修订日期： 2013 年 11 月

目录

合作伙伴认证

SAP® Sybase® IQ 合作伙伴生态系统包括获得认证的合作伙伴、数据仓库基础结构合作伙伴、分析解决方案合作伙伴以及商务智能合作伙伴应用程序。

有关认证报告和 SAP Sybase IQ 合作伙伴列表，请参见 SAP Sybase IQ Marketplace。

平台认证

*获得认证*表示某种产品可在特定平台环境中运行并且受到该平台支持。SAP Sybase IQ 经认证可在具有特定 CPU 体系结构组合的操作系统中运行。

有关获得认证的产品平台组合，请访问 http://certification.sybase.com/ucr/search.do。

SAP Sybase IQ 作为客户端应用程序的数据服务器

SAP Sybase IQ 支持通过 ODBC 或 JDBC 连接客户端应用程序。将 SAP Sybase IQ 用作客户端应用程序的数据服务器。

存在某些限制时，SAP Sybase IQ 可能也会作为特定客户端应用程序的 Open Server™ 出现。

本章所介绍的功能不为 Windows 和 Sun Solaris 系统上的 IQ 用户提供远程数据访问。远程数据访问由组件集成服务 (CIS) 提供，CIS 为 Enterprise Connect™ Data Access (ECDA) 的核心互操作功能。

Open Client 体系结构

Sybase Open Client™ 应用程序开发的主要文档为 Open Client 文档，您可从 SAP 获得。本节介绍的是专门面向 SAP Sybase IQ 的功能，而不是 Sybase Open Client 应用程序编程的详尽指南。

Sybase Open Client 具有两种组件：编程接口和网络服务。

DB-Library 和 Client Library。

Sybase Open Client 提供了两个核心编程接口以供编写客户端应用程序时使用：DB-Library™ 和 Client-Library。

Open Client DB-Library 为早期版本的 Open Client 应用程序提供支持，是与 Client-Library 完全分开的编程接口。Sybase Open Client 产品附带的 Open Client DB-Library/C 参考手册对 DB-Library 进行了介绍。

Client-Library 程序还依赖于 CS-Library，在 Client-Library 和 Server-Library 应用程序中都使用 CS-Library 提供的例程。Client-Library 应用程序还可以使用 Bulk-Library 中的例程来促进数据的高速传输。

CS-Library 和 Bulk-Library 都包含在 Sybase Open Client 中，两者分开使用。

网络服务

Open Client 网络服务包括 Sybase Net-Library，Sybase Net-Library 为特定网络协议（如 TCP/IP 和 DECnet）提供支持。Net-Library 接口对于应用程序开发人员来说是不可见的。但在某些平台上，针对不同的系统网络配置，应用程序可能需要另外一种 Net-Library 驱动程序。根据您的主机平台，Net-Library 驱动程序由系统的 Sybase 配置指定，或者在您编译和链接程序时指定。

有关驱动程序配置的说明，请参见 Open Client/Server 配置指南。

有关构建 Client-Library 程序的说明，请参见 Open Client/Server 程序员补充材料。

Open Client 和 jConnect 连接

当 SAP Sybase IQ 通过 TDS 为应用程序提供服务时，会自动将相关数据库选项的值设置为与 SAP Sybase SQL Anywhere® 服务器的缺省行为兼容。这些选项是临时设置的，它们仅在连接期间有效。客户端应用程序可以随时替换这些选项。

注意： SAP Sybase IQ 不支持 `ANSI_BLANKS`、`FLOAT_AS_DOUBLE` 和 `TSQL_HEX_CONSTANT` 选项。

虽然 SAP Sybase IQ 允许使用更长的用户名和口令，但 TDS 客户端用户名和口令不能超过 30 个字节。如果口令或用户 ID 超过 30 个字节，则通过 TDS 尝试连接（如使用 jConnect）时将返回无效的用户 `ID` 或口令错误。

注意： ODBC 应用程序（包括 Interactive SQL 应用程序）会自动将某些数据库选项设置为 ODBC 规范所规定的值。这将覆盖通过 `LOGIN_PROCEDURE` 数据库选项所做的设置。

login_procedure 选项

指定一个在启动时设置连接兼容性选项的登录过程。

允许值
字符串

缺省值
sp_login_environment 系统过程

作用域
可以为单个连接或 PUBLIC 设置此选项。您必须具有 SET ANY SECURITY OPTION 系统特权才能设置此选项。

注释
此登录过程在运行时调用 sp_login_environment 过程来判断数据库连接设置。执行完所有检查以验证连接有效后，调用该登录过程。对于事件连接，不执行 login_procedure 选项指定的过程。但对于 Web 服务连接执行该过程。

通过新建过程并设置 login_procedure 来调用新过程，可自定义缺省数据库选项设置。此自定义过程需要调用 sp_login_environment 或在检测到 TDS 连接时（参见缺省的 sp_login_environment 代码），直接调用 sp_tsql_environment。否则可能会使基于 TDS 的连接不能工作。请不要编辑 sp_login_environment 或 sp_tsql_environment。

可由用户定义的登录过程发出口令已到期错误消息 (SQLSTATE 08WA0) 的信号，向用户指明其口令已到期。发出错误信号可使应用程序检查该错误并处理到期的口令。

建议使用登录策略来实现口令到期功能，不建议使用会返回口令已过期错误消息的登录过程。

如果使用 NewPassword=* 连接参数，则需要对客户端库发出此错误信号，提示输入新口令。如果过程发出 SQLSTATE 28000（无效用户 ID 或口令）或 SQLSTATE 08WA0（已过期的口令）信号，或者过程通过 RAISERROR 报告错误，则登录失败并向用户返回一条错误消息。如果发出任何其它错误信号或发生另外错误，则用户登录成功并向数据库服务器消息日志写入一条消息。

示例

以下示例显示了如何通过发出 INVALID_LOGON 错误信号禁止一个连接。

```
CREATE PROCEDURE DBA.login_check( )
   BEGIN
      DECLARE INVALID_LOGON EXCEPTION FOR SQLSTATE '28000';
      // Allow a maximum of 3 concurrent connections
      IF( DB_PROPERTY( 'ConnCount' ) > 3 ) THEN
          SIGNAL INVALID_LOGON;
      ELSE
          CALL sp_login_environment;
      END IF;
   END
go

GRANT EXECUTE ON DBA.login_check TO PUBLIC
go

SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

以下示例显示了用户在 30 分钟的时间段内连接失败次数超过 3 次时，您如何阻止连接尝试。受阻周期内的所有受阻尝试都会接收到无效的口令错误，并被记录为失败。该日志会保留足够长的时间以供 DBA 对它进行分析。

```
CREATE TABLE DBA.ConnectionFailure(
    pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
    user_name CHAR(128) NOT NULL,
    tm TIMESTAMP NOT NULL DEFAULT CURRENT TIMESTAMP
)
go

CREATE INDEX ConnFailTime ON DBA.ConnectionFailure(
    user_name, tm )
go

CREATE EVENT ConnFail TYPE ConnectFailed
HANDLER
BEGIN
    DECLARE usr CHAR(128);
    SET usr = event_parameter( 'User' );

    // Put a limit on the number of failures logged.
    IF (SELECT COUNT(*) FROM DBA.ConnectionFailure
        WHERE user_name = usr
```

```
        AND tm >= DATEADD( minute, -30,
            CURRENT TIMESTAMP )) < 20 THEN
        INSERT INTO DBA.ConnectionFailure( user_name )
            VALUES( usr );
        COMMIT;
        // Delete failures older than 7 days.
        DELETE DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm < dateadd( day, -7, CURRENT TIMESTAMP );
        COMMIT;
    END IF;
END
go

CREATE PROCEDURE DBA.login_check( )
BEGIN
    DECLARE usr CHAR(128);
    DECLARE INVALID_LOGON EXCEPTION FOR SQLSTATE '28000';
    SET usr = CONNECTION_PROPERTY( 'Userid' );
    // Block connection attempts from this user
    // if 3 or more failed connection attempts have occurred
    // within the past 30 minutes.
    IF ( SELECT COUNT( * ) FROM DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm >= DATEADD( minute, -30,
            CURRENT TIMESTAMP ) ) >= 3 THEN
        SIGNAL INVALID_LOGON;
    ELSE
        CALL sp_login_environment;
    END IF;
END
go

GRANT EXECUTE ON DBA.login_check TO PUBLIC
go

SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

以下示例显示了如何发出提示用户口令已到期的错误信号。建议使用登录策略来实现口令到期通知。

```
CREATE PROCEDURE DBA.check_expired_login( )
BEGIN
  DECLARE PASSWORD_EXPIRED EXCEPTION FOR SQLSTATE '08WA0';

  IF( condition-to-check-for-expired-password ) THEN
      SIGNAL PASSWORD_EXPIRED;
  ELSE
      CALL sp_login_environment;
  END IF;
END;
```

具有多个数据库的服务器

使用 Open Client Library，可以连接到包含多个数据库的服务器上的特定数据库。

- 在服务器的 interfaces 文件中设置相应条目。
- 使用 **start_iq** 命令的 **-n** 参数为数据库名称设置快捷方式。
- 在 **isql** 命令中用数据库名称指定 **-S***database_name* 参数。进行连接时，需要此参数。

可以在不更改程序本身的情况下，通过将快捷方式名称放到程序中并只更改快捷方式定义，对多个数据库运行相同的程序。

例如，以下 interfaces 文件片段定义了 live_sales 和 test_sales 两个服务器：

```
live_sales

     query tcp ether myhostname 5555
    master tcp ether myhostname 5555

test_sales

     query tcp ether myhostname 7777
    master tcp ether myhostname 7777
```

启动服务器并为特定数据库设置别名。以下命令将 live_sales 设置为等效于 salesbase.db：

```
start_iq -n sales_live <other parameters> -x \  'tcpip{port=5555}'
salesbase.db -n live_sales
```

要连接到 live_sales 服务器：

```
isql -Udba -Psql -Slive_sales
```

服务器名只能在 interfaces 文件中出现一次。因为与 SAP Sybase IQ 的连接现在基于数据库名称，所以数据库名称必须唯一。如果所有脚本都设置为作用于 salesbase 数据库，则不必将脚本修改为处理 live_sales 或 test_sales。

在应用程序中使用数据库内分析

SAP Sybase IQ 提供了三种形式的数据库内分析：本地内置分析、本地 UDF 插件分析和外部 UDF 插件分析。开发人员可以通过以外部 UDF 的形式提供分析，启用对大数据的复杂分析。

- **本地内置分析** – 本地内核分析的示例包括 OLAP 和全文搜索。**CUME_DIST** 函数就是一个内置 ANSI SQL OLAP 内置集合函数的示例。
- **本地 UDF 插件分析** – 使用进程外共享库，可以开发*文本分析*解决方案。通过开发进程外的数据库内 UDF，可以将运行进程内用户定义代码所固有的安全性和稳健性风险降到最低。有关 LOB 文档，请参见非结构化数据分析。
- **外部 UDF 插件分析** – 使用 Java UDF、表 UDF 和表参数化函数 (TPF) 为大数据开发进程外分析解决方案。

另请参见

- 附录：使用 OLAP （第 543 页）

标量 C 或 C++ UDF

标量 UDF 是作用于单个值的 V3 或 V4 外部 C 或 C++ 过程。

有关详细信息和示例，请参见用户定义的函数。外部 C 和 C++ 过程需要单独授权的 SAP Sybase IQ 组件。

集合 C 或 C++ UDF

集合 UDF 是作用于多个值的 V3 或 V4 外部 C 或 C++ 过程。有时，集合 UDF 也称为 UDA 或 UDAF。用于编写集合 UDF 代码的环境结构与用于编写标量 UDF 代码的环境结构稍有不同。

有关详细信息和示例，请参见用户定义的函数。外部 C 和 C++ 过程需要单独授权的 SAP Sybase IQ 组件。

Java UDF

Java UDF 的行为与 SQL 函数的行为基本相同，只是过程或函数的代码以 Java 编写并且在数据库服务器外（即 Java VM 环境内）执行。可以定义 Java 标量 UDF 和 Java 表 UDF。

Java UDF 不需要单独授权的 SAP Sybase IQ 组件。

Java 标量 UDF

以 Java 代码实现的进程外（外部环境）标量用户定义函数。

有关详细信息和示例，请参见用户定义的函数。

Java 表 UDF

以 Java 代码实现的进程外（外部环境）表 UDF。

有关详细信息和示例，请参见用户定义的函数。

表 UDF

表 UDF 是用户定义的外部 C、C++ 或 Java 表函数。与标量 UDF 和集合 UDF 不同，表 UDF 产生行集形式的输出。SQL 查询可以在 SQL 语句的 FROM 子句中将行集用作表表达式。

标量 UDF 和集合 UDF 可以使用 v3 或 v4 extfn API，而表 UDF 只能使用 v4。

有关详细信息和示例，请参见用户定义的函数。

TPF

表参数化函数 (TPF) 是增强的表 UDF，接受标量值或行集作为输入。可以为 TPF 配置用户指定的分区。UDF 开发人员可以声明一种分区方案，用于将数据集分解为跨 Multiplex 节点分配的小型查询处理块。这样，便可在分布式服务器系统中对行集分区并行执行 TPF。查询引擎支持大量并行的 TPF 处理。

有关详细信息和示例，请参见用户定义的函数。

Hadoop 集成

SAP Sybase IQ 包含可用于构建 MapReduce 组件的 UDF API，这些 MapReduce 组件可用于 Hadoop 集成。SAP Sybase 解决方案存储库中包含 Hadoop 集成的示例。

MapReduce 编程模型用于进行大量并行的分布式计算。MapReduce 编程模型包含两个主要阶段：

- **映射阶段** – 主节点将问题分为多个子问题或*映射*。这些映射必须彼此独立且并行执行。
- **缩减阶段** – 主节点收集子问题的答案并以有意义的方式将其组合在一起，从而得出原始问题的答案。

Apache Hadoop 是一种 MapReduce 实现。Hadoop 是一种可以自动调度映射和缩减作业的 Java 软件框架。

SAP Sybase IQ 支持使用表参数化函数 (TPF)（一类外部用户定义的函数）进行类似 Hadoop 的并行调度。TPF 接受表值输入参数的任意行集，并且可以在分布式服务器环境中并行处理。可以指定 TPF 输入的分区和排序要求。借助 SQL，开发人员可以在数据库服务器中通过 TPF 来利用 MapReduce 范例。

有关 TPF 基本原理，请参见用户定义的函数 指南。

将 SAP Sybase IQ 与 Hadoop 分布式文件系统集成

Hadoop 分析返回的数据可以多种方式集成到 SAP Sybase IQ 数据库中。

- **ETL 处理** – 使用开放源实用程序 SCOOP 将 Hadoop 数据存储库中的数据批量装载到 SAP Sybase IQ。
- **数据联合** – 将 HDFS 文件显示为 SAP Sybase IQ 数据库中参与 SQL 查询的表。HDFS 文件无需装载到 SAP Sybase IQ。
- **查询联合** – 允许 SAP Sybase IQ 中的 SQL 查询执行 Hadoop 进程，从而返回将并入 SQL 结果集中的数据。
- **客户端联合** – 使用 TOAD™ SQL 工具联合跨 SAP Sybase IQ 数据库和 Hadoop 文件的查询。

将 Hadoop 分布式文件系统中的文件作为内存中的表读取

在数据联合示例中，SAP Sybase IQ 将 Hadoop 分布式文件系统 (HDFS) 中的文件作为内存中的表读取。

注意： 此示例代码主要用于说明目的，并非针对生产用途。尽管已尽力确保合理处理了错误，但这些示例并非处于生产级别，所以在投入生产前仍需额外的安全保护措施和测试。

1. 创建 Java 类：

```
public class HDFSclient {
    public static void readFileByLine(String file, ResultSet
rset[])
throws IOException {

// Set Configuration to point to HDFS NameNode and find input dir
Configuration conf = new Configuration();
conf.addResource(new Path( “/home/mymachine/hadoop/conf/core-
site.xml” ));
FileSystem fileSystem = FileSystem.get(conf);
Path path = new Path(file);
if (!fileSystem.exists(path)); {
    System.out.println( “File ”  + file +  “ does not exists” );
    return;
}

// Create meta data for the result set
ResultSetMetaDataImpl rsmd = new ResultSetMetaDataImpl(1);
rsmd.setColumnType(1, Typs.VARCHAR);
rsmd.setColumnName(1, ” c1” );
rsmd.setColumnLabel(1, ” c1” );
```

```
rsmd.setColumnDisplaySize(1, ”c1” );
rsmd.setTableName(1, ” MyTable” );

// Create ResultSet using the meta data
ResultSetImpl rs = null;
try {
    rs = new ResultSetImpl((ResultSetMetaDataImpl)rsmd);
    rs.beforeFirst();// Make sure we are at the beginning
} catch(Exception e) {
    System.out.println(” Could not create result set.” );
    System.out.println(e.toString());
}

// Read files from input dir line by line inserting into rs
String line;
DataInputStream in = new DataInputSteam(fileSystem.open(path));
BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
while ((line = reader.readline()) != null) {
try {
        rs.insertRow();// Insert a new row
        rs.updateString(1,(line));
} catch(Exception e) {
    System.out.println(” Could not insert row/data” );
    System.out.println(e.toString());
}
}
try {
rs.beforeFirst();// Make sure we are at the beginning
} catch(Exception e) {
    System.out.println(e.toString());
    }

rset[0] = rs; // Assign result set to the 1st of the passé din
array.

in.close();
reader.close();
fileSystem.close();

}
}
}
```

2. 安装类或封装的 JAR 文件：

```
INSTALL JAVA NEW JAR ‘myjar’ FROM FILE ‘/home/mymachine/UDFs/
myjar.jar’ ;
```

3. 创建函数：

```
CREATE or REPLACE PROCEDURE readFileByLine( IN fileName CHAR(50) )
RESULT ( c1 VARCHAR(255) )
EXTERNAL NAME 'example.HDFSclient.readFileByLine(Ljava/lang/
String;[Ljava/sql/ResultSet;)V'
LANGUAGE JAVA;
```

4. 执行函数：

```
SELECT c1 FROM readFileByLine('/home/mymachine/input/input.txt');
```

启动外部 Hadoop MapReduce 作业并使用查询中的结果

定义映射和缩减方法，以输入和输出按 <key, value> 对构成的数据。

假定有一个目录含有两个文本文件，文本文件的内容如下：

- `File1.txt`：Hello World Goodbye World
- `File2.txt`：Goodbye World Hadoop

1. 在映射步骤中，将每个文件作为单独的映射作业处理，所有这些映射的输出均为以下 <key, value>：
   - 作业 1：<Hello, one> <World, one> <Goodbye, one> <World, one>
   - 作业 2：<Goodbye, one> <world, one> <Hadoop, one>
2. 调用仅添加映射阶段输出的 <key, value> 的缩减程序。本地缩减程序的输出为：
   - 作业 1：<Hello, one> <World, two> <Goodbye, one>
   - 作业 2：<Goodbye, one> <World, one> <Hadoop, one>
3. 组合起来，获得最终输出：
   `<Hello, one> <World, 3><Goodbye,2><Hadoop, 1>`

注意： 此示例代码主要用于说明目的，并非针对生产用途。尽管已尽力确保合理处理了错误，但这些示例并非处于生产级别，所以在投入生产前仍需额外的安全保护措施和测试。

```
     public class WordCountDriver extends Configured {
    public static void String HADOOP_ROOT_DIR = “hdfs://localhost:
9000”
private Text word = new Text();
private final IntWritable one = new IntWritable(1);

static class WordCountMapper extends Mapper<LongWritable, Text,
Text, IntWritable> {

public void map(LongWritable key Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
StringTokenizer itr = new StringTokenizer(line.toLowerCase());
while (itr.hasMoreTokens()){
    word.set(itr.next(Token));
    context.write(word, one);
}
};

static class WordCountReducer extends Reducer<Text, IntWritable,
Text, IntWritable > {

public void reduce (Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
    int sum = 0;
for (IntWritable value : values) {
```

```
    sum += value.get();
}
context.write(key, new IntWrtiable(sum));
}
};

Public static void run(String input, String output, ResultSet rs[])
throws Exception {
Configuration conf = new Configuration();
conf.addResource(new Path(“/home/mymachine/hadoop/conf/core-
site.xml”));
conf.set(“fs.default.name”,” hdfs://localhost:9000”);
conf.set(“mapred.job.tracker”,” localhost:9000”);

// Specify output types
Job job = new Job(conf, “Word Count”);
Job.setOutputKeyClass(Text.class);
Job.setOutputValueClass(IntWritable.class);

// Specify input and output locations
FileInputFormat.addInputPath(job, new Path(HADOOP_ROOT_DIR+input));)
FileOutputFormat.addInputPath(job, new Path(HADOOP_ROOT_DIR
+output));

// Specify a mapper
job.setMapperClass(WordCountDriver.WordCountMapper.class);

// Specify a reducer
job.setReducerClass(WordCountDriver.WordCountReducer.class);
job.setCombinerClass(WordCountDriver.WordCountReducer.class);
job.setJarByClass(WordCountDriver.class)

// Wait for MR job to complete
while (job.waitForCompletion(true) ? false : true) {
    // Waiting…
}
    HDFSclient hdfsc = new HDFSclient();
    hdfsc.readFileByLine(file, rs);
   }
}
}
```

a_v4_extfn 的 API 参考

针对 a_v4_extfn 函数、方法和属性的参考信息。

Blob (a_v4_extfn_blob)

请使用 a_v4_extfn_blob 结构以表示一个独立的 BLOB 对象。

实现

```
typedef struct a_v4_extfn_blob {
    a_sql_uint64 (SQL_CALLBACK *blob_length)(a_v4_extfn_blob *blob);
    void (SQL_CALLBACK *open_istream)(a_v4_extfn_blob *blob,
a_v4_extfn_blob_istream **is);
    void (SQL_CALLBACK *close_istream)(a_v4_extfn_blob *blob,
a_v4_extfn_blob_istream *is);
    void (SQL_CALLBACK *release)(a_v4_extfn_blob *blob);
} a_v4_extfn_blob;
```

方法总结

方法名称	数据类型	描述
blob_length	a_sql_uint64	以字节为单位返回指定 BLOB 的长度。
open_istream	无类型	打开一个可用于自指定 BLOB 开始读取数据的输入流。
close_istream	无类型	关闭针对于指定 BLOB 的输入流。
release	无类型	指示调用方已完成对此 blob 的调用，并且 blob 所有者可以释放资源。**release()** 调用完后引用 blob 会产生错误。调用 **release()** 时所有者通常会删除内存。

描述

如下情况时请使用 a_v4_extfn_blob 对象：

- 表 UDF 需要从标量输入值读取 LOB 或 CLOB 数据
- TPF 需要从输入表中的列读取 LOB 或 CLOB 数据

约束和限制

无。

blob_length

使用 blob_length v4 API 方法返回指定 BLOB 的长度（以字节为单位）。

声明

```
a_sql_uint64 blob_length(
    a_v4_extfn_blob *
)
```

用法

以字节为单位返回指定 BLOB 的长度。

参数

参数	描述
blob	需要获取长度值的 BLOB。

返回
指定的 BLOB 的长度。

另请参见

- open_istream （第 18 页）
- close_istream （第 19 页）
- release （第 19 页）

open_istream

请使用 `open_istream` v4 方法打开输入流，以便从 BLOB 读取数据。

声明

```
void open_istream(
    a_v4_extfn_blob *blob,
    a_v4_extfn_blob_istream **is
    )
```

用法
打开一个可用于自指定 BLOB 开始读取数据的输入流。

参数

参数	描述
blob	需要打开输入流的 BLOB。
is	输出参数，用于标识所返回的已打开输入流。

返回
无。

另请参见

- blob_length （第 17 页）
- close_istream （第 19 页）
- release （第 19 页）

close_istream

请使用 `close_istream` v4 API 方法关闭指定 BLOB 的输入流。

声明

```
void close_istream(
    a_v4_extfn_blob *blob,
    a_v4_extfn_blob_istream *is
    )
```

用法

关闭以前使用 `open_istream` API 打开的输入流。

参数

参数	描述
blob	需要关闭输入流的 BLOB。
is	标识所要关闭的输入流的参数。

返回

无。

另请参见

- blob_length （第 17 页）
- open_istream （第 18 页）
- release （第 19 页）

release

请使用 `release` v4 API 方法指示调用方已经使用完当前选定的 BLOB 。所有者得以释放内存。

声明

```
void release(
a_v4_extfn_blob *blob
)
```

用法

指示调用方已完成对此 blob 的调用，并且 blob 所有者可以释放资源。**release()** 调用完后引用 blob 会产生错误。调用 **release()** 时所有者通常会删除内存。

参数

参数	描述
blob	需要释放的 BLOB。

返回
无。

另请参见

- blob_length （第 17 页）
- open_istream （第 18 页）
- close_istream （第 19 页）

Blob 输入流 (a_v4_extfn_blob_istream)

请使用 `a_v4_extfn_blob_istream` 结构为 LOB 或 CLOB 标量输入列、或输入表中的 LOB 或 CLOB 列读取 BLOB 数据。

实现

```
typedef struct a_v4_extfn_blob_istream {
    size_t (SQL_CALLBACK *get)( a_v4_extfn_blob_istream *is, void
*buf, size_t len );
    a_v4_extfn_blob     *blob;
    a_sql_byte          *beg;
    a_sql_byte          *ptr;
    a_sql_byte          *lim;
} a_v4_extfn_blob_istream;
```

方法总结

方法名称	数据类型	描述
get	size_t	从 BLOB 输入流中获取指定量的数据。

数据成员及数据类型的摘要

数据成员	数据类型	描述
Blob	a_v4_extfn_blob	基本 BLOB 结构（输入流据此创建）。
Beg	a_sql_byte	指向当前数据块起始位置的指针。
Ptr	a_sql_byte	指向数据块中当前字节的指针。
Lim	a_sql_byte	指向当前数据块结尾位置的指针。

get

请使用 `get` v4 API 方法从 BLOB 输入流获取指定量的数据。

声明

```
size_t get(
    a_v4_extfn_blob_istream *is,
    void *buf,
    size_t len
)
```

用法

从 BLOB 输入流中获取指定量的数据。

参数

参数	描述
is	需要从中检索数据的输入流。
buf	需要存储数据的缓存。
len	需要检索的数据量。

返回

接收的数据量。

列数据 (a_v4_extfn_column_data)

`a_v4_extfn_column_data` 结构代表一个单列的数据值。当生成结果集数据时，生产者将使用该结构，而当读取输入表列数据时，消耗程序将使用该结构。

实现

```
typedef struct a_v4_extfn_column_data {
    a_sql_byte          *is_null;
    a_sql_byte          null_mask;
    a_sql_byte          null_value;

    void                *data;
    a_sql_uint32        *piece_len;
    size_t              max_piece_len;

    void                *blob_handle;
} a_v4_extfn_column_data;
```

数据成员及数据类型的摘要

数据成员	数据类型	描述
is_null	a_sql_byte *	指向 存储 NULL 信息的字节。
null_mask	a_sql_byte	用于表示空值的一位或多位数据
null_value	a_sql_byte	表示空值
data	void*	指向列数据的指针。根据提取机制的类型，或者指向消耗程序中的地址，或者指向数据在 UDF 中的存储地址。
piece_len	a_sql_uint32 *	可变长度数据类型的实际数据长度
max_piece_len	size_t	此列的最大允许数据长度。
blob_handle	void*	非空值表示必须 使用 `blob` API 读取列数据

描述

`a_v4_extfn_column_data` 结构代表数据值以及特定数据列的相关属性。当生成结果集数据时，生产者将使用此结构。数据生产者也希望为 *data*、*piece_len* 和 *is_null* 标志创建存储空间。

is_null、*null_mask* 和 *null_value* 数据成员指示列中有空值，并且对某些情况（8 列共用一个包含有 NULL 位编码的字节）或者其他情况（每列使用一个完整字节）进行处置。

此示例描述如何解释用于代表 NULL 的三个字段：*is_null*、*null_mask* 和 *null_value*。

```
is_value_null()
   return( (*is_null & null_mask) == null_value )

set_value_null()
   *is_null = ( *is_null & ~null_mask) | null_value

set_value_not_null()
   *is_null = *is_null & ~null_mask | (~null_value & null_mask)
```

列的列表 (a_v4_extfn_column_list)

对 **PARTITION BY** 或 **TABLE_UNUSED_COLUMNS** 进行描述时，请使用 `a_v4_extfn_column_list` 结构以提供列的列表。

实现

```
typedef struct a_v4_extfn_column_list {
    a_sql_int32         number_of_columns;
    a_sql_uint32        column_indexes[1];      // there are
```

```
number_of_columns entries
} a_v4_extfn_column_list;
```

数据成员及数据类型的摘要

数据成员	数据类型	描述
number_of_columns	a_sql_uint32	列表中的列数。
column_indexes	a_sql_uint32 *	大小为 number_of_columns 且带有列索引（从 1 开始）的连续数组。

描述

列列表中的内容，其含义的改变取决于该列表是否同 **TABLE_PARTITIONBY** 或 **TABLE_UNUSED_COLUMNS** 一起使用。

列顺序 (a_v4_extfn_order_el)

请使用 `a_v4_extfn_order_el` 结构以描述列中的元素顺序。

实现

```
typedef struct a_v4_extfn_order_el {
    a_sql_uint32       column_index;    // Index of the column in the
table (1-based)
    a_sql_byte       ascending;             // Nonzero if the column
is ordered "ascending".
} a_v4_extfn_order_el;
```

数据成员及数据类型的摘要

数据成员	数据类型	描述
column_index	a_sql_uint32	表中列的索引（从 1 开始）。
ascending	a_sql_byte	如果列顺序为“升序”，则其值“非零”。

描述

`a_v4_extfn_order_el` 结构对列作了描述，并且表明该列是否应为升序或降序。`a_v4_extfn_orderby_list` 结构包含具有这些结构的一个数组。对于 **ORDERBY** 子句中的每一列，都有一个 `a_v4_extfn_order_el` 结构。

列子集 (a_v4_extfn_col_subset_of_input)

使用 `a_v4_extfn_col_subset_of_input` 结构声明输出列具有一个始终从特定输入列获取到 UDF 的值。

实现

```
typedef struct a_v4_extfn_col_subset_of_input {
    a_sql_uint32    source_table_parameter_arg_num;    // arg_num of
```

```
the source table parameter
    a_sql_uint32    source_column_number;        // source column of
the source table
} a_v4_extfn_col_subset_of_input;
```

数据成员及数据类型的摘要

数据成员	数据类型	描述
source_table_parameter_arg_num	a_sql_uint32 *	源 TABLE 参数的 *arg_num*
source_column_number	a_sql_uint32 *	源表中的源列

描述
查询优化程序使用输入子集来推断输出列中的值的逻辑属性。例如，输入列中离散值数量为输出列中离散值数量的上限，并且输入列上的任何本地谓词同样保持于输出列之上。

描述 API

_describe_extfn 函数是 `a_v4_extfn_proc` 的组成部分。UDF 使用 `a_v4_extfn_proc_context` 对象中的 `describe_column`、`describe_parameter` 和 `describe_udf` 属性以获取和设置逻辑属性。

_describe_extfn 声明

```
void (UDF_CALLBACK *_describe_extfn)(a_v4_extfn_proc_context
*cntxt );
)
```

用法
_describe_extfn 函数向服务器描述了过程评测。

每个 `describe_column`、`describe_parameter` 和 `describe_udf` 属性都有一个相关的获取和设置方法、一组属性类型以及与之其关联的数据类型。获取方法从服务器检索信息；而设置方法则向服务器描述 UDF 的逻辑属性（例如输出列的数量或者某一输出列的离散值数量）。

*describe_column_get

表 UDF 使用 `describe_column_get` v4 API 方法检索 TABLE 参数单个列的相关属性。

声明

```
a_sql_int32 (SQL_CALLBACK *describe_column_get)(
    a_v4_extfn_proc_context       *cntxt,
    a_sql_uint32                     arg_num,
    a_sql_uint32                        column_num,
    a_v4_extfn_describe_parm_type describe_type,
```

```
    void                                    *describe_buffer,
    size_t                                  describe_buffer_len );
```

参数

参数	描述
cntxt	此 UDF 的过程上下文对象。
arg_num	TABLE 参数的序号（0 为结果表，1 表示首个输入参数）。
column_num	列的序数从 1 开始。
describe_type	指示要检索的属性的选择器。
describe_buffer	对于指定的要从服务器获取的属性，是用于存储描述信息的结构。具体结构或数据类型由 **describe_type** 参数表示。
describe_buffer_length	**describe_buffer** 的字节长度。

返回
成功时会返回已写入 **describe_buffer** 的字节数。如果出错或未检索到属性，则此函数会返回某个通用 `describe_column` 错误。

***describe_column_get** 的属性*
以下代码中有 `describe_column_get` 第 4 版 API 方法的属性。

```
typedef enum a_v4_extfn_describe_col_type {
    EXTFNAPIV4_DESCRIBE_COL_NAME,
    EXTFNAPIV4_DESCRIBE_COL_TYPE,
    EXTFNAPIV4_DESCRIBE_COL_WIDTH,
    EXTFNAPIV4_DESCRIBE_COL_SCALE,
    EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
    EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
    } a_v4_extfn_describe_col_type;
```

EXTFNAPIV4_DESCRIBE_COL_NAME (Get)
EXTFNAPIV4_DESCRIBE_COL_NAME 属性指示列的名称。用于 `describe_column_get` 情形。

数据类型
`char[]`

描述
列名。该属性仅对表参数有效。

用法
如果 UDF 获取该属性，则返回指定列的名称。

返回
成功时会返回列名长度。

如果失败，则返回通用 `describe_column` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 缓冲区长度为 0 或字符数不足时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – 参数不为 TABLE 参数时返回此 get 错误。

查询处理阶段
在以下状态下有效：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_COL_TYPE (Get)

`EXTFNAPIV4_DESCRIBE_COL_TYPE` 属性指示列的数据类型。用于 `describe_column_get` 场景。

数据类型
`a_sql_data_type`

描述
列的数据类型。该属性仅对表参数有效。

用法
如果 UDF 获取该属性，则返回指定列的数据类型。

返回
如果成功，则返回 `sizeof(a_sql_data_type)`。

如果失败，则返回通用 `describe_column` 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 描述缓冲区大小不是 a_sql_data_type 时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理阶段不大于“初始”阶段时返回此 get 错误。

查询处理阶段

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_COL_WIDTH (Get)

EXTFNAPIV4_DESCRIBE_COL_WIDTH 属性指示列的宽度。用于 describe_column_get 情形。

数据类型
a_sql_uint32

描述
列宽。列宽是以字节为单位的存储空间量，用于存储关联数据类型的值。该属性仅对表参数有效。

用法
如果 UDF 获取该属性，则返回由 **CREATE PROCEDURE** 语句所定义的列的宽度。

返回
如果成功，则返回 sizeof(a_sql_uint32)。

如果失败，则返回通用 describe_column 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 描述缓冲区大小不是 a_sql_uint32 时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理阶段不大于“初始”阶段时返回此 get 错误。

查询处理阶段

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_COL_SCALE (Get)
The **EXTFNAPIV4_DESCRIBE_COL_SCALE** 属性指示列的标度。用于 `describe_column_get` 情形。

数据类型
`a_sql_uint32`

描述
列的标度。对于算术型数据类型，参数标度是指数字的小数点右边的位数。该属性仅对表参数有效。

用法
如果 UDF 获取该属性，则返回由 **CREATE PROCEDURE** 语句所定义的列的标度。此属性仅对算术数据类型有效。

返回
如果成功，则返回 `sizeof(a_sql_uint32)`，前提是返回值，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 指定列的数据类型的标度不可用时返回此 get 错误。

如果失败，则返回通用 `describe_column` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_sql_uint32` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“初始”阶段时返回此 get 错误。

查询处理阶段

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL (Get)
The **EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL** 属性指示列是否可以为空。用于 `describe_column_get` 情形。

数据类型
`a_sql_byte`

描述
如果列可以是 Null，则该值为“true”。该属性仅对表参数有效。该属性仅对参数 0 有效。

用法
当某个 UDF 获取该属性时，如果列可以为 NULL 则返回 1，否则返回 0。

返回
如果成功，则返回 `sizeof(a_sql_byte)`，前提是属性可用，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 无法获取属性时返回。列不包含在查询中时可能发生此情况。

如果失败，则返回通用 `describe_column` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_sql_byte` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 指定的参数为输入表且查询处理阶段不大于“计划构建”阶段时返回此 get 错误。

查询处理阶段
在以下状态下有效：

- 执行阶段

EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES (Get)

EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES 属性用于描述某一列的离散值。用于 `describe_column_get` 情形。

数据类型
`a_v4_extfn_estimate`

描述
某一列的估计离散值数量。该属性仅对表参数有效。

用法
如果 UDF 获得了该属性，则会返回某一列的估计离散值数量。

返回
如果成功，则返回 `sizeof(a_v4_extfn_estimate)`，前提是它返回值，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 无法获取属性时返回。列不包含在查询中时可能发生此情况。

如果失败，则返回通用 `describe_column` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_v4_extfn_estimate` 时返回此 get 错误。

- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 指定的参数为输入表且查询处理阶段大于“优化”阶段时返回此 get 错误。

查询处理阶段
在以下状态下有效:
- 计划构建阶段
- 执行阶段

示例
考虑 _describe_extfn API 函数中的过程定义及代码段:

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
RESULTS ( r1 INT, r2 INT, r3 INT )
EXTERNAL ‘my_tpf_proc@mylibrary’;
```

```
CREATE TABLE T( x INT, y INT, z INT );
```

```
select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

本示例显示了 TPF 如何获取输入表中某个列 的离散值数量。如果有益于选择合适的处理算法，则 TPF 可能需要得到 该值。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_PLAN_BUILDING ) {
           a_v4_extfn_estimate num_distinct;

           a_sql_int32 ret = 0;

         // Get the number of distinct values expected from the first
column
        // of the table input parameter 'col_table'
           ret = cntxt->describe_column_get( cntxt, 2, 1
                     EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
                     &num_distinct,
                     sizeof(a_v4_extfn_estimate) );

           // default algorithm is 1
           _algorithm = 1;

           if( ret > 0 ) {
               // choose the best algorithm for sample size.

               if ( num_distinct.value < 100 ) {
                   // use faster algorithm for small distinct values.
                       _algorithm = 2;
               }
           }
           else {
               if ( ret < 0 ) {
                       // Handle the error
                       //   or continue with default algorithm
               } else {
```

```
                        // Attribute was unavailable
                        // We will use the default algorithm.
                    }
                }
            }
}
```

EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE (Get)
EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE 属性指示某列在表中是唯一的。用于 `describe_column_get` 情形。

数据类型
`a_sql_byte`

描述
如果表中的列是唯一的，则该值为“true”。该属性仅对表参数有效。

用法
当 UDF 获取该属性时，如果列具有唯一性，则返回 1，否则返回 0。

返回
如果成功，则返回 `sizeof(a_sql_byte)`，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 无法获取属性时返回。列不包含在查询中时可能发生此情况。

失败时会返回某个通用 `describe_column` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_sql_byte` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“初始”阶段时返回此 get 错误。

查询处理阶段
在以下状态下有效：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT (Get)
EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT 属性指示某一列是否是常量。用于 `describe_column_get` 情形。

数据类型
`a_sql_byte`

描述
如果该列在语句的生存期内保持不变，则为“true”。该属性仅对输入的表参数有效。

用法
当 UDF 获取该属性时，如果列在语句的生存期内保持不变，则返回值为 1，否则返回 0。如果输入表的选择列表中的列为常量表达式或者 NULL，则输入表的列保持不变。

返回
如果成功，则返回 `sizeof(a_sql_byte)`，前提是它返回值，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性不可获取。列不包含在查询中时返回此错误。

失败时会返回某个通用 `describe_column` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_sql_byte` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定参数不是输入表时返回此 get 错误。

查询处理阶段
在以下状态下有效：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE (Get)
EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE 属性表示列的常量值。用于 `describe_column_get` 情形。

数据类型
`an_extfn_value`

描述
如果列在语句的生存期内保持不变，则为列值。如果该列的 `EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT` 返回为“true”，则该值可用。该属性仅对表参数有效。

用法
对于输入表中包含常量值的列，返回该值。如果该值不可用，则返回 NULL。

返回
如果成功，则返回 `sizeof(a_sql_byte)`，前提是返回值，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 无法获取属性。列不包含在查询中或值未视为常量时返回此错误。

失败时会返回某个通用 `describe_column` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_sql_byte` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定参数不是输入表时返回此 get 错误。

查询处理阶段
在以下状态下有效：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER (Get)

EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER 属性指示结果表中的某列是否为用户所使用。用于 `describe_column_get` 情形。

数据类型
`a_sql_byte`

描述
用于确定消耗程序是否用到结果表中的某个列，或用于表明不需要输入表中的某个列。对于表参数有效。允许用户设置或检索关于单列的信息，而类似的属性 `EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS` 则设置或检索关于一次调用中涉及的所有列的信息。

用法
UDF 查询该属性，以确定结果表中的某列是否为用户所需。这有助于 UDF 避免对未使用的列执行不必要的操作。

返回
如果成功，则返回 `sizeof(a_sql_byte)` 或：

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – 无法获取属性时返回。列不包含在查询中时可能发生此情况。

失败时会返回某个通用 describe_column 错误，或者：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 描述缓冲区大小不是 a_v4_extfn_estimate 时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 指定的参数为参数 0 时返回此 get 错误。

查询处理阶段
有效于：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

在 _describe_extfn API 函数中的过程定义及代码段：

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2
INT ) )
      RESULTS ( r1 INT, r2 INT, r3 INT )
      EXTERNAL ‘my_tpf_proc@mylibrary’ ;

CREATE TABLE T( x INT, y INT, z INT );

select r2,r3 from my_tpf( 'test', TABLE( select x,y from T ) )
```

当此 TPF 运行时，了解用户是否已选择结果集的列 r1 将十分有用。如果用户不需要 r1，则无需对其进行计算，也无需针对服务器生成 r1。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state > EXTFNAPIV4_STATE_INITIAL ) {
      a_sql_byte  col_is_used = 0;
      a_sql_int32 ret = 0;

      ret = cntxt->describe_column_get( cntxt, 0, 1,
            EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
            &col_is_used,
            sizeof(a_sql_byte) );

      if( ret < 0 ) {
          // Handle the error.
      }

    }
```

EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE (Get)
EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE 属性指示列的最小值。用于 `describe_column_get` 情形。

数据类型
`an_extfn_value`

描述
如果可用，则为列的最小值。仅对参数 0 和表参数有效。

用法
如果 UDF 获得 **EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE** 属性，则列数据的最小值通过 **describe_buffer** 予以返回。如果输入表为基表，则基于表中的所有列数据产生最小值，并且仅当表列具有索引时才可以访问该最大值。如果输入表为其他 UDF 的结果，则最小值为由该 UDF 所设置的 `EXTFNAPIV4_DESCRIBE_COL_TYPE`。

该属性的数据类型对于不同的列是不同的。UDF 可以使用 `EXTFNAPIV4_DESCRIBE_COL_TYPE` 确定列的数据类型。UDF 也可以根据 `EXTFNAPIV4_DESCRIBE_COL_WIDTH` 确定列的存储需求，以便提供相应大小的缓冲区保存该值。

describe_buffer_length 允许服务器确定缓冲区是否有效。

如果 **EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE** 属性不可用，则 `describe_buffer` 为 NULL。

返回
如果成功，则返回 `describe_buffer_length`，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 无法获取属性时返回。列不包含在查询中，或所请求列的最小值不可用时将返回此错误。

失败时会返回某个通用 `describe_column` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区的大小不足以容纳最小值时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状态不大于“初始”阶段时返回此 get 错误。

查询处理状态
在除初始状态外的任何状态中皆有效：

- 标注状态
- 查询优化状态
- 计划构建状态

- 执行状态

示例
_describe_extfn API 函数中的过程定义及代码段：

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
    RESULTS ( r1 INT, r2 INT, r3 INT )
    EXTERNAL  'my_tpf_proc@mylibrary' ;

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

本示例描述了 TPF 如何获取输入表中两列的最小值以供内部优化使用。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
      if( cntxt->current_state > EXTFNAPIV4_STATE_INITIAL ) {
           a_sql_int32 min_value = 0;
           a_sql_int32 ret = 0;

        // Get the minimum value of the second column of the
        // table input parameter 'col_table'

        ret = cntxt->describe_column_get( cntxt, 2, 2
                 EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
                 &min_value,
                 sizeof(a_sql_int32) );

     if( ret < 0 ) {
               // Handle the error.
        }

      }
}
```

EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE (Get)
EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE 属性指示列的最大值。用于 describe_column_get 情形。

数据类型
an_extfn_value

描述
列的最大值。该属性仅对参数 0 和表参数有效。

用法
如果 UDF 获取 EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE 属性，则 **describe_buffer** 中将返回列数据的最大值。如果输入表为基表，则最大值将基于表中

所有列数据且仅当表列上存在索引时可访问。如果输入表为另一 UDF 的结果，则最大值为该 UDF 设置的 COL_MAXIMUM_VALUE。

该属性的数据类型对于不同的列是不同的。UDF 可以使用 EXTFNAPIV4_DESCRIBE_COL_TYPE 确定列的数据类型。UDF 也可以根据 EXTFNAPIV4_DESCRIBE_COL_WIDTH 确定列的存储需求，以便提供相应大小的缓冲区保存该值。

describe_buffer_length 允许服务器确定缓冲区是否有效。

如果 EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE 不可用，则 describe_buffer 为 NULL。

返回
如果成功，则返回 describe_buffer_length 或：

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – 无法获取属性时返回。列不包含在查询中，或所请求列的最大值不可用时将发生此情况。

失败时会返回某个通用 describe_column 错误，或者：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 描述缓冲区的大小不足以容纳最大值时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理阶段不大于“初始”阶段时返回此 get 错误。

查询处理阶段
在除“初始”阶段外的其它阶段均有效：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

示例
_describe_extfn API 函数中的 **PROCEDURE** 定义和代码段：

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
         RESULTS ( r1 INT, r2 INT, r3 INT )
         EXTERNAL ‘my_tpf_proc@mylibrary’ ;

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

本示例描述了 TPF 如何获取输入表中两列的最大值以供内部优化使用。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
```

```
    if( cntxt->current_state > EXTFNAPIV4_STATE_INITIAL ) {
         a_sql_int32 max_value = 0;
         a_sql_int32 ret = 0;

         // Get the maximum value of the second column of the
         // table input parameter  'col_table'
         ret = cntxt->describe_column_get( cntxt, 2, 2
                  EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
                  &max_value,
                  sizeof(a_sql_int32) );

         if( ret < 0 ) {
             // Handle the error.
            }
       }
}
```

EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT (Get)
EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT 属性为输入行中指定的值设置子集。在 `describe_column_get` 情形中使用该属性将返回错误。

数据类型
`a_v4_extfn_col_subset_of_input`

描述
列值是输入列中所指定的值的子集。

用法
仅可对此属性进行设置。

返回
返回错误 `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE`。

查询处理状态
任何状态下都将返回错误 `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE`。

***describe_column_set**
`describe_column_set` 第 4 版 API 方法可在服务器中设置 UDF 列级属性。

说明
列级属性用于描述结果集或 TPF 输入表中的列的各种特性。例如，UDF 可告知服务器其结果集中的一列仅会有十个非重复值。

声明

```
a_sql_int32 (SQL_CALLBACK *describe_column_set)(
    a_v4_extfn_proc_context        *cntxt,
    a_sql_uint32        arg_num,
    a_sql_uint32        column_num,
```

```
    a_v4_extfn_describe_udf_type        describe_type,
    const void        *describe_buffer,
    size_t        describe_buffer_len );
```

参数

参数	描述
cntxt	此 UDF 的过程上下文对象。
arg_num	TABLE 参数的序号（0 为结果表，1 表示首个输入参数）。
column_num	列的序数从 1 开始。
describe_type	指示要设置的属性的选择器。
describe_buffer	对于指定的要在服务器中设置的属性，是用于存储描述信息的结构。具体结构或数据类型由 **describe_type** 参数表示。
describe_buffer_length	**describe_buffer** 的字节长度。

返回

成功时会返回已写入 **describe_buffer** 的字节数。如果出错或未检索到属性，则此函数会返回某个通用 `describe_column` 错误。

***describe_column_set** 的属性*

以下代码中有 `describe_column_set` 属性。

```
typedef enum a_v4_extfn_describe_col_type {
    EXTFNAPIV4_DESCRIBE_COL_NAME,
    EXTFNAPIV4_DESCRIBE_COL_TYPE,
    EXTFNAPIV4_DESCRIBE_COL_WIDTH,
    EXTFNAPIV4_DESCRIBE_COL_SCALE,
    EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
    EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
    } a_v4_extfn_describe_col_type;
```

EXTFNAPIV4_DESCRIBE_COL_NAME (Set)
EXTFNAPIV4_DESCRIBE_COL_NAME 属性指示列名。用在 `describe_column_set` 情形中。

数据类型
`char[]`

描述
列名。该属性仅对表参数有效。

用法
对于参数 0，如果 UDF 设置此属性，则服务器会比较该值与 **CREATE PROCEDURE** 语句中所提供的列名。这种比较可以确保 **CREATE PROCEDURE** 语句的列名和 UDF 预期列名相同。

返回
成功时会返回列名长度。

如果失败，则返回通用 `describe_column` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状态不为“标注”时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – 参数不为 TABLE 参数时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_ INVALID_ATTRIBUTE_VALUE` – 输入列名称长度超过 128 个字符或输入列名称与存储于目录中的列名不匹配时返回此 set 错误。

查询处理状态

- 标注状态

示例

```
short desc_rc = 0;
     char name[7] = ‘column1’;
     // Verify that the procedure was created with the second column
of the result table as an int
     if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
          desc_rc = ctx->describe_column_set( ctx, 0, 2,
EXTFNAPIV4_DESCRIBE_COL_NAME,
                                                                  name,
                                                          sizeof(name) );
          if( desc_rc < 0 ) {
          // handle the error.
          }
     }
```

EXTFNAPIV4_DESCRIBE_COL_TYPE (Set)
EXTFNAPIV4_DESCRIBE_COL_TYPE 属性指示列的数据类型。用于 describe_column_set 场景。

数据类型
a_sql_data_type

描述
列的数据类型。该属性仅对表参数有效。

用法
对于参数零，如果 UDF 设置此属性，则服务器会比较该值与 **CREATE PROCEDURE** 语句中所提供的列数据类型。UDF 借此可确保 **CREATE PROCEDURE** 语句的数据类型与 UDF 预期数据类型相同。

返回
如果成功，则返回 a_sql_data_type。

如果失败，则返回通用 describe_column 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 描述缓冲区大小不是 a_sql_data_type 时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状态不为“标注”时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_ INVALID_ATTRIBUTE_VALUE – 输入数据类型与存储于目录中的数据类型不匹配时返回此 set 错误。

查询处理状态

- 标注状态

示例

```
short desc_rc = 0;
a_sql_data_type type = DT_INT;

   // Verify that the procedure was created with the second column of
the result table as an int
    if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
        desc_rc = ctx->describe_column_set( ctx, 0, 2,
EXTFNAPIV4_DESCRIBE_COL_TYPE,
        &type,
                sizeof(a_sql_data_type) );
        if( desc_rc < 0 ) {
            // handle the error.
        }
    }
```

EXTFNAPIV4_DESCRIBE_COL_WIDTH (Set)
EXTFNAPIV4_DESCRIBE_COL_WIDTH 属性指示列的宽度。用在 `describe_column_set` 情形中。

数据类型
`a_sql_uint32`

描述
列宽。列宽是以字节为单位的存储空间量，用于存储关联数据类型的值。该属性仅对表参数有效。

用法
如果 UDF 设置此属性，则服务器会比较该值与 **CREATE PROCEDURE** 语句中所提供的列的宽度。UDF 借此可确保 **CREATE PROCEDURE** 语句的列宽与 UDF 预期列宽相同。

返回
如果成功，则返回 `sizeof(a_sql_uint32)`。

如果失败，则返回通用 `describe_column` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_sql_uint32` 时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理状态不为“标注”时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – 输入宽度与存储于目录中的宽度不匹配时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 标注状态

EXTFNAPIV4_DESCRIBE_COL_SCALE (Set)
EXTFNAPIV4_DESCRIBE_COL_SCALE 属性指示列的标度。用在 `describe_column_set` 情形中。

数据类型
`a_sql_uint32`

描述
列的标度。对于算术型数据类型，参数标度是指数字的小数点右边的位数。该属性仅对表参数有效。

用法
如果 UDF 设置此属性，则服务器会比较该值与 **CREATE PROCEDURE** 语句中所提供的列的标度。UDF 借此可确保 **CREATE PROCEDURE** 语句的列宽与 UDF 预期列宽相同。此属性仅对算术数据类型有效。

返回
如果成功，则返回 `sizeof(a_sql_uint32)`，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 指定列的数据类型的标度不可用时返回此 set 错误。

如果失败，则返回通用 `describe_column` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_sql_uint32` 时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理状态不为“标注”时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – 输入标度与存储于目录中的标度不匹配时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 标注状态

示例

```
    short desc_rc = 0;
    a_sql_uint32 scale = 0;

            // Verify that the procedure has a scale of zero for the
second result table column.
        if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
        desc_rc = ctx->describe_column_set( ctx, 0, 2,
EXTFNAPIV4_DESCRIBE_COL_SCALE,
            &scale,
            sizeof(a_sql_data_type) );
        if( desc_rc < 0 ) {
            // handle the error.
        }
    }
```

EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL (Set)

EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL 属性指示列是否可以为空。用在 `describe_column_set` 情形中。

数据类型
`a_sql_byte`

描述
如果列可以是 Null，则该值为“true”。该属性仅对表参数有效。该属性仅对参数 0 有效。

用法
对于结果表列，如果可以是空列，则 UDF 可设置此属性。如果 UDF 不特意设置此属性，则会假定列可以是空列。服务器可在优化状态下使用这些信息。

返回
如果成功，则返回 `sizeof(a_sql_byte)`，前提是已设置属性或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 无法设置属性时返回，列不包含在查询中时可能发生此情况。

如果失败，则返回通用 `describe_column` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_sql_byte` 时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状态不等于“优化”时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF 尝试将此属性设置为 0 或 1 之外的其它值时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 优化状态

EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES (Set)
EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES 属性用于描述列的非重复值。用在 `describe_column_set` 情形中。

数据类型
`a_v4_extfn_estimate`

描述
某一列的估计离散值数量。该属性仅对表参数有效。

用法
如果 UDF 知道其结果表中的一列能有多少非重复值，则 UDF 可设置此属性。服务器可在优化状态下使用这些信息。

返回
如果成功，则返回 `sizeof(a_v4_extfn_estimate)`，前提是它设置值，或：

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – 无法设置属性时返回。列不包含在查询中时可能发生此情况。

如果失败，则返回：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 描述缓冲区大小不是 a_v4_extfn_estimate 时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状态不等于“优化”时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 优化状态

EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE (Set)

EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE 属性指示列在表中是否为唯一的列。用在 describe_column_set 情形中。

数据类型
a_sql_byte

描述
如果表中的列是唯一的，则该值为“true”。该属性仅对表参数有效。

用法
如果 UDF 知道结果表列值是唯一的值，则 UDF 可设置此属性。服务器可在优化状态下使用这些信息。UDF 仅可对参数 0 设置此属性。

返回
如果成功，则返回 sizeof(a_sql_byte) 或：

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – 无法设置属性时返回。列不包含在查询中时可能发生此情况。

如果失败，则返回通用 describe_column 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 描述缓冲区大小不是 a_sql_byte 时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理状态不为“优化”时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – **arg_num** 不为零时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF 尝试将此属性设置为 0 或 1 之外的其它值时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 优化状态

EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT (Set)
EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT 属性指示列是否为常量。用在 `describe_column_set` 情形中。

数据类型
`a_sql_byte`

描述
如果该列在语句的生存期内保持不变，则为“true”。该属性仅对输入的表参数有效。

用法
这是只读属性。所有的设置尝试都将返回 `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE`。

返回

- `EXTFNAPIV4_DESCRIBE_INVALID ATTRIBUTE` – 这是只读属性；所有的设置尝试都将返回此错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状态不为“优化”时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – **arg_num** 不为零时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF 尝试将此属性设置为 0 或 1 之外的其它值时返回此 set 错误。

查询处理状态
不适用。

EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE (Set)
EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE 属性指示列的常量值。用在 `describe_column_set` 情形中。

数据类型
`an_extfn_value`

描述
如果列在语句的生存期内保持不变，则为列值。如果该列的 `EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT` 返回为“true”，则该值可用。该属性仅对表参数有效。

用法
此属性是只读属性。

返回

- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` – 这是只读属性；所有的设置尝试都将返回此错误。

查询处理状态
不适用。

EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER (Set)

EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER 属性指示消耗程序是否使用结果表中的列。用在 `describe_column_set` 情形中。

数据类型
`a_sql_byte`

描述
用于确定消耗程序是否用到结果表中的某个列，或用于表明不需要输入表中的某个列。对于表参数有效。允许用户设置或检索关于单列的信息，而类似的属性 `EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS` 则设置或检索关于一次调用中涉及的所有列的信息。

用法
UDF 会对输入表中的列设置 **EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER**，以告知生产者 UDF 不需要该列的值。

返回
如果成功，则返回 `sizeof(a_sql_byte)` 或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 无法设置属性时返回。列不包含在查询中时可能发生此情况。

失败时会返回某个通用 `describe_column` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_v4_extfn_estimate` 时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定的参数为参数 0 时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状态不等于“优化”时返回此 set 错误。

- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF 设置的值不为 0 或 1 时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 优化状态

_describe_extfn API 函数中的 PROCEDURE 定义和代码段：

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2
INT ) )
      RESULTS ( r1 INT, r2 INT, r3 INT )
      EXTERNAL 'my_tpf_proc@mylibrary' ;

CREATE TABLE T( x INT, y INT, z INT );

select r2,r3 from my_tpf( 'test', TABLE( select x,y from T ) )
```

当此 TPF 运行时，如果服务器了解此 TPF 是否使用了列 y 将十分有用。如果 TPF 不需要 y，则服务器可使用此信息进行优化，而不将此列信息发送到 TPF。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
      a_sql_byte  col_is_used = 0;
      a_sql_int32 ret = 0;

      ret = cntxt->describe_column_get( cntxt, 2, 2,
            EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
            &col_is_used,
            sizeof(a_sql_byte) );

      if( ret < 0 ) {
          // Handle the error.
      }

    }
}
```

EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE (Set)
EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE 属性用于对列指定最小值。用在 describe_column_set 情形中。

数据类型
an_extfn_value

描述
是列能有的最小值（如果有）。仅对参数 0 有效。

用法
如果 UDF 知道列的最小数据值是什么，则 UDF 可设置 EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE。服务器可在优化过程中使用这些信息。

UDF 可用 EXTFNAPIV4_DESCRIBE_COL_TYPE 确定列的数据类型，也可用 EXTFNAPIV4_DESCRIBE_COL_WIDTH 确定列的存储要求，以便提供大小调整得相当的缓冲区，以存储要设置的值。

返回
成功时会返回 describe_buffer_length，或者：

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE — 如果该属性不能设置。如果查询不涉及该列，或者所请求的列未提供最小值，则返回。

失败时会返回某个通用 describe_column 错误，或者：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH — 如果描述缓冲区不够大，不能存储最小值，则会返回设置错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE — 如果状态不是优化状态，则会返回设置错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER — 如果 **arg_num** 不是 0，则会返回设置错误。

查询处理状态
在以下状态下有效：

- 优化状态

示例
用于实现 _describe_extfn 回调 API 函数的 **PROCEDURE** 定义和 UDF 代码段：

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
    RESULTS ( r1 INT, r2 INT, r3 INT )
    EXTERNAL ‘my_tpf_proc@mylibrary’;

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

此例所示的是可从中将其用于服务器（或用于另一可接收此 TPF 的结果， 将其用作输入的 TPF）的 TPF，以了解第一个结果集列的最小值。 在此例中，第一列的最小输出值是 27。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
    a_sql_int32 min_value = 27;
```

```
    a_sql_int32 ret = 0;

// Tell the server what the minimum value of the first column
// of our result set will be.

    ret = cntxt->describe_column_set( cntxt, 0, 1
        EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
        &min_value,
        sizeof(a_sql_int32) );

    if( ret < 0 ) {
        // Handle the error.
    }
}
}
```

EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE (Set)
EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE 属性用于对列指定最大值。用在 `describe_column_set` 情形中。

数据类型
`an_extfn_value`

描述
列的最大值。该属性仅对参数 0 和表参数有效。

用法
如果 UDF 知道列的最大数据值是什么，则 UDF 可设置 `EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE`。服务器可在优化过程中使用这些信息。

UDF 可用 `EXTFNAPIV4_DESCRIBE_COL_TYPE` 确定列的数据类型，也可用 `EXTFNAPIV4_DESCRIBE_COL_WIDTH` 确定列的存储要求，以便提供大小调整得相当的缓冲区，以存储要设置的值。

`describe_buffer_length` 是此缓冲区的 `sizeof()`。

返回
如果成功，则返回 `describe_buffer_length`，前提是已设置值，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 无法设置属性时返回。列不包含在查询中，或所请求列的最大值不可用时将返回此错误。

失败时会返回某个通用 `describe_column` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区的大小不足以容纳最大值时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理状态不等于“优化”时返回此 set 错误。

- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – **arg_num** 不为 0 时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 优化状态

示例
用于实现 _describe_extfn 回调 API 函数的 PROCEDURE 定义和 UDF 代码段：

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
         RESULTS ( r1 INT, r2 INT, r3 INT )
         EXTERNAL ‘my_tpf_proc@mylibrary’ ;

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

此例所示的是可从中将其用于服务器（或用于另一可接收此 TPF 的结果，将其用作输入的 TPF）的 TPF，以了解第一个结果集列的最大值。在此例中，第一列的最大输出值是 500000。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
         a_sql_int32 max_value = 500000;
         a_sql_int32 ret = 0;

        // Tell the server what the maximum value of the first column
         // of our result set will be.

      ret = cntxt->describe_column_set( cntxt, 0, 1
                  EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
                  &max_value,
                  sizeof(a_sql_int32) );

         if( ret < 0 ) {
             // Handle the error.
         }

   }
}
```

EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT (Set)
EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT 属性为输入行中指定的值设置子集。用于 describe_column_set 情形。

数据类型
a_v4_extfn_col_subset_of_input

描述
列值是输入列中所指定的值的子集。

用法
设置此描述属性将通知查询优化程序，所指列值是输入列中指定值的子集。例如，考虑一个 TPF 过滤器，该 TPF 消耗表资源，并且基于函数对行进行过滤。在此种情况下，返回表为输入表的子集。为 TPF 过滤器设置 **EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT** 属性以优化查询。

返回
如果成功，则返回 `sizeof(a_v4_extfn_col_subset_of_input)`。

失败时会返回某个通用 `describe_column` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 缓冲区长度小于 `sizeof (a_v4_extfn_col_subset_of_input)` 时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – 源表的列索引超出范围时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 设置了 `subset_of_input` 的列不适用（例如，该列不在选择列表中）时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVAILD_STATE` – 查询处理状态不为“优化”时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 缓冲区长度为零时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 调用参数而非返回表时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 优化状态

示例

```
a_v4_extfn_col_subset_of_input colMap;

    colMap.source_table_parameter_arg_num = 4;
    colMap.source_column_number = i;

    desc_rc = ctx->describe_column_set( ctx,
        0, i,
        EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
        &colMap, sizeof(a_v4_extfn_col_subset_of_input) );
```

*describe_parameter_get

describe_parameter_get 第 4 版 API 方法可从服务器中获取 UDF 参数属性。

声明

```
a_sql_int32 (SQL_CALLBACK *describe_parameter_get)(
            a_v4_extfn_proc_context             *cntxt,
            a_sql_uint32                        arg_num,
            a_v4_extfn_describe_udf_type        describe_type,
            const void                          *describe_buffer,
           size_t                              describe_buffer_len );
```

参数

参数	描述
cntxt	过程上下文对象。
arg_num	TABLE 参数的序号（0 为结果表，1 表示首个输入参数）。
describe_type	指示要设置的属性的选择器。
describe_buffer	对于指定的要在服务器中设置的属性，是用于存储描述信息的结构。具体结构或数据类型由 **describe_type** 参数表示。
describe_buffer_length	**describe_buffer** 的字节长度。

返回

成功时会返回 0 或已写入 **describe_buffer** 的字节数。值为 0 表明服务器不能获取属性，但没出错。如果出错或未检索到属性，则此函数会返回某个通用 **describe_parameter** 错误。

describe_parameter_get 的属性

以下代码中有 describe_parameter_get 属性。

```
typedef enum a_v4_extfn_describe_parm_type {
    EXTFNAPIV4_DESCRIBE_PARM_NAME,
    EXTFNAPIV4_DESCRIBE_PARM_TYPE,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,

    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
```

```
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS,

} a_v4_extfn_describe_parm_type;
```

EXTFNAPIV4_DESCRIBE_PARM_NAME 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_NAME 属性指示参数名。用于 describe_parameter_get 场景。

数据类型
char[]

描述
UDF 的参数名称。

用法
用于获取 **CREATE PROCEDURE** 语句中所定义的参数名称。对于参数 0 无效。

返回
成功时会返回参数名称长度。

如果失败，则返回通用 describe_parameter 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 的大小不足以保存名称时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 参数为结果表时返回此 get 错误。

查询处理阶段
在以下状态下有效：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_PARM_TYPE 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_TYPE 属性在 describe_parameter_get 情形下用于返回数据类型。

数据类型
a_sql_data_type

描述
UDF 的参数数据类型。

用法
用于获取 **CREATE PROCEDURE** 语句中所定义的参数数据类型。

返回
如果成功，则返回 `sizeof(a_sql_data_type)`。

失败时会返回某个通用 `describe_parameter` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** 不是 `sizeof(a_sql_data_type)` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“初始”阶段时返回此 get 错误。

查询处理阶段
在以下状态下有效：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_PARM_WIDTH 属性 (Get)

EXTFNAPIV4_DESCRIBE_PARM_WIDTH 属性指示参数的宽度。用在 `describe_parameter_get` 情形中。

数据类型
`a_sql_uint32`

描述
UDF 的参数宽度。`EXTFNAPIV4_DESCRIBE_PARM_WIDTH` 仅适用于标量参数。参数宽度是以字节为单位的存储空间量，用于存储关联数据类型的参数。

- **定长数据类型** – 存储数据所需的字节。
- **变长数据类型** – 长度上限。
- **LOB 数据类型** – 将句柄存入数据所需的存储空间量。
- **TIME 数据类型** – 存储经过编码的时间所需的存储空间量。

用法
用于获取 **CREATE PROCEDURE** 语句中所定义的参数宽度。

返回
如果成功，则返回 `sizeof(a_sql_uint32)`。

如果失败，则返回通用 `describe_parameter` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** 大小不是 `a_sql_uint32` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定参数为 TABLE 参数时返回此 get 错误。这包括参数 0 或参数 *n*（*n* 为输入表）。

查询处理阶段
有效于：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

示例
示例过程定义：

```
CREATE PROCEDURE my_udf(IN p1 INT, IN p2 char(100))
RESULT (x INT)
EXTERNAL NAME ‘my_udf@myudflib’;
```

示例 `_describe_extfn` API 函数代码段：

```
my_udf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
    a_sql_uint32 width = 0;
    a_sql_int32 ret = 0;

    // Get the width of parameter 1
    ret = cntxt->describe_parameter_get( cntxt, 1,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    &width,
    sizeof(a_sql_uint32) );

    if( ret < 0 ) {
        // Handle the error.
    }

    //Allocate some storage based on parameter width
    a_sql_byte *p = (a_sql_byte *)cntxt->alloc( cntxt, width )

    // Get the width of parameter 2
    ret = cntxt->describe_parameter_get( cntxt, 2,
        EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
```

```
        &width,
        sizeof(a_sql_uint32) );
    if( ret <= 0 ) {
        // Handle the error.
    }

    // Allocate some storage based on parameter width
    char *c = (char *)cntxt->alloc( cntxt, width )

    …

}
}
```

EXTFNAPIV4_DESCRIBE_PARM_SCALE 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_SCALE 属性指示参数的标度。用在 `describe_parameter_get` 情形中。

数据类型
`a_sql_uint32`

描述
UDF 的参数标度。对于算术型数据类型，参数标度是指数字的小数点右边的位数。

此属性对于以下数据无效：

- 非算术数据类型
- TABLE 参数

用法
用于获取 **CREATE PROCEDURE** 语句中所定义的参数标度。

返回
如果成功，则返回 `(a_sql_uint32)` 的大小。

如果失败，则返回通用 `describe_parameter` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** 大小不是 `a_sql_uint32` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定参数为 TABLE 参数时返回此 get 错误。这包括参数 0 或参数 *n*（*n* 为输入表）。

查询处理阶段
有效于：

- 标注阶段

- 查询优化阶段
- 计划构建阶段
- 执行阶段

示例
用于获取参数 1 的标度的示例 `_describe_extfn` API 函数代码段：

```
if( cntxt->current_state > EXTFNAPIV4_STATE_ANNOTATION ) {
    a_sql_uint32 scale = 0;
    a_sql_int32 ret = 0;

    ret = ctx->describe_parameter_get( ctx, 1,
EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    &scale, sizeof(a_sql_uint32) );

    if( ret <= 0 ) {
    // Handle the error.
    }
}
```

EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL 属性指示参数是否为空。用在 `describe_parameter_get` 情形中。

数据类型
`a_sql_byte`

描述
如果执行时参数值可为 NULL，则为 true。对于 TABLE 参数或参数 0，值为 false。

用法
获取查询执行期间指定的参数是否可以为 NULL。

返回
如果成功，则返回 `sizeof(a_sql_byte)`。

如果失败，则返回通用 `describe_parameter` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** 大小不是 `a_sql_byte` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“计划构建”阶段时返回此 get 错误。

查询处理阶段
有效于：

- 执行阶段

示例：EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL (Get)
示例过程定义，_describe_extfn API 函数代码段，以及用于获取 **EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL** 值的 SQL 查询。

过程定义
本主题中示例查询所用的示例过程定义：

```
CREATE PROCEDURE my_udf(IN p INT)
RESULT (x INT)
EXTERNAL NAME ‘my_udf@myudflib’;
```

API 函数代码段
本主题中示例查询所用的示例 _describe_extfn API 函数代码段：

```
my_udf_describe(a_v4_extfn_proc_context *cntxt)
{

    if( cntxt->current_state > EXTFNAPIV4_STATE_OPTIMIZATION ) {
    a_sql_byte can_be_null = 0;
    a_sql_int32 ret = 0;

    ret = cntxt->describe_parameter_get( cntxt, 1,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    &can_be_null,
    sizeof(a_sql_byte) );

    if( ret <= 0 ) {
        // Handle the error.
    }
}
}
```

示例 1：未使用 NOT NULL
本示例所创建的表包含单个整数列，且没有指定 **NOT NULL** 修饰符。相关子查询传入 has_nulls 表中的 *c1* 列。当在执行状态中调用过程 **my_udf_describe** 时，对 describe_parameter_get 的调用将为 **can_be_null** 赋值 1。

```
CREATE TABLE has_nulls ( c1 INT );
INSERT INTO has_nulls VALUES(1);
INSERT INTO has_nulls VALUES(NULL);
SELECT * from has_nulls WHERE (SELECT sum(my_udf.x) FROM
my_udf(has_nulls.c1)) > 0;
```

示例 2：使用 NOT NULL
本示例所创建的表包含单个整数列，且没有指定 **NOT NULL** 修饰符。相关子查询传入 no_nulls 表中的 *c1* 列。当在执行状态中调用过程 **my_udf_describe** 时，对 describe_parameter_get 的调用将为 **can_be_null** 赋值 0。

```
CREATE TABLE no_nulls ( c1 INT NOT NULL);
INSERT INTO no_nulls VALUES(1);
INSERT INTO no_nulls VALUES(2);
```

```
SELECT * from no_nulls WHERE (SELECT sum(my_udf.x) FROM
my_udf(no_nulls.c1)) > 0;
```

示例 3：使用常量
本示例使用常量调用过程 **my_udf**。当在执行状态中调用过程 **my_udf_describe** 时，对 `describe_parameter_get` 的调用将为 **can_be_null** 赋值 0。

```
SELECT * from my_udf(5);
```

示例 4：使用 NULL
本示例使用 NULL 调用过程 **my_udf**。当在执行状态中调用过程 **my_udf_describe** 时，对 `describe_parameter_get` 的调用将为 **can_be_null** 赋值 1。

```
SELECT * from my_udf(NULL);
```

EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性用于返回非重复值数量。用在 `describe_parameter_get` 情形中。

数据类型
`a_v4_extfn_estimate`

描述
返回所有调用中的估计离散值数量。仅对标量参数有效。

用法
如果有这些信息，则 UDF 会返回估计的非重复值数量，可信度为 100%。如果没有这些信息，则 UDF 会返回估计的数量 0，可信度为 0%。

返回
如果成功，则返回 `sizeof(a_v4_extfn_estimate)`。

如果失败，则返回通用 `describe_parameter` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** 大小不是 `a_v4_extfn_estimate` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 参数为 TABLE 参数时返回此 get 错误。

查询处理阶段
有效于：

- 标注阶段
- 查询优化阶段

- 计划构建阶段
- 执行阶段

示例
示例 _describe_extfn API 函数代码段：

```
if( ctx->current_state >= EXTFNAPIV4_STATE_ANNOTATION ) {
    desc_est.value = 0.0;
    desc_est.confidence = 0.0;

    desc_rc = ctx->describe_parameter_get( ctx,
        1,
        EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
        &desc_est, sizeof(a_v4_extfn_estimate) );
}
```

EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT 属性 (Get)
无论参数是否为常量，都将返回 EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性。用于 describe_parameter_get 场景。

数据类型
a_sql_byte

描述
如果语句的参数为常量，则该值为“true”。仅对标量参数有效。

用法
如果指定的参数的值不是常量，则会返回 0；如果指定的参数的值是常量，则会返回 1。

返回
如果成功，则返回 sizeof(a_sql_byte)。

如果失败，则返回通用 describe_parameter 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 大小不是 a_sql_byte 时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 参数为 TABLE 参数时返回此 get 错误。

查询处理阶段
有效于：

- 标注阶段

- 查询优化阶段
- 计划构建阶段
- 执行阶段

示例
示例 `_describe_extfn` API 函数代码段：

```
if( ctx->current_state >= EXTFNAPIV4_STATE_ANNOTATION ) {
    desc_rc = ctx->describe_parameter_get( ctx,
        1,
        EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
        &desc_byte, sizeof( a_sql_byte ) );
}
```

EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE 属性指示参数值。用在 `describe_parameter_get` 情形中。

数据类型
`an_extfn_value`

描述
如果在描述时是已知的，则为参数值。仅对标量参数有效。

用法
用于返回参数值。

返回
如果成功，则返回 `sizeof(an_extfn_value)`，前提是值可用，或：

- `EXTFNAPIV4_DESCRIBE_NOT_AVILABLE` – 值不为常量时返回此值。

如果失败，则返回通用 `describe_parameter` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** 大小不是 `an_extfn_value` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 阶段不大于“初始”阶段时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 参数为 TABLE 参数时返回此 get 错误。

查询处理阶段
有效于：

- 标注阶段
- 查询优化阶段

- 计划构建阶段
- 执行阶段

示例
示例 _describe_extfn API 函数代码段:

```
if( ctx->current_state >= EXTFNAPIV4_STATE_ANNOTATION ) {
    a_sql_int32 desc_rc;
    desc_rc = ctx->describe_parameter_get( ctx,
        1,
        EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,
        &arg,
        sizeof( an_extfn_value ) );
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS 属性指示表中的列数。用于 describe_parameter_get 场景。

数据类型
a_sql_uint32

描述
表中的列数。仅对参数 0 和表参数有效。

用法
返回指定的表参数中的列数。参数 0 用于返回结果表中的列数。

返回
如果成功，则返回 sizeof(a_sql_uint32)。

如果失败，则返回通用 describe_parameter 错误之一，或:

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 大小不是 size of a_sql_uint32 时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – 参数不为 TABLE 参数时返回此 get 错误。

查询处理阶段
有效于:

- 标注阶段
- 查询优化阶段
- 计划构建阶段

- 执行阶段

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS 属性指示表中的行数。用于 describe_parameter_get 场景。

数据类型
a_v4_extfn_estimate

描述
表中的估计行数。仅对参数 0 和表参数有效。

用法
用于在指定的表参数或结果集中返回估计的行数，可信度为 100%。

返回
如果成功，则返回 a_v4_extfn_estimate 的大小。

失败时会返回某个通用 describe_parameter 错误，或者：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 大小不是 a_v4_extfn_estimate 时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理阶段不大于“初始”阶段时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – 参数不为 TABLE 参数时返回此 get 错误。

查询处理阶段
有效于：

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY 属性指示表中的行的顺序。用于 describe_parameter_get 场景。

数据类型
a_v4_extfn_orderby_list

描述
表中各行的次序。该属性仅对参数 0 和表参数有效。

用法
通过此属性可用 UDF 代码：

- 确定输入 **TABLE** 参数是否已经过排序
- 声明结果集已经过排序

如果参数数量是 0，则该属性是指外发结果集。如果该参数 > 0，并且参数类型是表参数，则该属性是指输入 **TABLE** 参数。

顺序由 `a_v4_extfn_orderby_list` 指定，是支持列顺序号及其相关升序或降序属性的列表的一个结构。如果 UDF 将出站结果集设置为按属性排序，服务器则能够按优化执行排序。例如，如果 UDF 按升序生成结果集首列，则服务器将通过请求消除针对同一列的冗余排序。

如果 UDF 不对外发结果集设置 orderby 属性，则服务器会假定数据未经排序。

如果 UDF 对输入 **TABLE** 参数设置 orderby 属性，则服务器保证会对输入数据进行排序。在这种情况下，UDF 会向服务器作出输入数据必须经过排序的描述。如果服务器检测到运行时冲突，则会报告 SQL 异常。例如，如果 UDF 作出这样的描述，即输入 **TABLE** 参数第一列的顺序必须是升序，但 SQL 语句含有降序子句，则服务器会报告 SQL 异常。

如果 SQL 不含排序子句，则服务器会自动进行排序，以确保按要求对输入 **TABLE** 参数排序。

返回
如果成功，则返回从 `a_v4_extfn_orderby_list` 中复制的字节数。

查询处理状态
在以下状态下有效：

- 标注状态
- 查询优化状态

EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY (Get)

`EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY` 属性指示 UDF 需要进行分区。用于 `describe_parameter_get` 场景。

数据类型
`a_v4_extfn_column_list`

描述
UDF 开发者使用 **EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY** 以编程方式声明，调用之前须先对 UDF 进行分区。

用法
UDF 可以查询分区以强制执行，或者动态调整分区。UDF 负责分配 **a_v4_extfn_column_list**，需要考虑输入表的总列数，并将其发送至服务器。

返回
如果成功，则返回 a_v4_extfn_column_list 的大小。此值等于：

```
sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) *
number_of_partition_columns
```

失败时会返回某个通用 describe_parameter 错误，或者：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 缓冲区长度小于预期大小时返回此 get 错误。

查询处理阶段
有效于：

- 查询优化阶段
- 计划构建阶段
- 执行阶段

示例

```
void UDF_CALLBACK my_tpf_proc_describe( a_v4_extfn_proc_context
*ctx )
{
    if( ctx->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
        a_sql_uint32    col_count   = 0;
        a_sql_uint32    buffer_size = 0;
        a_v4_extfn_column_list    *clist = NULL;

        col_count = 3;    // Set to the max number of possible pby
columns

       buffer_size = sizeof( a_v4_extfn_column_list ) + (col_count -
1) * sizeof( a_sql_uint32 );

        clist = (a_v4_extfn_column_list *)ctx->alloc( ctx,
buffer_size );

        clist->number_of_columns = 0;
        clist->column_indexes[0] = 0;
        clist->column_indexes[1] = 0;
        clist->column_indexes[2] = 0;

    args->r_api_rc = ctx->describe_parameter_get( ctx,
        args->p3_arg_num,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    clist,
    buffer_size );
```

```
}
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND 属性 (Get)

EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND 属性指示消费者请求输入表回绕。用于 describe_parameter_get 场景。

数据类型
a_sql_byte

描述
表示消耗程序想要回绕输入表。仅对表输入参数有效。缺省情况下，该属性为“false”。

用法
UDF 会查询此属性以检索 true/false 值。

返回
如果成功，则返回 sizeof(a_sql_byte)。

如果失败，则返回通用 describe_parameter 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 大小不是 a_sql_byte 时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 阶段不为“优化”或“计划构建”时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – UDF 尝试从参数 0 获取此属性时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – UDF 尝试从非表的参数获取此属性时返回此 get 错误。

查询处理阶段
有效于：

- 优化阶段
- 计划构建阶段

EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND 属性 (Get)

EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND 属性指示参数是否支持回绕。用于 describe_parameter_get 场景。

数据类型
a_sql_byte

描述
表示生产者是否支持回绕。仅对表参数有效。

如果您计划将 DESCRIBE_PARM_TABLE_HAS_REWIND 设置为 true，则还必须实施回绕表回调 (**_rewind_extfn()**)。如果不提供回调方法，服务器将无法执行 UDF。

用法
UDF 会询问表输入参数是否支持回绕。UDF 必须先通过 **DESCRIBE_PARM_TABLE_REQUEST_REWIND** 请求回绕（这是一个前提条件），然后才能使用此属性。

返回
如果成功，则返回 sizeof(a_sql_byte)。

如果失败，则返回通用 describe_parameter 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 大小不是 a_sql_byte 时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理阶段不大于“标注”阶段时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – UDF 尝试从非表的参数获取此属性时返回此 get 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – UDF 尝试从结果表获取此属性时返回此 get 错误。

查询处理阶段
有效于：

- 优化阶段
- 计划构建阶段
- 执行阶段

EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS 属性 (Get)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS 属性列出未使用的列。用于 describe_parameter_get 场景。

数据类型
a_v4_extfn_column_list

描述
服务器或者 UDF 将不会使用的输出表列的列表。

对于输出 TABLE 参数，UDF 通常为所有列生成数据，而服务器将使用所有的列。输入 TABLE 参数亦是如此，其中服务器通常为所有列生成数据，而 UDF 将使用所有的列。

但是，某些情况下服务器或 UDF 可能不会消耗所有列。这种情况下，最佳做法是让 UDF 对描述属性 **EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS** 执行输出表 **GET**。此操作可查询服务器，以列出不会被服务器消耗的输出表列。然后 UDF

就可以在填充输出表列数据时使用列出的输出表列，也就是说，UDF 不会尝试对未使用的列填充数据。

总之，对于输出表，UDF 会轮询未使用的列的列表。对于输出表，UDF 会推送未使用的列的列表。

用法
UDF 会询问服务器是否要使用所有输出表列。UDF 必须分配包括所有输出表列的 `a_v4_extfn_column_list`，然后还必须将其传递给服务器。服务器随后将所有估计不会用到的列的序数标为 1。生成数据时可使用服务器返回的列表。

返回
如果成功，则返回列列表 `sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) * number result columns` 的大小。

失败时会返回某个通用 `describe_parameter` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 查询处理阶段不大于“计划构建”阶段时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** 的大小不足以保存返回的列表时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – UDF 尝试从输入表获取此属性时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – UDF 尝试从非表的参数获取此属性时返回此 get 错误。

查询处理阶段
有效于：

- 执行阶段

***describe_parameter_set**
`describe_parameter_set` 第 4 版 API 方法可以设置有关一个 UDF 参数的属性。

声明

```
a_sql_int32 (SQL_CALLBACK *describe_parameter_set)(
            a_v4_extfn_proc_context             *cntxt,
            a_sql_uint32                        arg_num,
            a_v4_extfn_describe_udf_type        describe_type,
            const void                          *describe_buffer,
           size_t                             describe_buffer_len );
```

参数

参数	描述
cntxt	过程上下文对象。
arg_num	TABLE 参数的序号（0 为结果表，1 表示首个输入参数）。
describe_type	指示要设置的属性的选择器。
describe_buffer	对于指定的要在服务器中设置的属性，是用于存储描述信息的结构。具体结构或数据类型由 **describe_type** 参数表示。
describe_buffer_length	**describe_buffer** 的字节长度。

返回

成功时会返回 0 或已写入 **describe_buffer** 的字节数。值为 0 表明服务器不能设置属性，但没出错。如果出错或未检索到属性，则此函数会返回某个通用 **describe_parameter** 错误。

**describe_parameter_set 的属性*

以下代码中有 `describe_parameter_set` 属性。

```
typedef enum a_v4_extfn_describe_parm_type {
    EXTFNAPIV4_DESCRIBE_PARM_NAME,
    EXTFNAPIV4_DESCRIBE_PARM_TYPE,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,

    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS,

    } a_v4_extfn_describe_parm_type;
```

EXTFNAPIV4_DESCRIBE_PARM_NAME 属性 (Set)

`EXTFNAPIV4_DESCRIBE_PARM_NAME` 属性指示参数名。用于 `describe_parameter_set` 场景。

数据类型

`char[]`

描述
UDF 的参数名称。

用法
如果 UDF 设置此属性，则服务器会比较该值与 **CREATE PROCEDURE** 语句中所提供的参数的名称。如果两个值不一致，则服务器会返回错误。UDF 借此可确保 **CREATE PROCEDURE** 语句的参数名称与 UDF 预期参数名称相同。

返回
成功时会返回参数名称长度。

如果失败，则返回通用 `describe_parameter` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状态不等于“标注”时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 参数为结果表时返回此 set 错误。
- `EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF 尝试重置名称时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 标注状态

EXTFNAPIV4_DESCRIBE_PARM_TYPE 属性 (Set)
`EXTFNAPIV4_DESCRIBE_PARM_TYPE` 属性指示参数的数据类型。用于 `describe_parameter_set` 场景。

数据类型
`a_sql_data_type`

描述
UDF 的参数数据类型。

用法
如果 UDF 设置此属性，则服务器会比较该值与 **CREATE PROCEDURE** 语句中所提供的参数类型。如果两个值不一致，则服务器会返回错误。这项检查可确保 **CREATE PROCEDURE** 语句的参数数据类型和 UDF 的预期参数数据类型相同。

返回
如果成功，则返回 `sizeof(a_sql_data_type)`。

失败时会返回某个通用 `describe_parameter` 错误，或者：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 不是 sizeof(a_sql_data_type) 时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理状态不等于“标注”时返回此 set 错误。
- EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF 尝试将参数设置为不是已定义的数据类型时返回此 set 错误。

查询处理状态
在以下状态下有效:

- 标注状态

EXTFNAPIV4_DESCRIBE_PARM_WIDTH 属性 (Set)

EXTFNAPIV4_DESCRIBE_PARM_WIDTH 属性指示参数的宽度。用在 describe_parameter_set 情形中。

数据类型
a_sql_uint32

描述
UDF 的参数宽度。EXTFNAPIV4_DESCRIBE_PARM_WIDTH 仅适用于标量参数。参数宽度是以字节为单位的存储空间量，用于存储关联数据类型的参数。

- **定长数据类型** – 存储数据所需的字节。
- **变长数据类型** – 最大长度。
- **LOB 数据类型** – 将句柄存入数据所需的存储空间量。
- **TIME 数据类型** – 存储经过编码的时间所需的存储空间量。

用法
这是一个只读属性。其宽度由相关联的列数据类型派生而来。数据类型一设置完毕，就不能更改宽度。

返回
如果成功，则返回 sizeof(a_sql_uint32)。

如果失败，则返回通用 describe_parameter 错误之一，或:

- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 查询处理状态不等于“标注”时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 不是 a_sql_uint32 的大小时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 指定参数为 TABLE 参数时返回此 set 错误。这包括参数 0 或参数 *n*（*n* 为输入表）。

- EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF 尝试重置参数宽度时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 标注状态

EXTFNAPIV4_DESCRIBE_PARM_SCALE 属性 (Set)

EXTFNAPIV4_DESCRIBE_PARM_SCALE 属性指示参数的标度。用在 describe_parameter_set 情形中。

数据类型
a_sql_uint32

描述
UDF 的参数标度。对于算术型数据类型，参数标度是指数字的小数点右边的位数。

此属性对于以下数据无效：

- 非算数数据类型
- TABLE 参数

用法
这是一个只读属性。其标度由相关联的列数据类型派生而来。数据类型一设置完毕，就不能更改标度。

返回
如果成功，则返回 sizeof(a_sql_uint32)。

如果失败，则返回通用 describe_parameter 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 不是 a_sql_uint32 的大小时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状态不为“标注”时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 指定参数为 TABLE 参数时返回此 set 错误。这包括参数 0 或参数 *n*（*n* 为输入表）。

查询处理状态
在以下状态下有效：

- 标注状态

EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL 属性 (Set)
EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL 属性用于返回参数是否为空值。将此属性用在 `describe_parameter_set` 情形下会返回错误。

数据类型
`a_sql_byte`

描述
如果执行时参数值可为 NULL，则为 true。对于 TABLE 参数或参数 0，值为 false。

用法
这是一个只读属性。

返回
这是只读属性；所有的设置尝试都将导致 `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` 错误。

查询处理状态
不适用。

EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性 (Set)
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性用于返回非重复值数量。将此属性用在 `describe_parameter_set` 情形下会返回错误。

数据类型
`a_v4_extfn_estimate`

描述
返回所有调用中的估计离散值数量。仅对标量参数有效。

用法
这是一个只读属性。

返回
这是只读属性；所有的设置尝试都将导致 `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` 错误。

查询处理状态
不适用。

EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT 属性 (Set)
无论参数是否为常量，都将返回
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性。在
describe_parameter_set 场景使用此属性将返回错误。

数据类型
a_sql_byte

描述
如果语句的参数为常量，则该值为“true”。仅对标量参数有效。

用法
这是一个只读属性。

返回
这是只读属性；所有的设置尝试都将导致
EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE 错误。

查询处理状态
不适用。

EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE 属性 (Set)
EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE 属性指示参数值。用在
describe_parameter_set 情形中。

数据类型
an_extfn_value

描述
如果在描述时是已知的，则为参数值。仅对标量参数有效。

用法
这是一个只读属性。

返回
这是只读属性；所有的设置尝试都将导致
EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE 错误。

查询处理状态
不适用。

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS 属性 (Set)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS 属性指示表中的列数。用于 describe_parameter_set 场景。

数据类型
a_sql_uint32

描述
表中的列数。仅对参数 0 和表参数有效。

用法
如果 UDF 设置此属性，则服务器会比较该值与 **CREATE PROCEDURE** 语句中所提供的参数的名称。如果两个值不一致，则服务器会返回错误。UDF 借此可确保 **CREATE PROCEDURE** 语句的参数名称与 UDF 预期参数名称相同。

返回
如果成功，则返回 sizeof(a_sql_uint32)。

如果失败，则返回通用 describe_parameter 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 不是 size of a_sql_uint32 的大小时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状态不为“标注”时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – 参数不为 TABLE 参数时返回此 set 错误。
- EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF 尝试重置特定表的列数时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 标注状态

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS 属性 (Set)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS 属性指示表中的行数。用于 describe_parameter_set 场景。

数据类型
a_sql_a_v4_extfn_estimate

描述
表中的估计行数。仅对参数 0 和表参数有效。

用法
如果 UDF 估计结果集中的行数，则 UDF 会对参数 0 设置此属性。服务器会在优化过程中用估计的行数作出查询处理决策。不能对输入表设置此值。

如果不设置值，则服务器缺省情况下会使用由 **DEFAULT_TABLE_UDF_ROW_COUNT** 选项指定的行数。

返回
如果成功，则返回 `a_v4_extfn_estimate`。

失败时会返回某个通用 `describe_parameter` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** 不是 `a_v4_extfn_estimate` 的大小时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状态不为“优化”时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – 参数不为 TABLE 参数时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – TABLE 参数不是结果表时返回此 get 错误。
- `EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF 尝试重置特定表的列数时返回此 get 错误。

查询处理状态
在以下状态下有效：

- 查询优化状态

EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY 属性 (Set)
`EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY` 属性指示表中的行的顺序。用于 `describe_parameter_set` 场景。

数据类型
`a_v4_extfn_orderby_list`

描述
表中各行的次序。该属性仅对参数 0 和表参数有效。

用法
通过此属性可用 UDF 代码：

- 确定输入 **TABLE** 参数是否已经过排序
- 声明结果集已经过排序。

如果参数数量是 0，则该属性是指外发结果集。如果该参数 > 0，并且参数类型是表参数，则该属性是指输入 **TABLE** 参数。

顺序由 a_v4_extfn_orderby_list 指定，是支持列顺序号及其相关升序或降序属性的列表的一个结构。如果 UDF 将出站结果集设置为按属性排序，服务器则能够按优化执行排序。例如，如果 UDF 按升序生成结果集首列，则服务器将通过请求消除针对同一列的冗余排序。

如果 UDF 不对外发结果集设置 orderby 属性，则服务器会假定数据未经排序。

如果 UDF 对输入 **TABLE** 参数设置 orderby 属性，则服务器保证会对输入数据进行排序。在这种情况下，UDF 会向服务器作出输入数据必须经过排序的描述。如果服务器检测到运行时冲突，则会报告 SQL 异常。例如，如果 UDF 作出这样的描述，即输入 **TABLE** 参数第一列的顺序必须是升序，但 SQL 语句含有降序子句，则服务器会报告 SQL 异常。

如果 SQL 不含排序子句，则服务器会自动进行排序，以确保按要求对输入 **TABLE** 参数排序。

返回
如果成功，则返回从 a_v4_extfn_orderby_list 中复制的字节数。

查询处理状态
在以下状态下有效：
- 标注状态
- 查询优化状态

EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY (Set)

EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY 属性指示 UDF 需要进行分区。用于 describe_parameter_set 场景。

数据类型
a_v4_extfn_column_list

描述
UDF 开发者使用 **EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY** 以编程方式声明，调用之前须先对 UDF 进行分区。

用法
UDF 可以查询分区以强制执行，或者动态调整分区。UDF 必须分配 **a_v4_extfn_column_list**，需要考虑输入表的总列数，并将其发送至服务器。

返回
如果成功，则返回 a_v4_extfn_column_list 的大小。此值等于：

```
sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) *
number_of_partition_columns
```

失败时会返回某个通用 describe_parameter 错误，或者：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 缓冲区长度小于预期大小时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 标注状态
- 查询优化状态

示例

```
void UDF_CALLBACK my_tpf_proc_describe( a_v4_extfn_proc_context
*ctx )
{
    if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
        a_sql_int32    rc = 0;
        a_v4_extfn_column_list pbcol =
        { 1,        // 1 column in the partition by list
        2 };    // column index 2 requires partitioning

        // Describe partitioning for argument 1 (the table)
        rc = ctx->describe_parameter_set(
        ctx, 1,
        EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
        &pbcol,
        sizeof(pbcol) );

        if( rc == 0 ) {
        ctx->set_error( ctx, 17000,
            “Runtime error, unable set partitioning requirements for
column.” );
        }
    }
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND 属性 (Set)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND 属性指示消费者请求输入表回绕。用于 describe_parameter_set 场景。

数据类型
a_sql_byte

描述
表示消耗程序想要回绕输入表。仅对表输入参数有效。缺省情况下，该属性为“false”。

用法
如果 UDF 需要输入表回绕功能，则 UDF 必须在优化过程中设置此属性。

返回
如果成功，则返回 `sizeof(a_sql_byte)`。

如果失败，则返回通用 `describe_parameter` 错误之一，或：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** 不是 `a_sql_byte` 的大小时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状态不等于“优化”时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – UDF 尝试在参数 0 上设置此属性时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – UDF 尝试为非表的参数设置此属性时返回此 set 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF 尝试将此属性设置为 0 或 1 之外的其它值时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 优化状态

示例
在本示例中，如果在优化状态下调用函数 **my_udf_describe**，则 `describe_parameter_set` 的调用操作将通知表输入参数 1 的生产者可能需要回绕。

示例过程定义：

```
CREATE PROCEDURE my_udf(IN t TABLE(c1 INT))
RESULT (x INT)
EXTERNAL NAME ‘my_udf@myudflib’;
```

示例 `_describe_extfn` API 函数代码段：

```
my_udf_describe(a_v4_extfn_proc_context *cntxt)
{
if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
a_sql_byte rewind_required = 1;
a_sql_int32 ret = 0;

ret = cntxt->describe_parameter_set( cntxt, 1,
EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
&rewind_required,
sizeof(a_sql_byte) );

if( ret <= 0 ) {
    // Handle the error.
}
}
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND 属性 (Set)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND 属性指示参数是否支持回绕。用于 describe_parameter_set 场景。

数据类型
a_sql_byte

描述
表示生产者是否支持回绕。仅对表参数有效。

如果您计划将 DESCRIBE_PARM_TABLE_HAS_REWIND 设置为 true，则还必须实施回绕表回调 (**_rewind_extfn()**)。如果未提供回调方法，则服务器将无法执行 UDF。

用法
如果 UDF 可在没有开销的情况下对其结果表提供回绕功能，则 UDF 会在优化状态下设置此属性。如果 UDF 提供回绕功能的开销很大，请不要设置此属性，或将其设置为 0。如果设置为 0，则服务器会提供回绕支持。

返回
如果成功，则返回 sizeof(a_sql_byte)。

如果失败，则返回通用 describe_parameter 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** 不是 a_sql_byte 的大小时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状态不等于“优化”时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – UDF 尝试为非表的参数设置此属性时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 指定参数不为结果表时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF 尝试将此属性设置为 0 或 1 之外的其它值时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 优化状态

EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS 属性 (Set)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS 属性列出未使用的列。用于 describe_parameter_set 场景。

数据类型
a_v4_extfn_column_list

描述
服务器或者 UDF 将不会使用的输出表列的列表。

对于输出 TABLE 参数，UDF 通常为所有列生成数据，而服务器将使用所有的列。输入 TABLE 参数亦是如此，其中服务器通常为所有列生成数据，而 UDF 将使用所有的列。

但是，某些情况下服务器或 UDF 可能不会消耗所有列。此种情况下的最佳做法是，由 UDF 对输出表执行 **GET** 操作以获取描述属性 **EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS**。该操作对服务器进行查询，以获取输出表中将不会被服务器用到的列的列表。接下来，当为输出表生成列数据时，UDF 可以使用该列表；也就是说，UDF 跳过了对未使用列生成数据的操作。

总之，对于输出表，UDF 会轮询未使用的列的列表。对于输入表，UDF 将推入未使用列的列表。

用法
如果不使用输入 TABLE 参数的特定列，则 UDF 将在优化时设置此属性。UDF 必须分配一个包含输出表所有列的 a_v4_extfn_column_list，然后必须将其传递到服务器。随后服务器将所有未计划列的顺序号标记为 1。服务器将此列表复制到其内部数据结构中。

返回
如果成功，则返回列列表的大小：sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) * number result columns.

失败时会返回某个通用 describe_parameter 错误，或者：

- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状态不为“优化”时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – UDF 尝试从输入表获取此属性时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – UDF 尝试为非表的参数设置此属性时返回此 set 错误。

查询处理状态
在以下状态下有效：

- 优化状态

***describe_udf_get**

`describe_udf_get` 第 4 版 API 方法可从服务器中获取 UDF 属性。

声明

```
a_sql_int32 (SQL_CALLBACK *describe_udf_get)(
    a_v4_extfn_proc_context    *cntxt,
    a_v4_extfn_describe_udf_type    describe_type,
    void    *describe_buffer,
    size_t    describe_buffer_len );
```

参数

参数	描述
cntxt	此 UDF 的过程上下文对象。
describe_type	指示要检索的属性的选择器。
describe_buffer	对于指定的要在服务器中设置的属性，是用于存储描述信息的结构。具体结构或数据类型由 **describe_type** 参数表示。
describe_buffer_length	**describe_buffer** 的字节长度。

返回

成功时会返回 0 或已写入 **describe_buffer** 的字节数。值为 0 表明服务器不能获取属性，但没出错。如果出错或未检索到属性，则此函数会返回某个通用 `describe_udf` 错误。

*返回 *describe_udf_get 的属性*

以下代码中有 `describe_udf_get` 属性。

```
typedef enum a_v4_extfn_describe_udf_type {
    EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS,
    EXTFNAPIV4_DESCRIBE_UDF_LAST
} a_v4_extGetfn_describe_udf_type;
```

EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS 属性 (Get)

`EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS` 属性指示参数数。用于 `describe_udf_get` 场景。

数据类型

`a_sql_uint32`

描述

提供给 UDF 的参数的数量。

用法
用于获取 **CREATE PROCEDURE** 语句中所定义的参数数量。

返回
如果成功，则返回 `sizeof(a_sql_uint32)`。

失败时会返回某个通用 `describe_udf` 错误，或者：

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 描述缓冲区大小不是 `a_sql_uint32` 时返回此 get 错误。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 阶段不大于“初始”阶段时返回此 get 错误。

查询处理阶段

- 标注阶段
- 查询优化阶段
- 计划构建阶段
- 执行阶段

*describe_udf_set

`describe_udf_set` 第 4 版 API 方法可在服务器中设置 UDF 属性。

声明

```
a_sql_int32 (SQL_CALLBACK *describe_udf_set)(
    a_v4_extfn_proc_context    *cntxt,
    a_v4_extfn_describe_udf_type    describe_type,
    const void    *describe_buffer,
    size_t    describe_buffer_len );
```

参数

参数	描述
cntxt	此 UDF 的过程上下文对象。
describe_type	指示要设置的属性的选择器。
describe_buffer	对于指定的要在服务器中设置的属性，是用于存储描述信息的结构。具体结构或数据类型由 **describe_type** 参数表示。
describe_buffer_length	**describe_buffer** 的字节长度。

返回
成功时会返回已写入 **describe_buffer** 的字节数。如果出错或未检索到属性，则此函数会返回某个通用 `describe_udf` 错误。

如果出错或未检索到属性，则此函数会返回某个通用 describe_udf 错误，或者：

- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER — 如果任一 **cntxt** 或 **describe_buffer** 参数是空值，或者 **describe_buffer_length** 是 0，都会返回设置错误。
- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH — 如果所请求的属性的大小和所提供的 **describe_buffer_length** 不一致，则会返回设置错误。

**describe_udf_set 的属性*

以下代码中有 describe_udf_set 属性。

```
typedef enum a_v4_extfn_describe_udf_type {
    EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS,
    EXTFNAPIV4_DESCRIBE_UDF_LAST
} a_v4_extGetfn_describe_udf_type;
```

EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS 属性 (Set)

EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS 属性指示参数数。用于 describe_udf_set 场景。

数据类型

a_sql_uint32

描述

提供给 UDF 的参数的数量。

用法

如果 UDF 设置此属性，则服务器会比较该值与 **CREATE PROCEDURE** 语句中所提供的参数数量。如果两个值不一致，则服务器会返回 SQL 错误。UDF 借此可确保 **CREATE PROCEDURE** 语句的参数数量与 UDF 预期参数数量相同。

返回

如果成功，则返回 sizeof(a_sql_uint32)。

如果失败，则返回通用 describe_udf 错误之一，或：

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – 描述缓冲区大小不是 a_sql_uint32 时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状态不等于“标注”时返回此 set 错误。
- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE — 如果 UDF 尝试重置参数数据类型，则会返回设置错误。

查询处理状态

- 标注状态

描述列的类型 (a_v4_extfn_describe_col_type)

枚举类型 a_v4_extfn_describe_col_type 选择由 UDF 检索或设置的列属性。

实现

```
typedef enum a_v4_extfn_describe_col_type {
    EXTFNAPIV4_DESCRIBE_COL_NAME,
    EXTFNAPIV4_DESCRIBE_COL_TYPE,
    EXTFNAPIV4_DESCRIBE_COL_WIDTH,
    EXTFNAPIV4_DESCRIBE_COL_SCALE,
    EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
    EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
    EXTFNAPIV4_DESCRIBE_COL_LAST
} a_v4_extfn_describe_col_type;
```

成员摘要

组成成员	描述
EXTFNAPIV4_DESCRIBE_COL_NAME	列名称（有效标识符）。
EXTFNAPIV4_DESCRIBE_COL_TYPE	列数据类型。
EXTFNAPIV4_DESCRIBE_COL_WIDTH	字符串宽度（数值精度）。
EXTFNAPIV4_DESCRIBE_COL_SCALE	数值标量。
EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL	如果列可以为空，则为 true。
EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES	该列中的估计离散值数量。
EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE	如果表中的列是唯一的，则该值为 true。
EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT	如果列在语句的生存期内保持不变，则为 true。
EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE	如果在描述时是已知的，则为参数值。
EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER	如果列为表的消耗程序所必需，则该值为 true。

组成成员	描述
EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE	列的最小值（如果已知）。
EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE	列的最大值（如果已知）。
EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT	结果列值为输入表列的子集。
EXTFNAPIV4_DESCRIBE_COL_LAST	v4 API 的第一个非法值。带外值。

描述参数的类型 (a_v4_extfn_describe_parm_type)

枚举类型 `a_v4_extfn_describe_parm_type` 选择由 UDF 检索或设置的参数属性。

实现

```
typedef enum a_v4_extfn_describe_parm_type {
    EXTFNAPIV4_DESCRIBE_PARM_NAME,
    EXTFNAPIV4_DESCRIBE_PARM_TYPE,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,

    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS,

    EXTFNAPIV4_DESCRIBE_PARM_LAST
} a_v4_extfn_describe_parm_type;
```

成员摘要

组成成员	描述
EXTFNAPIV4_DESCRIBE_PARM_NAME	参数名称（有效标识符）。
EXTFNAPIV4_DESCRIBE_PARM_TYPE	数据类型。
EXTFNAPIV4_DESCRIBE_PARM_WIDTH	字符串宽度（数值精度）。
EXTFNAPIV4_DESCRIBE_PARM_SCALE	数值标量。

组成成员	描述
EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL	如果该值可以是空值，则为 true。
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES	所有调用中的估计离散值数量。
EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT	如果语句的参数为常量，则该值为 true。
EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE	如果在描述时是已知的，则为参数值。
这些选择程序可检索或设置 TABLE 参数的属性。这些枚举器值无法与标量参数一起使用：	
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS	表中的列数。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS	表中的估计行数。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY	表中各行的次序。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY	分区；将 *number_of_columns*=0 用于 ANY。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND	如果消耗程序希望具备回绕输入表的功能，则为 true。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND	如果生产者支持回绕功能，则返回 true。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS	服务器或者 UDF 将不会使用的输出表列的列表。
EXTFNAPIV4_DESCRIBE_PARM_LAST	v4 API 的第一个非法值。带外值。

描述返回值 (a_v4_extfn_describe_return)

当 `a_v4_extfn_proc_context.describe_xxx_get()` 或 `a_v4_extfn_proc_context.describe_xxx_set()` 不成功时，枚举类型 `a_v4_extfn_describe_return` 将提供一个返回值。

实现

```
typedef enum a_v4_extfn_describe_return {
   EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE          =  0,   // the specified operation has no
meaning either for this attribute or in

the current context.
```

```
    EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH        = -1,   // the provided buffer size
does not match the required length or the

length is insufficient.
    EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER       = -2,   // the provided parameter number
is invalid
    EXTFNAPIV4_DESCRIBE_INVALID_COLUMN           = -3,   // the column number is invalid
for this TABLE parameter
    EXTFNAPIV4_DESCRIBE_INVALID_STATE          = -4,   // the describe method call is not
valid in the present state
    EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE       = -5,   // the attribute is known but not
appropriate for this object
    EXTFNAPIV4_DESCRIBE_UNKNOWN_ATTRIBUTE        = -6,   // the identified attribute is
not known to this server version
    EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER       = -7,   // the specified parameter is
not a TABLE parameter (for describe_col_get()

or set())
    EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE       = -8,   // the specified attribute
value is illegal
    EXTFNAPIV4_DESCRIBE_LAST                = -9
} a_v4_extfn_describe_return;
```

成员摘要

组成成员	返回值	描述
EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE	0	指定操作对于此属性或者在当前上下文中没有意义。
EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH	-1	所提供的缓冲区大小同需要的长度不匹配，或者长度不够。
EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER	-2	提供的参数数量无效。
EXTFNAPIV4_DESCRIBE_INVALID_COLUMN	-3	列编号对于此 TABLE 参数无效。
EXTFNAPIV4_DESCRIBE_INVALID_STATE	-4	在当前状态中对 describe 方法的调用无效。
EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE	-5	属性已知，但并不适合于此对象。
EXTFNAPIV4_DESCRIBE_UNKNOWN_ATTRIBUTE	-6	该服务器版本不能识别所确定的属性。
EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER	-7	指定的参数不是 TABLE 参数（针对 `describe_col_get()` 或 `describe_col_set()`）。
EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE	-8	所指定的属性无效。

组成成员	返回值	描述
EXTFNAPIV4_DESCRIBE_LAST	-9	v4 API 的第一个非法值。

描述

`a_v4_extfn_proc_context.describe_xxx_get()` 和 `a_v4_extfn_proc_context.describe_xxx_set()` 的返回值为带符号整数。如果结果为正整数，则操作成功，返回值为 复制的字节数量。如果返回值小于或等于零，则操作不成功，返回值为 `a_v4_extfn_describe_return` 值中的一个。

描述 UDF 的类型 (a_v4_extfn_describe_udf_type)

使用枚举类型 `a_v4_extfn_describe_udf_type` 对 UDF 检索或设置的逻辑属性进行选择。

实现

```
typedef enum a_v4_extfn_describe_udf_type {
    EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS,
    EXTFNAPIV4_DESCRIBE_UDF_LAST
} a_v4_extfn_describe_udf_type;
```

成员摘要

组成成员	描述
EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS	提供给 UDF 的参数的数量。
EXTFNAPIV4_DESCRIBE_UDF_LAST	带外值。

描述

UDF 使用 `a_v4_extfn_proc_context.describe_udf_get()` 方法以检索属性， 使用 `a_v4_extfn_proc_context.describe_udf_set()` 方法设置 关于 UDF 的整体属性。 枚举器 `a_v4_extfn_describe_udf_type` 对 UDF 检索或设置的逻辑属性进行选择。

执行状态 (a_v4_extfn_state)

`a_v4_extfn_state` 枚举类型表示 UDF 的查询处理阶段。

实现

```
typedef enum a_v4_extfn_state {
   EXTFNAPIV4_STATE_INITIAL,                // Server initial state,
not used by UDF
    EXTFNAPIV4_STATE_ANNOTATION,                // Annotating parse
tree with UDF reference
    EXTFNAPIV4_STATE_OPTIMIZATION,              // Optimizing
```

```
   EXTFNAPIV4_STATE_PLAN_BUILDING,             // Building execution
plan
    EXTFNAPIV4_STATE_EXECUTING,                 // Executing UDF and
fetching results from UDF
    EXTFNAPIV4_STATE_LAST
} a_v4_extfn_state;
```

成员摘要

组成成员	描述
EXTFNAPIV4_STATE_INITIAL	服务器初始阶段。此查询处理阶段调用的唯一 UDF 方法为 `_start_extfn`。
EXTFNAPIV4_STATE_ANNOTATION	含 UDF 参考的加标注的分析树。UDF 并非在此阶段调用。
EXTFNAPIV4_STATE_OPTIMIZATION	优化。服务器调用 UDF 的 `_start_extfn` 方法，然后调用 `_describe_extfn` 函数。
EXTFNAPIV4_STATE_PLAN_BUILDING	构建查询执行计划。服务器调用 UDF 的 `_describe_extfn` 函数。
EXTFNAPIV4_STATE_EXECUTING	执行 UDF 并从 UDF 中读取结果。服务器先调用 `_describe_extfn` 函数，然后再从 UDF 中读取数据。接下来，服务器调用 `_evaluate_extfn` 以启动读取循环。读取循环期间，服务器调用 `a_v4_extfn_table_func` 中所定义的函数。读取结束后，服务器调用 UDF 的 `_close_extfn` 函数。
EXTFNAPIV4_STATE_LAST	v4 API 的第一个非法值。带外值。

描述

`a_v4_extfn_state` 枚举指示服务器所处的 UDF 执行阶段。当服务器从一个阶段转换到下一阶段时，服务器将通过调用 UDF 的 `_leave_state_extfn` 函数通知 UDF 它将要结束前一个阶段。服务器通过调用 UDF 的 `_enter_state_extfn` 函数通知 UDF 它将要进入新的阶段。

UDF 的查询处理阶段会限制 UDF 可执行的操作。例如，在标注阶段，UDF 只可检索常量参数的数据类型。

外部函数 (a_v4_extfn_proc)

服务器使用 a_v4_extfn_proc 结构调用 UDF 中的各入口点。服务器将 a_v4_extfn_proc_context 实例传给每个函数。

方法总结

方法	描述
_start_extfn	分配一个结构，并将其地址存储于 a_v4_extfn_proc_context 中的 **_user_data** 字段。
_finish_extfn	释放一个结构，其地址存储于 a_v4_extfn_proc_context 中的 **user_data** 字段。
_evaluate_extfn	基于新参数组的每次函数调用都将调用所需的函数指针。
_describe_extfn	请参见描述 API （第 24 页）。
_enter_state_extfn	UDF 可以使用此函数来分配结构。
_leave_state_extfn	UDF 可以使用此函数来释放状态所需的内存或资源。

_start_extfn

使用 _start_extfn 第 4 版 API 方法作为指向初始化函数的可选指针，其唯一参数是指向 a_v4_extfn_proc_context 结构的指针。

声明

```
_start_extfn(
a_v4_extfn_proc_context *
)
```

用法

使用 _start_extfn 方法分配结构并将其地址存入 a_v4_extfn_proc_context 中的 _user_data 字段。如果无须进行初始化，则必须将此函数指针设置为空指针。

参数

参数	描述
cntxt	过程上下文对象。

_finish_extfn

将 _finish_extfn v4 API 方法用作指向一个关闭函数的可选指针，其唯一参数为指向 a_v4_extfn_proc_context 的指针。

声明

```
_finish_extfn(
     a_v4_extfn_proc_context *cntxt,
)
```

用法

_finish_extfn API 释放一个结构，其地址存储于 a_v4_extfn_proc_context 中的 user_data 字段。如果无需进行清理，则必须将此函数指针设置为空指针。

参数

参数	描述
cntxt	过程上下文对象。

_evaluate_extfn

将 _evaluate_extfn v4 API 方法作为所需的函数指针，并于每次基于新参数组调用函数时调用该指针。

声明

```
_evaluate_extfn(
     a_v4_extfn_proc_context *cntxt,
     void *args_handle
)
```

用法

_evaluate_extfn 函数必须向服务器描述如何通过填充 a_v4_extfn_table 结构的 a_v4_extfn_table_func 部分来提取结果，并且在使用参数 0 的上下文中通过 set_value 方法将该信息发送至服务器。在基于参数 0 调用 set_value 方法之前，此函数还必须通过填充 a_v4_extfn_table 结构的 a_v4_extfn_value_schema 部分来通知服务器其输出模式。它可以使用回调函数 get_value 来访问其输入参数值。此时 UDF 可以使用常量和非常量参数。

参数

参数	描述
cntxt	过程上下文对象。
args_handle	服务器中的参数的句柄。

_describe_extfn

在每个状态开始时调用 _describe_extfn，以允许服务器获取或设置逻辑属性。UDF 可以使用 a_v4_proc_context object 中的六种 describe 方法（describe_parameter_get，describe_parameter_set，describe_column_get，describe_column_set，describe_udf_get，和 describe_udf_set）来完成此操作。

请参见描述 API （第 24 页）。

_enter_state_extfn

UDF 可以将 _enter_state_extfn v4 API 方法作为可选入口点来执行，每当 UDF 进入新的状态时则给予通知。

声明

```
_enter_state_extfn(
     a_v4_extfn_proc_context *cntxt,
)
```

用法

UDF 可以使用此通知来分配结构。

参数

参数	描述
cntxt	过程上下文对象。

_leave_state_extfn

_leave_state_extfn v4 API 方法为可选入口点，当 UDF 转出查询处理状态时，可以使用该方法接收通知。

声明

```
_leave_state_extfn(
     a_v4_extfn_proc_context *cntxt,
)
```

用法

UDF 可用该通知释放状态所需的内存或资源。

参数

参数	描述
cntxt	过程上下文对象。

外部过程上下文 (a_v4_extfn_proc_context)

请使用 a_v4_extfn_proc_context 结构以保留来自于服务器和 UDF 的上下文信息。

实现

```
typedef struct a_v4_extfn_proc_context {
.
.
.
} a_v4_extfn_proc_context;
```

方法总结

返回类型	方法	描述
短整型	**get_value**	获取 UDF 的输入参数。
短整型	**get_value_is_constant**	允许 UDF 查询其所获取的参数是否为常量。
短整型	**set_value**	UDF 在 _evaluate_extfn 或 _describe_extfn 函数中使用该方法，向服务器描述其所输出的结果值的格式，并告知服务器如何从 UDF 中提取结果值。
a_sql_uint32	**get_is_cancelled**	每秒（或每两秒）调用一次 **get_is_cancelled** 方法，以查看用户是否已中断当前语句的执行。
短整型	**set_error**	对当前语句执行回滚操作，并且生成一个错误。
无类型	**log_message**	将消息写入到消息日志中。
短整型	**convert_value**	将一种数据类型转换为另一种数据类型。
短整型	**get_option**	获取可设置选项的值。
无类型	**alloc**	分配长度至少为 "len" 的内存块。
无类型	**free**	释放由 **alloc()** 所分配的具有指定生命周期的内存块。
a_sql_uint32	**describe_column_get**	请参见*describe_column_get （第 24 页）。
a_sql_uint32	**describe_column_set**	请参见*describe_column_set （第 38 页）。
a_sql_uint32	**describe_parameter_get**	请参见*describe_parameter_get （第 53 页）。
a_sql_uint32	**describe_parameter_set**	请参见*describe_parameter_set （第 69 页）。

返回类型	方法	描述
a_sql_uint32	**describe_udf_get**	请参见*describe_udf_get （第 83 页）。
a_sql_uint32	**describe_udf_set**	请参见*describe_udf_set （第 84 页）。
短整型	**open_result_set**	为表中的值打开结果集。
短整型	**close_result_set**	关闭已打开的结果集。
短整型	**get_blob**	检索为 BLOB 的输入参数。
短整型	**set_cannot_be_distrib-uted**	禁用 UDF 级别的分发（即使库具有可分发性）。

数据成员及数据类型的摘要

数据成员	数据类型	描述
_user_data	void*	使用外部例程所需的任何环境数据皆可对该指针赋值。
_executionMode	a_sql_uint32	通过 **External_UDF_Execution_Mode** 选项指出请求的调试／跟踪级别。这是只读字段。
current_state	a_sql_uint32	*current_state* 属性反映了当前环境的执行模式。可以通过诸如 `_describe_extfn` 的函数进行查询，以确定应采取何种措施。

描述

除了保留来自服务器和 UDF 的上下文信息之外，`a_v4_extfn_proc_context` 结构允许 UDF 回调至服务器以执行某些操作。UDF 可以在 `_user_data` 成员的该结构中存储私有数据。由服务器将此结构的实例传至 `a_v4_extfn_proc` 方法中的函数。直至服务器达到标注状态之后，用户数据才予以保留。

get_value

使用 `get_value` 第 4 版 API 方法在 SQL 查询中获取向 UDF 发送的输入参数的值。

声明

```
short get_value(
       void *           arg_handle,
       a_sql_uint32     arg_num,
       an_extfn_value *value
       )
```

用法

`get_value` API 在计算方法中用于检索每个 UDF 输入参数的值。对于窄型参数数据 (>32K)，调用 `get_value` 就足以检索整个参数值。

通过任何有权访问 arg_handle 指针的 API，均可调用 get_value API。这包括可接收 a_v4_table_context 并将其作为参数的 API 函数。a_v4_table_context 有可用于执行上述调用的 args_handle 成员变量。

对于所有长度固定的数据类型，均可从返回值中得到数据，无须执行其他调用即可获取所有数据。生产程序可决定最大长度，而后者会通过调用 get_value 方法全部返回。所有长度固定的数据类型都应该保证可以存入一个连续的缓冲区。对于变长数据，限制取决于生产程序。

对于长度不固定的数据类型，可能必须用 get_blob 方法创建 blob 才能获得数据，具体情况取决于数据的长度。可将宏 **EXTFN_IS_INCOMPLETE** 用于 get_value 返回的值，以判断是否需要 blob 对象。如果 **EXTFN_IS_INCOMPLETE** 的计算结果是 true，则需要 blob。

对于表输入参数，类型是 **AN_EXTFN_TABLE**。对于此类参数，必须通过 open_result_set 方法创建结果集，才能读取表中的值。

如果调用 _evaluate_extfn API 前 UDF 需要参数值，则 UDF 应该实现 _describe_extfn API。通过 _describe_extfn API，UDF 能用 describe_parameter_get 方法获取常量表达式的值。

参数

参数	描述
arg_handle	消耗程序提供的上下文指针。
arg_num	要获取其值的参数的索引。参数索引始于 1。
value	指定的参数的值。

返回
如果成功则返回 1，否则返回 0。

an_extfn_value 结构
an_extfn_value 结构代表着 get_value API 返回的输入参数的值。

这段代码显示 **an_extfn_value** 结构的声明方法：

```
 short typedef struct an_extfn_value {
        void*           data;
        a_sql_uint32    piece_len,
        an_extfn_value *value {
                                a_sql_uint32    total_len;
                                a_sql_uint32    remain_len;
        } len;
        a_sql_data_type    type;
} an_extfn_value;
```

下表说明调用 `get_value` 方法后返回的 **an_extfn_value** 对象的值：

get_value API 返回的值	EXTFN_IS_INCOMPLETE	total_len	piece_len	数据
空值	FALSE	0	0	空值
空字符串	FALSE	0	0	非空值
大小 < MAX_UINT32	FALSE	actual	actual	非空值
大小 < MAX_UINT32	TRUE	actual	0	非空值
大小 >= MAX_UINT32	TRUE	MAX_UINT32	0	非空值

an_extfn_value 的类型字段含有值的数据类型。对于以表为输入参数的 UDF，那种参数的数据类型是 **DT_EXTFN_TABLE**。对于第 4 版表 UDF，不会使用 `remain_len` 字段。

get_value_is_constant

使用 `get_value_is_constant` 第 4 版 API 方法判断指定的输入参数是否为常量。

声明

```
short get_value_is_constant(
       void *          arg_handle,
       a_sql_uint32    arg_num,
       an_extfn_value *value_is_constant
       )
```

用法

UDF 可询问给定的参数是否为常量。可借此优化 UDF，例如，可在首次调用 `_evaluate_extfn` 函数期间运行一次，而不是每当执行计算调用时都运行。

参数

参数	描述
arg_handle	服务器中参数的句柄。
arg_num	要检索的输入参数的索引值。 索引值是 1..N 的值。
value_is_constant	用于进行存储的 out 参数是常量。

返回

如果成功则返回 1，否则返回 0。

set_value

使用 `set_value` 第 4 版 API 方法向消耗程序描述结果集有多少列，以及应该如何读取数据。

声明

```
short set_value(
       void *          arg_handle,
       a_sql_uint32    arg_num,
       an_extfn_value *value
       )
```

用法

此方法由 UDF 在 `_evaluate_extfn` API 中使用。UDF 必须调用 `set_value` 方法才能告知消耗程序结果集中有多少列，以及 UDF 支持什么 `a_v4_extfn_table_func` 函数。

对于 `set_value` API，UDF 会通过 `_evaluate_extfn` API 或通过 `a_v4_extfn_table_context` 结构的 **args_handle** 成员提供相应 **arg_handle** 指针。

对于第 4 版表 UDF，`set_value` 方法的 **value** 的参数必须是 *DT_EXTFN_TABLE* 类型的参数。

参数

参数	描述
arg_handle	消耗程序提供的上下文指针。
arg_num	要对其设置值的参数的索引。只有 0 是受支持的参数。
value	指定的参数的值。

返回

如果成功则返回 1，否则返回 0。

get_is_cancelled

使用 `get_is_cancelled` 第 4 版 API 方法判断是否已取消语句。

声明

```
short get_is_cancelled(
       a_v4_extfn_proc_context *          cntxt,

       )
```

用法
如果 UDF 条目的运行时间加长（达到数秒），则应该（如有可能）每秒或每两秒都调用一次 `get_is_cancelled` 回调函数，以确定用户是否已打断当前语句的执行。如果已打断语句的执行，则会返回非零值，然后应该立即返回 UDF 条目。调用 `_finish_extfn` 函数以进行必要清理。此后不得调用任何其他 UDF 条目。

参数

参数	描述
cntxt	过程上下文对象。

返回
如果已打断语句的执行，则会返回非零值。

set_error
使用 `set_error` 第 4 版 API 方法将错误反馈给服务器并最终告知用户。

声明

```
void set_error(
       a_v4_extfn_proc_context *          cntxt,
       a_sql_uint32                       error_number,
       const char                         *error_desc_string
       )
```

用法
如果 UDF 条目遇到应该向用户发送错误消息，并且关闭当前语句的错误，则调用 `set_error` API。调用后，`set_error` API 会回退当前语句，用户还会看到“`Error raised by user-defined function: <error_desc_string>`”。SQLCODE 是所提供的 <error_number> 的求反形式。

为了防止与现有错误代码发生冲突，UDF 生成的错误编号应该介于 17000 和 99999 之间。如果提供的数值不在此范围内，仍会回退语句，但错误消息会是“`Invalid error raised by user-defined function: (<error_number>) <error_desc_string>`”，SQLCODE 为 -1577。 **error_desc_string** 的最大长度为 140 个字符。

对 `set_error` 的调用执行完毕后，UDF 条目会立即执行返回操作；最终会调用 `_finish_extfn` 函数，以执行必要清理。此后不得调用任何其他 UDF 条目。

参数

参数	描述
cntxt	过程上下文对象
error_number	要设置的错误编号
error_desc_string	要使用的消息字符串

log_message

使用 `log_message` 第 4 版 API 方法向服务器消息日志发送消息。

声明

```
short log_message(
       const char         *msg,
       short              msg_length
       )
```

用法

`log_message` 方法用于将消息写入消息日志。消息字符串必须 是可显示的文本字符串，不长于 255 字节；较长的消息可能会被截断。

参数

参数	描述
msg	要写入日志的消息字符串
msg_length	消息字符串的长度

convert_value

使用 `convert_value` 第 4 版 API 方法转换数据类型。

声明

```
short convert_value(
       an_extfn_value *input,
       an_extfn_value *output
       )
```

用法

. `convert_value` API 主要用于在 `DT_DATE`、`DT_TIME`、`DT_TIMESTAMP` 和 `DT_TIMESTAMP_STRUCT` 之间进行转换。会向该函数传递输入和输出 `an_extfn_value`。

输入参数

参数	描述
an_extfn_value.data	输入数据指针
an_extfn_value.total_len	输入数据的长度
an_extfn_value.type	输入的 DT_ datatype

输出参数

参数	描述
an_extfn_value.data	UDF 提供的输出数据点
an_extfn_value.piece_len	输出数据的最大长度
an_extfn_value.total_len	服务器设置的转换后的长度
an_extfn_value.type	所需输出的 DT_ datatype

返回

如果成功则返回 1，否则返回 0。

get_option

`get_option` 第 4 版 API 方法用于获取可设置选项的值。

声明

```
short get_option(
a_v4_extfn_proc_context * cntxt,
char *option_name,
an_extfn_value *output
)
```

参数

参数	描述
cntxt	过程上下文对象
option_name	要获取的选项的名称
输出	• `an_extfn_value.data` — UDF 提供的输出数据指针 • `an_extfn_value.piece_len` — 输出数据的最大长度 • `an_extfn_value.total_len` — 转换数据后的服务器设置 长度 • `an_extfn_value.type` — 服务器设置的值数据类型

返回
如果成功则返回 1，否则返回 0。

alloc
`alloc` v4 API 方法分配内存块。

声明

```
void*alloc(
     a_v4_extfn_proc_context *cntxt,
     size_t len
     )
```

用法
分配长度至少为 **len** 的内存块。返回的内存为 8 字节对齐。

提示： 将 `alloc()` 方法用作内存分配的唯一方法，这样便允许服务器跟踪外部例程使用的内存量。服务器可以修改其他内存用户、跟踪泄漏以及提供改进的诊断和监控功能。

仅当 **external_UDF_execution_mode** 设置为值 1 或 2（校验模式或跟踪模式）时才会启用内存跟踪。

参数

参数	描述
cntxt	过程上下文对象
len	分配的内存长度（以字节为单位）

free
`free` v4 API 方法释放已分配的内存块。

声明

```
void free(
     a_v4_extfn_proc_context *cntxt,
     void *mem
)
```

用法
释放由 `alloc()` 分配的具有指定生命周期的内存块。

仅当 **external_UDF_execution_mode** 设置为 1 或 2 时（校验模式或跟踪模式），启用内存跟踪。

参数

参数	描述
cntxt	过程上下文对象
mem	指向由 alloc 方法分配的内存块的指针。

open_result_set

open_result_set 第 4 版 API 方法用于打开表值结果集。

声明

```
short open_result_set(
a_v4_extfn_proc_context *cntxt,
a_v4_extfn_table *table,
a_v4_extfn_table_context **result_set
)
```

用法

open_result_set 用于打开表值结果集。UDF 可打开结果集，从 DT_EXTFN_TABLE 类输入参数中读取行。服务器（或另一 UDF）可打开结果集，通过 UDF 读取行。

参数

参数	描述
cntxt	过程上下文对象
table	要对其打开结果集的表对象
result_set	已设置为已打开的结果集的输出参数

返回

如果成功则返回 1，否则返回 0。

有关 open_result_set 用法示例，请参见 fetch_block 和 fetch_into 第 4 版 API 方法说明。

close_result_set

close_result_set 第 4 版 API 方法用于关闭已打开的结果集。

声明

```
short close_result_set(
    a_v4_extfn_proc_context *cntxt,
    a_v4_extfn_table_context *result_set
)
```

用法
仅可对每个结果集使用一次 close_result_set。

参数

参数	描述
cntxt	过程上下文对象
result_set	要关闭的结果集

返回
如果成功则返回 1，否则返回 0。

get_blob
使用 get_blob 第 4 版 API 方法检索输入 blob 参数。

声明

```
short get_blob(
       void             *arg_handle,
       a_sql_uint32     arg_num,
       a_v4_extfn_blob **blob
       )
```

用法
使用 get_blob 在调用 get_value() 后检索 blob 输入参数。如果 *piece_len* < *total_len*，则可用宏 EXTFN_IS_INCOMPLETE 判断是否必须有 blob 对象才能读取通过 get_value() 返回的值的数据。blob 对象会以输出参数的形式返回，并由调用方拥有。

get_blob 用于获取可用于读取 blob 内容的 blob 句柄。仅可对含有 blob 对象的列调用此方法。

参数

参数	描述
arg_handle	服务器中参数的句柄
arg_num	参数值是 1...N 的数值
blob	含有 blob 对象的输出参数

返回
如果成功则返回 1，否则返回 0。

set_cannot_be_distributed
即使已达到库级别的分配标准，set_cannot_be_distributed 第 4 版 API 方法也可在 UDF 级别禁止分配。

声明

```
void set_cannot_be_distributed( a_v4_extfn_proc_context *cntxt)
```

用法
缺省行为下，库可分配时 UDF 也可分配。可在 UDF 中用 set_cannot_be_distributed 作出禁止对服务器分配的决策。

许可证信息 (a_v4_extfn_license_info)

如果您是设计合作伙伴，那么请使用 a_v4_extfn_license_info 结构定义 UDF 的库级别许可证校验，包括公司名称、库版本信息和 SAP 提供的许可证密钥。

实现

```
typedef struct an_extfn_license_info {
    short     version;
} an_extfn_license_info;

typedef struct a_v4_extfn_license_info {
    an_extfn_license_info version;

    const char        name[255];
    const char        info[255];
    void *        key;
} a_v4_extfn_license_info;
```

数据成员汇总

数据成员	描述
version	仅供内部使用。必须设为 1。
name	UDF 设置为公司名称的值。
info	UDF 为附加库信息（如库版本和内部版本号）设置的值。
key	（仅设计合作伙伴）SAP 提供的许可证密钥。该密钥为 26 个字符的数组。

优化程序估计 (a_v4_extfn_estimate)

请使用 a_v4_extfn_estimate 结构来描述估计，其中包括一个值 和一个置信度。

实现

```
typedef struct a_v4_extfn_estimate {
    double    value;
```

```
    double    confidence;
} a_v4_extfn_estimate;
```

数据成员及数据类型的摘要

数据成员	数据类型	描述
value	double	估计值。
confidence	double	与估计相关联的置信度。置信度的变化范围为 0.0 至 1.0，其中 0.0 表示估计无效，1.0 表示估计经确认为真。

按列表排序 (a_v4_extfn_orderby_list)

请使用 `a_v4_extfn_orderby_list` 结构以描述表的 ORDER BY 属性。

实现

```
typedef struct a_v4_extfn_orderby_list {
    a_sql_uint32        number_of_elements;
    a_v4_extfn_order_el order_elements[1];      // there are
number_of_elements entries
} a_v4_extfn_orderby_list;
```

数据成员及数据类型的摘要

数据成员	数据类型	描述
number_of_elements	a_sql_uint32	条目数
order_elements[1]	a_v4_extfn_order_el	元素的顺序

描述

存在一些 *number_of_elements* 条目，其中每个条目都含有一个表明 元素是升序还是降序的标志，以及一个 指示相关表中对应列的列索引。

通过列号分区 (a_v4_extfn_partitionby_col_num)

`a_v4_extfn_partitionby_col_num` 枚举类型表示列号，允许 UDF 表示对 **PARTITION BY** 的支持（类似于 SQL 所提供的支持）。

实现

```
typedef enum a_v4_extfn_partitionby_col_num {
    EXTFNAPIV4_PARTITION_BY_COLUMN_NONE = -1,        // NO PARTITION
BY
    EXTFNAPIV4_PARTITION_BY_COLUMN_ANY  = 0,         // PARTITION BY
ANY
                                  // + INTEGER representing a specific
column ordinal
} a_v4_extfn_partitionby_col_num;
```

成员摘要

枚举类型 a_v4_extfn_partitionby_col_num 的成员	值	描述
EXTFNAPIV4_PARTITION_BY_COLUMN_NONE	-1	不进行分区
EXTFNAPIV4_PARTITION_BY_COLUMN_ANY	0	通过任意正整数（代表特定的列顺序号）进行分区
Column Ordinal Number	N > 0	进行分区的表列号的序号

描述

此结构允许 UDF 以编程方式描述分区以及进行分区的列。

当填充 `a_v4_extfn_column_list number_of_columns` 字段时，使用此枚举类型。当向服务器描述对分区操作的支持情况时，UDF 将 `number_of_columns` 设置为一个枚举值，或将其设置为一个代表列序号的正整数。 例如，若要向服务器描述不支持分区操作，则需要创建如下结构：

```
a_v4_extfn_column_list nopby = {
EXTFNAPIV4_PARTITION_BY_COLUMN_NONE,
0
};
```

EXTFNAPIV4_PARTITION_BY_COLUMN_ANY 成员通知服务器，UDF 支持任何形式的分区操作。

若要描述进行分区的一组序号，则需创建如下结构：

```
a_v4_extfn_column_list nopby = {
2,
3, 4
};
```

此结构表明将通过 2 列进行分区，其序号分别分别为 3 和 4。

注意： 此例仅用于描述目的，并非合法代码。调用方必须相应地分配容纳 3 个整数的结构。

行 (a_v4_extfn_row)

请使用 `a_v4_extfn_row` 结构表示单行中的数据。

实现

```
/* a_v4_extfn_row - */
typedef struct a_v4_extfn_row {
    a_sql_uint32          *row_status;
    a_v4_extfn_column_data    *column_data;
} a_v4_extfn_row;
```

数据成员及数据类型的摘要

数据成员	数据类型	描述
row_status	a_sql_uint32 *	行的状态。对于存在的行，将该值设置为 1，否则将该值设置为 0。
column_data	a_v4_extfn_column_data *	行的列数据数组。

描述

行结构包含特定行列的信息。此结构定义了一个单独行的状态，并且包括一个指向行内单独列的指针。 行的状态是一个表明该行是否存在的标志。可以使用嵌套的提取调用更改行的状态标志，而无需对行块结构进行处理。

将 *row_status* 标志设置为 1，则表明该行可用并且可被包括在结果集中。将 *row_status* 设置为 0，则表明应忽略该行。当使用 TPF 作为过滤器时，这一点非常有用。因为 TPF 可能会将输入表中的各行传递至结果集，但又想跳过其中的某些行，此时，它就可以通过将这些行的状态标志设置为 0 来予以实现。

行块 (a_v4_extfn_row_block)

请使用 `a_v4_extfn_row_block` 结构表示行块中的数据。

实现

```
/* a_v4_extfn_row_block - */
typedef struct a_v4_extfn_row_block {
    a_sql_uint32        max_rows;
    a_sql_uint32        num_rows;
    a_v4_extfn_row  *row_data;
} a_v4_extfn_row_block;
```

数据成员及数据类型的摘要

数据成员	数据类型	描述
max_rows	a_sql_uint32	该行块能够处理的最大行数
num_rows	a_sql_uint32	必须小于或等于行块所包含的最大行数
row_data	a_v4_extfn_row *	行块矢量

描述

`fetch_into` 和 `fetch_block` 方法使用行块结构以实现数据生成及数据消耗。分配器设置最大行数。生产者错误地设置了行数。数据消耗程序不应尝试执行超出所生成行数的读取操作。

由 row_block 结构的所有者确定 *max_rows* 数据成员的值。例如，当表 UDF 执行 fetch_into 时，服务器会将 *max_rows* 的值设为可以装入 128K 内存的行的数量。然而，当表 UDF 执行 fetch_block 时，将会自行确定 *max_rows* 的值。

约束和限制

num_rows 和 *max_rows* 的值皆大于 0。*num_rows* 必须小于等于 *max_rows*。对于有效行块而言，row_data 字段不应为 NULL。

表 (a_v4_extfn_table)

请使用 a_v4_extfn_table 结构以表示数据在表中的存储方法，以及消耗程序提取数据的方法。

实现

```
typedef struct a_v4_extfn_table {
    a_v4_extfn_table_func       *func;
    a_sql_uint32                number_of_columns;
} a_v4_extfn_table;
```

数据成员及数据类型的摘要

数据成员	数据类型	描述
func	a_v4_extfn_table_func *	该成员包含一组函数指针，供消耗程序提供结果数据使用
number_of_columns	a_sql_uint32 *	表中的列数

表上下文 (a_v4_extfn_table_context)

a_v4_extfn_table_context 结构表示某个表上打开的结果集。

实现

```
typedef struct a_v4_extfn_table_context {

//    size_t struct_size;

    /* fetch_into() - fetch into a specified row_block. This entry point
        is used when the consumer has a transfer area with a specific format.
       The fetch_into() function will write the fetched rows into the provided row block.
    */
    short (UDF_CALLBACK *fetch_into)(a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block *);

    /* fetch_block() - fetch a block of rows. This entry point is used
        when the consumer does not need the data in a particular format. For example,
        if the consumer is reading a result set and formatting it as HTML, the consumer
       does not care how the transfer area is layed out. The fetch_block() entry point is
        more efficient if the consumer does not need a specific layout.

        The row_block parameter is in/out. The first call should point to a NULL row
block.
        The fetch_block() call sets row_block to a block that can be consumed, and this
block
        should be passed on the next fetch_block() call.
```

```
    */
    short (UDF_CALLBACK *fetch_block)(a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block **row_block);

    /*  rewind() - this is an optional entry point. If NULL, rewind is not supported.
Otherwise,
        the rewind() entry point restarts the result set at the beginning of the table.
    */
    short (UDF_CALLBACK *rewind)(a_v4_extfn_table_context *);

    /* get_blob() - If the specified column has a blob object, return it.  The blob
         is returned as an out parameter and is owned by the caller.  This method should
         only be called on a column that contains a blob.  The helper macro
EXTFN_COL_IS_BLOB can
         be used to determine whether a column contains a blob.
    */
    short (UDF_CALLBACK *get_blob)(a_v4_extfn_table_context *cntxt,
                    a_v4_extfn_column_data *col,
                    a_v4_extfn_blob **blob);

   /* The following fields are reserved for future use and must be initialized to NULL.
*/
    void  *reserved1_must_be_null;
    void  *reserved2_must_be_null;
    void  *reserved3_must_be_null;
    void  *reserved4_must_be_null;
    void  *reserved5_must_be_null;

    a_v4_extfn_proc_context     *proc_context;
    void                        *args_handle;           // use in
a_v4_extfn_proc_context::get_value() etc.
    a_v4_extfn_table            *table;
    void                        *user_data;
    void                        *server_internal_use;

   /* The following fields are reserved for future use and must be initialized to NULL.
*/
    void  *reserved6_must_be_null;
    void  *reserved7_must_be_null;
    void  *reserved8_must_be_null;
    void  *reserved9_must_be_null;
    void  *reserved10_must_be_null;

} a_v4_extfn_table_context;
```

方法总结

数据类型	方法	描述
短整型	**fetch_into**	读取到特定的 `row_block`
短整型	**fetch_block**	提取行块
短整型	**rewind**	在表的开头重启结果集
短整型	**get_blob**	如果指定的列包含 BLOB 对象，返回 BLOB 对象

数据成员及数据类型的摘要

数据成员	数据类型	描述
proc_context	a_v4_extfn_ proc_context *	指向过程上下文对象的指针。UDF 可以使用该指针设置错误、记录信息、取消操作等等。

数据成员	数据类型	描述
args_handle	void*	服务器所提供参数的句柄。
table	a_v4_extfn_table *	指向打开的结果集表。调用 a_v4_extfn_proc_context open_result_set 后对其赋值。
user_data	void*	使用外部例程所需的任何环境数据皆可对该指针赋值。
server_internal_use	void*	仅供内部使用。

描述

a_v4_extfn_table_context 结构充当生产者和消费者的中间层，在二者需要单独的格式时帮助管理数据。

UDF 可使用 a_v4_extfn_table_context 从输入 TABLE 参数中读取行。服务器或其它 UDF 可使用 a_v4_extfn_table_context 从 UDF 的结果表中读取行。

通过执行 a_v4_extfn_table_context 的方法，服务器有机会解决阻抗不匹配问题。

fetch_into

fetch_into v4 API 方法将数据提取至指定的行块中。

声明

```
short fetch_into(
a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block *)
```

用法

如果生产者不知道应该如何在内存中安排数据，则 fetch_into 方法将十分有用。当消耗程序拥有特定格式的传输区时，此方法将被用作入口点。fetch_into() 函数将提取的行写入所提供的行块。此方法是 a_v4_extfn_table_context 结构的一部分。

当消耗程序拥有数据传输区内存并且请求生产者使用该区时，请使用 fetch_into 方法。当消耗程序关注于数据传输区的设置方法， 并且由生产者将必要的数据复制到该区时， 请您使用 fetch_into 方法。

参数

参数	描述
cntxt	从 open_result_set API 获取的表上下文对象
row_block	用于存储提取结果的行块对象

返回
如果成功则返回 1，否则返回 0。

如果 UDF 返回 1，则消耗程序知道还有未提取的行，并将再次调用 `fetch_into` 方法。而如果 UDF 返回 0，则表示所有行已提取完毕，不再需要调用 `fetch_into` 方法。

请考虑如下过程定义，这是一个 TPF 函数的示例，它消耗输入参数表并将其生成为结果表。两者分别为通过 `get_value` 和 `set_value` v4 API 方法获取和返回的 SQL 值的实例。

```
CREATE PROCEDURE FETCH_EX( IN a INT, INT b TABLE( c1 INT ) )
    RESULT SET ( rc INT )
```

此过程定义包含两个表对象：

- 命名为 `b` 的输入 TABLE 参数
- 返回的结果集表

下面的示例描述了调用方（在本例中为服务器）将如何提取输出表。服务器可能会决定使用 `fetch_into` 方法。调用实体（在本例中为 TPF）提取输入表。TPF 决定将使用哪一个提取 API。

```
SELECT rc from FETCH_EX( 1, TABLE( SELECT c1 from TABLE ) )
```

此示例显示了在读取/使用输入表前，必须先通过 `a_v4_extfn_proc` 结构上的 `open_result_set` API 建立表上下文。`open_result_set` 需要可通过 `get_value` API 获取的表对象。

```
    an_extfn_value    arg;
    ctx->get_value( args_handle, 3, &arg );

    if( arg.type != DT_EXTFN_TABLE ) {
        // handle error
    }

    a_v4_extfn_table_context    *rs = NULL;
    a_v4_extfn_table         *inTable = arg.data;
    ctx->open_result_set( ctx, inTable, &rs );
```

创建表上下文后，`rs` 结构将执行 `fetch_into` API 并读取行。

```
a_v4_extfn_row_block    *rb = // get a row block to hold a series of
INT values.
rs->fetch_into( rs, &rb )  // fetch the rows.
```

生成结果表的行前，必须先通过 `a_v4_extfn_proc_context` 结构上的 `set_value` API 创建表对象并将其返回到调用者。

此示例显示了表 UDF 必须创建 `a_v4_extfn_table` 结构的实例。每次对表 UDF 的调用都应返回一个单独的 `a_v4_extfn_table` 结构的实例。表包含跟踪当前行和要生成的行数的状态字段。表的状态可存储为实例的字段。

```
    typedef struct rg_table : a_v4_extfn_table {
        a_sql_uint32        rows_to_generate;
        a_sql_uint32        current_row;
    } my_table;
```

在下面的示例中，每生成一行，则递增**current_row**，直至达到所生成的行的数量，此时 fetch_into 将返回 false 以表明到达文件末尾。 消耗程序执行由表 UDF 所实现的 fetch_into API。作为调用 fetch_into 方法的一部分，消耗程序提供了表上下文以及用于存储提取结果的行块。

```
     rs->fetch_into( rs, &rb )

short UDF_CALLBACK my_table_func_fetch_into(
    a_v4_extfn_table_context *tctx,
    a_v4_extfn_row_block *rb)
/*****************************************/
{
    my_table *myTable = tctx->table;

    if( rgTable->current_row < rgTable->rows_to_generate ) {
            // Produce the row...
    rgTable->current_row++;
    return 1;
    }

    return 0;
}
```

fetch_block

fetch_block v4 API 方法对行块执行提取操作。

声明

```
short fetch_block(
a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block **row_block)
```

用法

当消耗程序不需要特殊格式的数据时，使用 fetch_block 方法作为入口点。 fetch_block 请求生产者创建数据传输区，并提供指向该区的指针。消耗程序拥有数据传输区内存，并且负责从该区中复制数据。

如果消耗程序不需要特定布局，则 fetch_block 方法更为有效。 fetch_block 调用为可消耗块设置了 fetch_block，并且将该块传递至下一次 fetch_block 调用。此方法是 a_v4_extfn_table_context 结构的一部分。

参数

参数	描述
cntxt	表上下文对象。

参数	描述
row_block	输入／输出参数。首次调用应该总是指向一个空 **row_block**。

调用 fetch_block 且 **row_block** 指向 NULL 时，UDF 必须分配 a_v4_extfn_row_block 结构。

返回
如果成功则返回 1，否则返回 0。

如果 UDF 返回 1，则消耗程序知道还有未提取的行，并将再次调用 fetch_block 方法。而如果 UDF 返回 0 ，则表示所有行已提取完毕，不再需要调用 fetch_block 方法。

请考虑如下过程定义，这是一个 TPF 函数的示例，它消耗输入参数表并将其生成为结果表。两者分别为通过 get_value 和 set_value v4 API 方法获取和返回的 SQL 值的实例。

```
CREATE PROCEDURE FETCH_EX( IN a INT, INT b TABLE( c1 INT ) )
    RESULT SET ( rc INT )
```

此过程定义包含两个表对象:

- 命名为 b 的输入 TABLE 参数
- 返回的结果集表

下面的示例描述了调用方（在本例中为服务器）将如何提取输出表。服务器可能会决定使用 fetch_block 方法。调用实体（在本例中为 TPF）提取输入表，并决定将使用哪一个提取 API。

```
SELECT rc from FETCH_EX( 1, TABLE( SELECT c1 from TABLE ) )
```

此示例显示了在读取/使用输入表前，必须先通过 a_v4_extfn_proc 结构上的 open_result_set API 建立表上下文。open_result_set 需要可通过 get_value API 获取的表对象。

```
    an_extfn_value    arg;
    ctx->get_value( args_handle, 3, &arg );

    if( arg.type != DT_EXTFN_TABLE ) {
        // handle error
    }

    a_v4_extfn_table_context    *rs = NULL;
    a_v4_extfn_table                    *inTable = arg.data;
    ctx->open_result_set( ctx, inTable, &rs );
```

创建表上下文后，rs 结构将执行 fetch_block API 并读取行。

```
a_v4_extfn_row_block    *rb =  // get a row block to hold a series of
INT values.
rs->fetch_block( rs, &rb )  // fetch the rows.
```

生成结果表的行前，必须先通过 a_v4_extfn_proc_context 结构上的 set_value API 创建表对象并将其返回到调用者。

此示例显示了表 UDF 必须创建 a_v4_extfn_table 结构的实例。每次对表 UDF 的调用都应返回一个单独的 a_v4_extfn_table 结构的实例。表包含跟踪当前行和要生成的行数的状态字段。表的状态可存储为实例的字段。

```
    typedef struct rg_table : a_v4_extfn_table {
        a_sql_uint32        rows_to_generate;
        a_sql_uint32        current_row;
    } my_table;
```

rewind

使用 rewind 第 4 版 API 方法从表的开头重新启动结果集。

声明

```
short rewind(
       a_v4_extfn_table_context           *cntxt,
       )
```

用法

对已打开的结果集调用 rewind 方法，即可回绕到表的开头。如果 UDF 要回绕输入表，则必须在 **EXTFNAPIV4_STATE_OPTIMIZATION** 状态下用 **EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND** 参数通知生产程序。

rewind() 是可选条目。如果是空表，则不支持回绕。否则， rewind() 条目会在表的开头重新启动结果集。

参数

参数	描述
cntxt	表上下文对象

返回

如果成功则返回 1，否则返回 0。

get_blob

请使用 get_blob v4 API 方法从指定列返回 BLOB 对象。

声明

```
 short get_blob(
a_v4_extfn_table_context *cntxt,
a_v4_extfn_column_data *col,
a_v4_extfn_blob **blob
)
```

用法

BLOB 作为输出参数返回，并为调用方所有。仅对包含 BLOB 对象的列调用此方法。

请使用帮助程序宏 EXTFN_COL_IS_BLOB 来确定列是否包含 BLOB 对象。这是头文件 extfnapiv4.h 中的 EXTFN_COL_IS_BLOB 声明：

```
#define EXTFN_COL_IS_BLOB(c, n)         (c[n].blob_handle != NULL)
```

参数

参数	描述
cntxt	表上下文对象
col	为其获取 BOLB 的列数据指针
blob	成功时，则包含与列相关的 BLOB 对象

返回

如果成功则返回 1，否则返回 0。

表函数 (a_v4_extfn_table_func)

消耗程序使用 a_v4_extfn_table_func 结构检索来自生产者的结果。

实现

```
typedef struct a_v4_extfn_table_func {
//    size_t struct_size;

    /*  Open a result set. The UDF can allocate any resources needed
for the result set.
    */
    short (UDF_CALLBACK *_open_extfn)(a_v4_extfn_table_context *);

   /*  Fetch rows into a provided row block. The UDF should implement
this method if it does
        not have a preferred layout for its transfer area.
    */
   short (UDF_CALLBACK *_fetch_into_extfn)(a_v4_extfn_table_context
*, a_v4_extfn_row_block

*row_block);

   /*  Fetch a block that is allocated and configured by the UDF. The
UDF should implement this
        method if it has a preferred layout of the transfer area.
    */
    short (UDF_CALLBACK *_fetch_block_extfn)
(a_v4_extfn_table_context *, a_v4_extfn_row_block

**row_block);
```

```
   /*  Restart a result set at the beginning of the table. This is an
optional entry point.
    */
    short (UDF_CALLBACK *_rewind_extfn)(a_v4_extfn_table_context *);

    /*  Close a result set. The UDF can release any resources
allocated for the result set.
    */
    short (UDF_CALLBACK *_close_extfn)(a_v4_extfn_table_context *);

    /* The following fields are reserved for future use and must be
initialized to NULL. */
    void *_reserved1_must_be_null;
    void *_reserved2_must_be_null;

} a_v4_extfn_table_func;
```

方法总结

方法	数据类型	描述
_open_extfn	无类型	服务器调用该方法，通过打开结果集来启动行提取操作。UDF 可以分配结果集所需的任何资源。
_fetch_into_extfn	短整型	将行提取至所提供的行块中。如果 UDF 的传输区未设置首选布局，则它将执行此方法。
_fetch_block_extfn	短整型	提取由 UDF 所分配和配置的块。如果 UDF 的传输区设置了首选布局，则它将执行此方法。
_rewind_extfn	无类型	可选函数，服务器调用该函数从表的开头重启提取操作。
_close_extfn	无类型	服务器调用该方法，通过关闭结果集来终止行提取操作。UDF 可以释放已分配给结果集的任何资源。
_reserved1_must_be_null	无类型	留作将来使用。必须初始化为 NULL。
_reserved1_must_be_null	无类型	留作将来使用。必须初始化为 NULL。

描述

a_v4_extfn_table_func 结构定义了从表提取结果的方法。

_open_extfn

服务器会调用 _open_extfn 第 4 版 API 方法，以便开始提取行。

声明

```
void _open_extfn(
     a_v4_extfn_table_context *cntxt,
)
```

用法

UDF 用此方法打开结果集并分配向服务器发送结果所需的所有资源（如流）。

参数

参数	描述
cntxt	过程上下文对象

_fetch_into_extfn

_fetch_into_extfn 第 4 版 API 方法用于将行提取到所提供的行块中。

声明

```
short _fetch_into_extfn(
    a_v4_extfn_table_context *cntxt,
    a_v4_extfn_row_block *row_block
    )
```

用法

如果 UDF 的传输区域没有首选布局， 则 UDF 应该实现此方法。

参数

参数	描述
cntxt	过程上下文对象
row_block	要提取到的行块对象。

返回

如果成功则返回 1，否则返回 0。

_fetch_block_extfn

_fetch_block_extfn 第 4 版 API 方法用于提取由 UDF 分配和配置的块。

声明

```
short _fetch_block_extfn(
a_v4_extfn_table_context *cntxt,
```

```
a_v4_extfn_row_block **
)
```

用法
如果 UDF 的传输区域有首选布局，则 UDF 应该实现此方法。

参数

参数	描述
cntxt	过程上下文对象
row_block	要提取到的行块对象

返回
如果成功则返回 1，否则返回 0。

_rewind_extfn

_rewind_extfn 第 4 版 API 方法用于从表的开头重新启动结果集。

声明

```
void _rewind_extfn(
a_v4_extfn_table_context *cntxt,
)
```

用法
此函数是可选条目。回绕到结果表开头后，UDF 会实现 `_rewind_extfn` 方法。仅当 UDF 能以高效经济的方式提供回绕功能时，UDF 才会该考虑实现这种方法。

如果 UDF 决定实现 `_rewind_extfn` 方法，则应该在 **EXTFNAPIV4_STATE_OPTIMIZATION** 状态下通过设置参数 0 的 **EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND** 参数告知消耗程序。

UDF 可以决定不提供回绕功能，这种情况下服务器会进行补偿并提供回绕功能。

注意： 服务器可选择不调用 `_rewind_extfn` 方法来执行回绕。

参数

参数	描述
cntxt	过程上下文对象

返回
无返回值。

_close_extfn

服务器调用 `_close_extfn` v4 API 方法来终止对行的提取操作。

声明

```
void _close_extfn(
     a_v4_extfn_table_context *cntxt,
)
```

用法

当提取操作执行完毕之后，UDF 使用此方法关闭结果集，并释放分配给该结果集的所有资源。

参数

参数	描述
cntxt	过程上下文对象

在应用程序中使用 SQL

本节提供关于在应用程序中使用 SQL 的信息。

在应用程序中执行 SQL 语句

在应用程序中使用 SQL 语句的方式取决于您使用的应用程序开发工具和编程接口。

- **ADO.NET** – 可使用各种 ADO.NET 对象执行 SQL 语句。SACommand 对象就是一个示例：

```
SACommand cmd = new SACommand(
     "DELETE FROM Employees WHERE EmployeeID = 105", conn );
cmd.ExecuteNonQuery();
```

- **ODBC** – 如果您直接对 ODBC 编程接口编写代码，那么您的 SQL 语句将以函数调用的形式出现。例如，下面的 C 函数调用将执行 DELETE 语句：

```
SQLExecDirect( stmt,
    "DELETE FROM Employees
     WHERE EmployeeID = 105",
    SQL_NTS );
```

- **JDBC** – 如果使用的是 JDBC 编程接口，那么您可以通过调用语句对象的方法来执行 SQL 语句。例如：

```
stmt.executeUpdate(
    "DELETE FROM Employees
     WHERE EmployeeID = 105" );
```

- **嵌入式 SQL** – 如果您使用的是嵌入式 SQL，那么应在 C 语言 SQL 语句前用关键字 EXEC SQL 作为前缀。然后，代码在编译之前通过预处理器处理。例如：

```
EXEC SQL EXECUTE IMMEDIATE
 'DELETE FROM Employees
  WHERE EmployeeID = 105';
```

- **Sybase Open Client** – 如果使用的是 Sybase Open Client 接口，那么您的 SQL 语句以函数调用形式出现。例如，下面的两个调用将执行 DELETE 语句：

```
ret = ct_command( cmd, CS_LANG_CMD,
                   "DELETE FROM Employees
                    WHERE EmployeeID=105"
                  CS_NULLTERM,
                  CS_UNUSED);
ret = ct_send(cmd);
```

有关如何在应用程序中使用 SQL 的详细信息，请参见开发工具文档。如果使用的是 ODBC 或 JDBC，则请查阅软件开发工具包中有关这些接口的信息。

数据库服务器内的应用程序
在许多方面，存储过程和触发器都充当在数据库服务器内运行的应用程序或应用程序的组成部分。您也可以在存储过程中使用此处的许多技术。

数据库中的 Java 类可以采用与服务器之外的 Java 应用程序同样的方式来使用 JDBC 接口。本节讨论了 JDBC 的一些方面。

预准备语句

每次向数据库发送语句时，数据库服务器必须执行以下步骤：

- 服务器必须分析语句并将其转换为内部形式。此过程有时称为*准备*语句。
- 服务器必须验证对数据库对象的所有引用的正确性，例如通过检查查询中提到的列是否确实存在。
- 如果语句涉及连接或子查询，则查询优化程序会生成访问计划。
- 在所有的这些步骤都已经执行之后，它会执行该语句。

重复使用预准备语句可以改善性能
如果反复使用同一语句（例如在表中插入多行），则重复准备语句会产生大量的不必要的开销。为消除这种开销，一些数据库编程接口提供了使用预准备语句的方法。*预准备*语句是包含一系列占位符的语句。当要执行语句时，只需给占位符赋值，而不是再次重新准备整个语句。

预准备语句尤其适用于执行多个类似操作的情况，如插入多行。

一般来讲，使用预准备语句需要执行下面的步骤：

- **准备语句** – 在这一步，一般都要为语句提供一些占位符而非实际的值。
- **反复执行预准备语句** – 在此步骤中，每次执行语句时都提供要使用的值。不必每次都准备语句。
- **删除语句** – 在此步骤中，释放与预准备语句关联的资源。一些编程接口会自动处理此步骤。

不要准备那些只使用一次的语句
通常，不应该准备那些将只执行一次的语句。单独的准备和执行会产生轻微的性能损失，并且它会给应用程序带来不必要的复杂性。

然而，在一些接口中，确实需要准备一个语句以将它与游标关联。

用来准备和执行语句的调用并不是 SQL 的组成部分，并且它们也因接口而异。SAP Sybase IQ 中的每一个编程接口都提供了使用预准备语句的方法。

预准备语句概述

本节简短概述如何使用预准备语句。总的过程都是相同的，但细节方面将因接口而异。通过将不同接口中使用预准备语句的方式加以比较，即可看出这一点。

要使用预准备语句，通常需执行以下任务：

1. 准备语句。
2. 绑定将在语句中保存值的参数。

3. 给语句中的绑定参数赋值。
4. 执行语句。
5. 根据需要重复执行步骤 3 和 4。
6. 完成之后删除语句。在 JDBC 中，Java 垃圾回收机制会删除语句。

在 ADO.NET 中使用预准备语句
要在 ADO.NET 中使用预准备语句，通常需执行以下任务：

1. 创建一个保存语句的 SACommand 对象：
   ```
   SACommand cmd = new SACommand(
        "SELECT * FROM Employees WHERE Surname = ?", conn );
   ```
2. 为语句中的参数声明数据类型。
   使用 SACommand.CreateParameter 方法。
   ```
   SAParameter param = cmd.CreateParameter();
   param.SADbType = SADbType.Char;
   param.Direction = ParameterDirection.Input;
   param.Value = "Smith";
   cmd.Parameters.Add(param);
   ```
3. 使用 Prepare 方法准备此语句。
4. 执行该语句：
   ```
   SADataReader reader = cmd.ExecuteReader();
   ```

有关使用 ADO.NET 准备语句的示例，请参见 `%ALLUSERSPROFILE%\SybaseIQ \samples\SQLAnywhere\ADO.NET\SimpleWin32` 中的源代码。

在 ODBC 中使用预准备语句
要在 ODBC 中使用预准备语句，通常需执行以下任务：

1. 使用 SQLPrepare 准备语句。
2. 使用 SQLBindParameter 绑定语句参数。
3. 使用 SQLExecute 执行语句。
4. 使用 SQLFreeStmt 删除语句。

有关使用 ODBC 准备语句的示例，请参见 `%ALLUSERSPROFILE%\SybaseIQ \samples\SQLAnywhere\ODBCPrepare` 中的源代码。

在 JDBC 中使用预准备语句
要在 JDBC 中使用预准备语句，通常需执行以下任务：

1. 使用连接对象的 prepareStatement 方法准备语句。此步骤返回预准备语句对象。
2. 使用预准备语句对象的适当 `set`*Type* 方法设置语句参数。这里的 *Type* 是指派的数据类型。
3. 使用预准备语句对象的适当方法执行语句。对于插入、更新和删除，使用 executeUpdate 方法。

有关使用 JDBC 准备语句的示例，请参见源代码文件 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC\JDBCExample.java。

在嵌入式 SQL 中使用预准备语句
要在嵌入式 SQL 中使用预准备语句，通常需执行以下任务：

1. 使用 EXEC SQL PREPARE 语句准备语句。
2. 给语句中的参数赋值。
3. 使用 EXE SQL EXECUTE 语句执行语句。
4. 使用 EXEC SQL DROP 语句释放与该语句关联的资源。

在 Open Client 中使用预准备语句
要在 Open Client 中使用预准备语句，通常需执行以下任务：

1. 使用带 CS_PREPARE 类型参数的 ct_dynamic 函数准备语句。
2. 使用 ct_param 函数设置语句参数。
3. 使用带 CS_EXECUTE 类型参数的 ct_dynamic 执行语句。
4. 使用带 CS_DEALLOC 类型参数的 ct_dynamic 释放与该语句关联的资源。

游标用法

当您在应用程序中执行查询时，结果集由若干行组成。通常，您执行查询之前并不知道应用程序要接收多少行。游标提供了在应用程序中处理查询结果集的方式。

使用游标的方式以及可供使用的游标种类取决于所使用的编程接口。

SAP Sybase IQ 提供了几个系统过程来帮助确定哪些游标用于连接，以及它们包含的内容：

sa_list_cursors 系统过程
sa_describe_cursor 系统过程
sa_copy_cursor_to_temp_table 系统过程

利用游标可在任何编程接口内执行下面的任务：

- 在查询的结果内循环。
- 在结果集内的任何点的基础数据上执行插入、更新和删除。

此外，一些编程接口可使您使用特殊功能调整结果集返回到应用程序的方式，为您的应用程序提供了实实在在的性能优点。

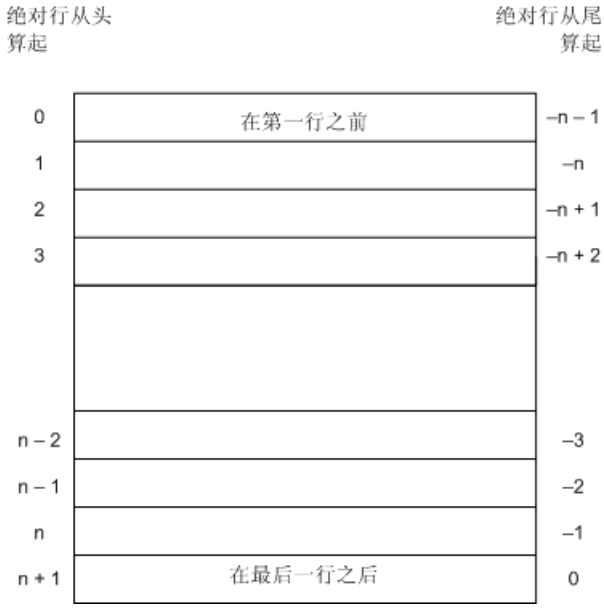
游标

游标是与结果集相关联的名称。结果集可从 SELECT 语句或存储过程调用获得。

游标是结果集上的句柄。任何时候，游标在结果集内都有定义明确的位置。利用游标您可以检查并可能一次一行地操纵数据。SAP Sybase IQ 游标支持在查询结果中向前和向后移动。

游标位置
游标可以位于以下位置：

- 在结果集的第一行之前。
- 在结果集的某一行上。
- 在结果集的最后一行之后。



游标位置和结果集在数据库服务器中进行维护。客户端会一次一行或一次多行地*读取*行，以便显示和处理。不需要将整个结果集传输到客户端。

使用游标的优点

尽管数据库应用程序中不需要使用服务器端游标，但使用游标确实具有多个好处。服务器端游标优于客户端游标的原因如下：

- **响应时间** – 服务器端游标不需要在客户端读取第一行之前对整个结果集进行汇编。客户端游标需要在客户端读取第一行之前获取整个结果集并将其传输到客户端。

- **客户端内存** – 如果结果集较大，要在客户端获取整个结果集，就会需要更多的内存。
- **并发控制** – 如果对数据进行多次更新而不在应用程序中使用服务器端游标，则必须向数据库服务器分别发送多条 SQL 语句（如 INSERT、UPDATE 或 DELETE）来应用这些更改。如果数据库中的任意对应行在结果集传送到客户端之后又发生了更改，这样做便会增加发生并发问题的可能性。因此，其它客户端所做的更新可能会丢失。

  服务器端游标可充当指向基础数据的指针，允许您通过设置适当隔离级别的方式对客户端进行的任何更改强制执行适当的并发约束。

游标原则

要在 ADO.NET、ODBC、JDBC 或 Open Client 中使用游标，请执行下列一般步骤：

1. 准备和执行语句。
   使用接口的常用方法执行语句。您可以先准备语句，然后再执行该语句；也可以直接执行语句。
   使用 ADO.NET 时，只有 SACommand.ExecuteReader 方法才返回游标。此命令提供只读、只进游标。
2. 进行测试，看一看语句是否返回结果集。
   在执行创建结果集的语句时，游标被隐式打开。在打开游标时，游标定位在结果集的第一行之前。
3. 读取结果。
   虽然简单读取操作会将游标移到结果集中的下一行，但是 SAP Sybase IQ 允许在结果集内进行更复杂的移动。
4. 关闭游标。
   当您用完游标之后，将它关闭以释放关联的资源。
5. 释放语句。
   如果您使用了预准备语句，则请释放它以回收内存。

在嵌入式 SQL 中使用游标的方法不同于在其它接口中使用游标的方法。要在嵌入式 SQL 中使用游标，请执行下列一般步骤：

1. 准备语句。
   游标通常使用语句句柄而不是字符串。要使用句柄，您需要准备语句。
2. 声明游标。
   每个游标都将引用单个 SELECT 或 CALL 语句。当声明游标时，应声明游标的名称和它所引用的语句。
3. 打开游标。
   对于 CALL 语句，打开游标会执行过程直到即将获得第一行时为止。
4. 读取结果。

虽然简单读取操作会将游标移到结果集中的下一行，但是 SAP Sybase IQ 允许在结果集内进行更复杂的移动。声明游标的方式决定了可以使用哪些读取操作。

5. 关闭游标。
   当您用完游标之后，应将它关闭。这会释放与游标关联的所有资源。
6. 删除语句。
   要释放与语句关联的内存，必须删除语句。

游标定位

当游标打开之后，它位于第一行之前。您可以将游标移到以查询结果的开头或结尾作为参照的一个绝对位置，或者移到以当前游标位置作为参照的一个相对位置。如何更改游标位置以及可以执行什么操作的具体情况受编程接口约束。

在游标中可读取的行位置编号受整数的大小制约。您最多可以读取到第 2147483646 行，这个数字比可以在整数中保存的值小 1。在使用负数（从末尾开始计算行数）时，您最多可以读取到的行数比整数中可保存的最大负值大 1。

您可以使用特殊定位的更新操作和删除操作来更新或删除位于游标当前位置的行。如果游标定位在第一行之前或最后一行之后，则会返回错误，指示没有相应的游标行。

注意： 对敏感性未定型游标进行的插入和某些更新会导致游标定位发生问题。除非 SELECT 语句中有 ORDER BY 子句，否则 SAP Sybase IQ 不会将插入的行放在游标内可预知的位置。有时，插入的行要等到关闭并再次打开游标后才会出现。对于 SAP Sybase IQ，如果必须创建工作表才能打开游标，则会出现这种情况。

UPDATE 语句可能导致行在游标中移动。如果游标有使用现有索引的 ORDER BY 子句（不必创建工作表），会发生这种情况。使用 STATIC SCROLL 游标会缓解这些问题，但需要更多的内存和处理。

打开游标时的游标行为

您可以在打开游标时配置游标行为的以下几个方面：

- **隔离级别** – 您可以将游标上操作的隔离级别显式设置为不同于事务的当前隔离级别。为此，请设置 isolation_level 选项。
- **保存** – 缺省情况下，嵌入式 SQL 中的游标在事务结束时关闭。游标 WITH HOLD 打开之后，您可以让它保持打开状态，直到连接结束，或者直到您将它显式关闭。缺省情况下，ADO.NET、ODBC、JDBC 和 Open Client 在事务结束时会让游标保持打开状态。

通过游标读取行

使用游标处理查询的结果集的最简单方式是在结果集的所有行中循环，直到没有行为止。可通过执行以下步骤来完成此任务：

1. 声明并打开游标（嵌入式 SQL），或者执行返回结果集的语句（ODBC、JDBC、Open Client），或者执行 SADataReader 对象 (ADO.NET)。

2. 继续读取下一行，直到收到 `Row Not Found` 错误。
3. 关闭游标。

用于读取下一行的方法取决于您使用的接口。例如：

- **ADO.NET** – 使用 SADataReader.Read 方法。
- **ODBC** – SQLFetch、SQLExtendedFetch 或 SQLFetchScroll 让游标前进到下一行并返回数据。
- **JDBC** – ResultSet 对象的 next 方法让游标前进并返回数据。
- **嵌入式 SQL** – FETCH 语句执行相同的操作。
- **Open Client** – ct_fetch 函数将游标前进到下一行并返回数据。

多行读取

请不要将多行读取与预取行混淆。多行读取由应用程序执行，而预取则对应用程序透明，并可以提供类似的性能改进。一次读取多行可以提高性能。

多行读取

某些接口提供了一次将多行读取到数组中的下几个字段的方法。通常，您执行单独的读取操作越少，服务器必须响应的单个请求也就越少，性能也就越好。修改后的检索多行的 FETCH 语句有时也称为*宽读取*。使用多行读取的游标有时称为*块状游标*或*胖游标*。

使用多行读取

- 在 ODBC 中，您可以通过设置 SQL_ATTR_ROW_ARRAY_SIZE 或 SQL_ROWSET_SIZE 属性来设置每一次调用 SQLFetchScroll 或 SQLExtendedFetch 返回的行数。
- 在嵌入式 SQL 中，FETCH 语句使用 ARRAY 子句控制一次读取的行数。
- Open Client 和 JDBC 不支持多行读取。但它们可以使用预取。

可滚动游标

ODBC 和嵌入式 SQL 提供了使用可滚动游标和动态可滚动游标的方法。这些方法使您能够在结果集中一次向前或向后移动多行。

JDBC 和 Open Client 接口不支持可滚动游标。

预取不适用于可滚动操作。例如，读取相反方向的某行并不能预取前面的多行。

用于修改行的游标

游标的用途不仅仅是读取查询的结果集。您还可以在处理游标时修改数据库中的数据。这些操作通常称为*定位*插入、更新和删除操作，或者如果操作是插入操作，则称为 PUT 操作。

并非所有的查询结果集都允许定位更新和删除。如果您对不可更新的视图执行查询，则基础表不会发生更改。此外，如果查询涉及连接，则必须指定您希望从哪一个表删除，或者您希望更新哪些列，何时执行操作。

只有在表中的某些非插入列允许 NULL 或有缺省值的情况下，才能通过游标执行插入。

将多行插入对值敏感的游标（由键集决定的游标）时，新插入的行出现在游标结果集的末尾处。即使这些行与查询的 WHERE 子句不匹配，或者 ORDER BY 子句通常将它们放置在结果集的其它位置，这些行也会出现在最后。此行为与编程接口无关。例如，当使用嵌入式 SQL PUT 语句或 ODBC SQLBulkOperations 函数时就会是这样。通过选择游标中的最后一行，可以找到最新插入行的 AUTOINCREMENT 列的值。例如，在嵌入式 SQL 中，可使用 `FETCH ABSOLUTE -1 cursor-name` 来获取值。此行为的结果是，对值敏感的游标的首次多行插入的开销会很大。

ODBC、JDBC、嵌入式 SQL 和 Open Client 允许使用游标进行数据操作，但 ADO.NET 不允许。对于 Open Client，您可以删除和更新行，但您只能在单表查询上插入行。

可以从哪个表中删除行?
如果您试图通过游标执行定位删除，那么请按如下所示的方法确定从哪个表删除行：

1. 如果 DELETE 语句中未包括 FROM 子句，那么游标必须只在单个表上。
2. 如果游标用于连接式查询（包括使用含有连接的视图），则必须使用 FROM 子句。只会删除指定表的当前行，连接中涉及的其它表不会受到影响。
3. 如果包括了 FROM 子句，但未指定表所有者，那么指定表为第一个与其值相匹配的相关名。
4. 如果给出了相关名，就用该相关名来确定指定表的名称。
5. 如果没有给出相关名，那么指定表的名称必须是游标中可明确标识的表名。
6. 如果给出的 FROM 子句中指定了表的所有者，那么指定表的名称必须像游标中的表名一样是明确可标识的。
7. 已定位的 DELETE 语句可以用于在某视图上打开的游标，只要该视图可更新即可。

可更新的语句

本节介绍 SELECT 语句中的子句如何影响可更新语句和游标。

只读语句的可更新性
在游标声明中指定 FOR READ ONLY 或在语句中包含 FOR READ ONLY 子句可使语句为只读语句。换句话说，FOR READ ONLY 子句或使用客户端 API 时适当的只读游标声明会覆盖任何其它可更新性指定。

如果 SELECT 语句的最外层块中包含 ORDER BY 子句且该语句未指定 FOR UPDATE，则游标为只读的。如果 SQL SELECT 语句指定 FOR XML，则游标为只读的。否则，游标是可更新的。

可更新语句和并发控制

对于可更新语句，SAP Sybase IQ 提供了优化和保守两种游标并发控制机制，以便确保滚动操作中结果集保持一致。虽然这两种机制各有各的语义和利弊，它们都可作为使用 INSENSITIVE 游标或快照隔离的替代方法。

FOR UPDATE 的说明可影响游标是否可更新。然而，在 SAP Sybase IQ 中，FOR UPDATE 语法对并发控制没有任何其它影响。如果使用其它参数指定 FOR UPDATE，SAP Sybase IQ 将按如下方式变更语句的处理过程，以合并两个并发控制选项之一：

- **保守** – 对于在游标的结果集中读取的所有行，数据库服务器将获得意图行锁，以防止任何其它事务更新这些行。
- **优化** – 数据库服务器使用的游标类型更改为由键集决定的游标（对行成员资格不敏感，对值敏感），以便在此事务或任何其它事务修改或删除结果中的行时通知应用程序。

通过带有 DECLARE CURSOR 或 FOR 语句的选项，或特定编程接口的并发设置 API，可在游标级别指定保守或优化并发。如果语句是可更新的，而游标未指定并发控制机制，则使用语句的说明。语法如下：

- **FOR UPDATE BY LOCK** – 数据库服务器在结果集的读取行上获得意图行锁。这些锁是长期锁，会一直保留到事务 COMMIT 或 ROLLBACK 执行。
- **FOR UPDATE BY { VALUES | TIMESTAMP }** – 数据库服务器利用键集决定的游标，这样，如果在滚动结果集过程中修改或删除了行，则会通过该游标通知应用程序。

限制可更新语句

FOR UPDATE (*column-list*) 强制实行以下限制：在随后的 UPDATE WHERE CURRENT OF 语句中只能修改指定的结果集属性。

取消游标操作

您可以通过接口函数取消请求。如果您取消正在执行游标操作的请求，则游标的位置是不确定的。取消请求之后，您必须按照其绝对位置给游标定位，或者将它关闭。

游标类型

本节介绍 SAP Sybase IQ 游标和 SAP Sybase IQ 支持的编程接口的可用选项之间的映射。

游标的可用性

并非所有的接口都为所有的游标类型提供支持。

- ADO.NET 只提供只进、只读游标。
- ADO/OLE DB 和 ODBC 支持所有的游标类型。
- 嵌入式 SQL™ 支持所有的游标类型。
- 对于 JDBC：
  - SQL Anywhere JDBC 驱动程序支持 JDBC 4.0 规范并允许不敏感、敏感和只进敏感性未定型游标的声明。
  - jConnect 支持不敏感、敏感和只进敏感性未定型游标的声明，采用的方式与 SQL Anywhere JDBC 驱动程序相同。但 jConnect 的底层实现仅支持敏感性未定型游标语义。
- Sybase Open Client 仅支持敏感性未定型游标。此外，在使用可更新的非唯一游标时，会产生严重的性能下降。

游标属性

您可以显式或者隐式从编程接口请求游标类型。不同的接口库提供不同的游标类型选择。例如，JDBC 和 ODBC 指定了不同的游标类型。

每个游标类型都由多个特性来定义：

- **唯一性** – 对游标进行唯一性声明将强制查询返回唯一标识每一行所需要的所有列。通常，这意味着返回主键中的所有列。所需要的但未指定的任何列都要添加到结果集中。缺省游标类型是非唯一的。
- **可更新性** – 声明为只读的游标不能在定位更新或删除操作中使用。缺省游标类型是可更新的。
- **可滚动性** – 您可以这样声明游标：当您在结果集中移动时，它表现出不同的行为。某些游标可以只读取当前行或后面的行。其它一些游标可以在结果集中来回移动。
- **敏感性** – 通过游标也许可以看到对数据库的更改，也许看不到。

这些特性可能会对性能以及数据库服务器内存的使用产生明显的副作用。

SAP Sybase IQ 可提供具有这些特性的各种组合的游标。请求指定类型的游标时，SAP Sybase IQ 会尝试匹配这些特性。

某些情况下，并非所有特性都可以提供。例如，SAP Sybase IQ 中的不敏感游标必须是只读的。如果您的应用程序请求可更新的不敏感游标，则提供给它的将是其它类型的游标（对值敏感）。

书签和游标

ODBC 提供*书签*或值，用以标识游标中的行。对于对值敏感的游标和不敏感游标，SAP Sybase IQ 支持书签。例如，ODBC 游标类型 SQL_CURSOR_STATIC 和

SQL_CURSOR_KEYSET_DRIVEN 支持书签，而游标类型 SQL_CURSOR_DYNAMIC 和 SQL_CURSOR_FORWARD_ONLY 不支持书签。

块状游标

ODBC 提供了一种称为块状游标的游标类型。当您使用 BLOCK 游标时，可以使用 SQLFetchScroll 或 SQLExtendedFetch 读取一整块行，而不是单行。块状游标的行为与嵌入式 SQL ARRAY 读取完全相同。

SAP Sybase IQ 目录存储游标

任何 SAP Sybase IQ 目录存储游标一经打开，就会有一个关联的结果集。游标在一段时间内保持打开状态。在这段时间内，与该游标关联的结果集可能被更改，要么是通过游标本身更改，要么是根据隔离级别要求被其它事务更改。有些游标允许看到对基础数据的更改，而其它游标却不会反映出这些更改。对基础数据的更改的敏感性会导致不同的游标行为，或称为*游标敏感性*。

SAP Sybase IQ 目录存储为游标提供了各种敏感性特性。本节介绍什么是敏感性，并介绍游标的敏感性特性。

成员资格、顺序和值更改
对基础数据的更改会在下列几个方面影响游标的结果集：

- **成员资格** – 结果集中的行集，如同它们的主键值所标识的那样。
- **顺序** – 结果集中行的顺序。
- **值** – 结果集中行的值。

例如，请看以下带有雇员信息的简单表（EmployeeID 是主键列）：

EmployeeID	Surname
1	Whitney
2	Cobb
3	Chin

下列查询上的游标将按主键顺序返回表中的所有结果：

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

通过添加新行或删除行可更改结果集的成员资格。值可以通过更改表中的名称来进行更改。顺序可以通过更改雇员的主键值来进行更改。

可见的和不可见的更改
根据隔离级别要求，游标的结果集的成员资格、顺序和值可以在游标打开之后进行更改。根据所使用的游标类型，通过应用程序所看到的结果集可能会更改以反映出这些更改，也可能不会更改。

对基础数据的更改可能会通过游标*看到*，也可能*看不到*。可见的更改是反映在游标的结果集中的更改。如果对基础数据的更改不反映在通过游标所看到的结果集中，那么这种更改就是不可见的。

目录存储游标敏感性

SAP Sybase IQ 游标按其对基础数据更改的敏感性来进行分类。也就是说，由可见的更改来定义游标的敏感性。

- **不敏感游标** – 当游标打开之后，结果集是固定的。对基础数据的更改都不可见。
- **敏感游标** – 结果集可以在游标打开之后进行更改。对基础数据的所有更改都是可见的。
- **敏感性未定型游标** – 更改可能会反映在通过游标看到的结果集的成员资格、顺序或值中，或者也可能根本没有反映。
- **对值敏感的游标** – 对基础数据的顺序或值的更改是可见的。当游标打开之后结果集的成员资格是固定的。

对游标的不同要求给执行设置了不同的约束，因此，会对性能造成影响。

游标敏感性示例：删除的行

此示例使用一个简单查询来阐释这样的情形：删除结果集中的某一行后，不同的游标会如何响应。

请看下面的事件序列：

1. 某个应用程序在以下针对示例数据库的查询中打开一个游标。

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

EmployeeID	Surname
102	Whitney
105	Cobb
160	Breault
...	...

2. 应用程序通过游标读取第一行 (102)。
3. 应用程序通过游标读取下一行 (105)。
4. 另外一个事务删除了 102 号雇员 (Whitney) 并提交了更改。

在此情况下，游标操作的结果取决于游标敏感性：

- **不敏感游标** – 在通过游标所看到的结果的成员资格中或值中，不反映这一 DELETE：

操作	结果
读取前一行	返回该行的原始副本 (102)。
读取第一行（绝对读取）	返回该行的原始副本 (102)。
读取第二行（绝对读取）	返回未更改的行 (105)。

- **敏感游标** – 结果集的成员资格已经更改，因此，行 105 现在是结果集中的第一行：

操作	结果
读取前一行	返回 `Row Not Found`。没有前一行。
读取第一行（绝对读取）	返回行 105。
读取第二行（绝对读取）	返回行 160。

- **对值敏感的游标** – 结果集的成员资格是固定的，因此行 105 仍是结果集的第二行。DELETE 反映在游标的值中，并在结果集中创建了一个有效洞。

操作	结果
读取前一行	返回 `No current row of cursor`。在游标中过去曾经是第一行的地方有一个洞。
读取第一行（绝对读取）	返回 `No current row of cursor`。在游标中过去曾经是第一行的地方有一个洞。
读取第二行（绝对读取）	返回行 105。

- **敏感性未定型游标** – 对于更改，结果集的成员资格和值是不确定的。读取前一行、第一行或第二行的响应取决于查询的特定优化方法：该方法是否涉及工作表的构造，另外，所读取的行是否已从客户端预读。

  敏感性未定型游标的优点是，对于许多应用程序，敏感性是不重要的。特别是，如果您要使用只进、只读游标，将不会看到基础更改。此外，如果您在高隔离级别运行，那么将不允许基础更改。

游标敏感性示例：更新的行

此示例使用一个简单查询阐释了这样的情形：更新了结果集中的某一行以致结果集的顺序改变时，不同的游标类型是如何响应的。

请看下面的事件序列：

1. 某个应用程序在以下针对示例数据库的查询中打开一个游标。

```
SELECT EmployeeID, Surname
FROM Employees;
```

EmployeeID	Surname
102	Whitney
105	Cobb
160	Breault
...	...

2. 应用程序通过游标读取第一行 (102)。
3. 应用程序通过游标读取下一行 (105)。
4. 另外一个事务将 102 号雇员 (Whitney) 的员工 ID 更新为 165 并提交更改。

在此情况下，游标操作的结果取决于游标敏感性：

- **不敏感游标** – 在通过游标看到的结果的成员资格或值中，没有反映出这一 UPDATE：

操作	结果
读取前一行	返回该行的原始副本 (102)。
读取第一行（绝对读取）	返回该行的原始副本 (102)。
读取第二行（绝对读取）	返回未更改的行 (105)。

- **敏感游标** – 结果集的成员资格已经更改，因此，行 105 现在是结果集中的第一行：

操作	结果
读取前一行	返回 SQLCODE 100。结果集的成员资格已经更改，因此，第 105 行现在是结果集中的第一行。游标已经移到第一行之前的位置。
读取第一行（绝对读取）	返回行 105。
读取第二行（绝对读取）	返回行 160。

此外，如果该行自上次读取后已更改，则在敏感游标上读取时会返回 SQLE_ROW_UPDATED_WARNING 警告。警告只发出一次。后面再读取同一行时，不会产生警告。

同样，自上次读取某行后，如果通过游标在该行上进行定位更新或删除，就会返回 SQLE_ROW_UPDATED_SINCE_READ 错误。应用程序必须再次读取该行才能使敏感游标上的更新或删除生效。

对任何列的更新都会导致警告/错误，即使该列未被游标引用也是如此。例如，返回 Surname 的查询上的游标将报告更新，即使只有 Salary 列进行了修改也是这样。

- **对值敏感的游标** – 结果集的成员资格是固定的，因此行 105 仍是结果集的第二行。UPDATE 反映在游标的值中，并在结果集中创建了一个有效“洞”。

操作	结果
读取前一行	返回 SQLCODE 100。结果集的成员资格已经更改，因此，第 105 行现在是结果集中的第一行：游标定位在该洞上：它在行 105 前面。
读取第一行（绝对读取）	返回 SQLCODE -197。结果集的成员资格已经更改，因此，第 105 行现在是结果集中的第一行：游标定位在该洞上：它在行 105 前面。
读取第二行（绝对读取）	返回行 105。

- **敏感性未定型游标** – 对于更改，结果集的成员资格和值是不确定的。读取前一行、第一行或第二行的响应取决于查询的特定优化方法：该方法是否涉及工作表的构造，另外，所读取的行是否已从客户端预读。

注意： 更新警告和错误情况在批量操作模式（-b 数据库服务器选项）下不会发生。

目录存储不敏感游标

这些游标具有不敏感的成员资格、顺序和值。游标打开之后进行的任何更改都是不可见的。

不敏感游标只用于只读游标类型。

标准

不敏感游标对应于不敏感游标的 ISO/ANSI 标准定义，并对应于 ODBC 静态游标。

编程接口

接口	游标类型	注释
ODBC、ADO/OLE DB	静态	如果请求可更新的静态游标，则改为使用对值敏感的游标。
嵌入式 SQL	INSENSITIVE	
JDBC	INSENSITIVE	仅 SQL Anywhere JDBC 驱动程序支持不敏感语义。
Open Client	不支持	

说明

不敏感游标会始终按任何可能存在的 ORDER BY 子句指定的顺序返回与查询的选择标准匹配的行。

当游标打开之后，不敏感游标的结果集将完全作为工作表实例化。这样会带来以下后果：

- 如果结果集非常大，那么管理结果集需要的磁盘空间和内存可能也非常大。
- 在整个结果集被汇编为工作表之前，没有行返回到应用程序。对于复杂的查询，这可能会导致经过一段延迟后第一行才返回到应用程序。
- 后面的行可以直接从工作表读取，因此可以快速地返回。客户端库可以一次预取多行，从而进一步改善性能。
- 不敏感游标不会受 ROLLBACK 或 ROLLBACK TO SAVEPOINT 的影响。

目录存储敏感游标

敏感游标可以用于只读或可更新的游标类型。

这些游标具有敏感的成员资格、顺序和值。

标准
敏感游标对应于敏感游标的 ISO/ANSI 标准定义，并对应于 ODBC 动态游标。

编程接口

接口	游标类型	注释
ODBC、ADO/ OLE DB	动态	
嵌入式 SQL	SENSITIVE	当不需要工作表并且 prefetch 选项设置为 Off 时，也响应 DYNAMIC SCROLL 游标请求而予以提供。
JDBC	SENSITIVE	SQL Anywhere JDBC 驱动程序全面支持敏感游标。

说明
敏感游标禁用预取。通过游标可看到所有更改，包括通过游标以及从其它事务做出的更改。如果隔离级别较高，则可能会因锁定而隐藏一些在其它事务中做出的更改。

对游标成员资格、顺序以及所有列值的更改都是可见的。例如，如果敏感游标包含连接，而且，修改了某个基础表的某个值，那么，由该基行组成的所有结果行都会显示新值。结果集成员资格和顺序可能会在每一次读取时更改。

敏感游标会始终返回与查询的选择标准匹配的行，返回顺序为任何 ORDER BY 子句指定的顺序。更新可能会影响结果集的成员资格、顺序和值。

敏感游标的要求会对敏感游标的实现施加限制：

- 行不能被预取，因为对预取的行进行的更改通过游标将不可见。这可能会影响性能。
- 敏感游标必须在没有构建任何工作表的情况下实现，这是因为如果行存储为工作表，则无法通过游标看到对这些行的更改。
- 由于存在着不能有工作表这样的限制，所以优化器选择连接方法时也会受到限制，因而性能也可能会受到影响。
- 对于某些查询，优化器不能构建不包括工作表并会使游标敏感的计划。

工作表通常用于对中间结果进行排序和分组。如果行可以通过索引进行访问，那么排序就不需要工作表。虽然无法准确说明都有哪些查询会使用工作表，但以下查询肯定要使用工作表：

- UNION 查询，虽然 UNION ALL 查询不一定使用工作表。
- 带有 ORDER BY 子句的语句，如果在 ORDER BY 列上没有索引。
- 任何使用散列连接优化的查询。
- 许多涉及 DISTINCT 或 GROUP BY 子句的查询。

在这些情况下，SAP Sybase IQ 或者会向应用程序返回错误，或者会将游标类型更改为敏感性未定型游标并返回警告。

目录存储敏感性未定型游标

这些游标在其成员资格、顺序或值方面没有明确定义的敏感性。在敏感性方面允许的这一灵活性可使敏感性未定型游标得以优化，进而改善性能。

敏感性未定型游标只用于只读游标类型。

标准

敏感性未定型游标对应于敏感性未定型游标的 ISO/ANSI 标准定义，并对应于带有非特定敏感性的 ODBC 游标。

编程接口

接口	游标类型
ODBC、ADO/OLE DB	未指定的敏感性
嵌入式 SQL	DYNAMIC SCROLL

说明

SAP Sybase IQ 可使用一些方法来优化查询并向应用程序返回行，请求敏感性未定型游标并不会给这些方法带来任何限制。因此，敏感性未定型游标提供的性能最佳。特别是，对于将中间结果实例化为工作表的任何措施，优化器均可自由使用，而且，客户端可以预取行。

SAP Sybase IQ 不保证用户能够看到对数据库基础行的更改。某些更改可能是可见的，而另外一些是不可见的。成员资格和顺序可能会在每一次读取时都更改。特别是，对基行的更新可能会导致只有部分更新的列反映在游标的结果中。

敏感性未定型游标不能保证返回与查询的选择和顺序匹配的行。行成员资格在游标打开时是固定的，但后续对基础值的更改将反映在结果中。

敏感性未定型游标会始终返回这样的行：这些行在建立游标成员资格时匹配客户的 WHERE 和 ORDER BY 子句。如果在游标打开之后列值被更改，那么可能返回不再匹配 WHERE 和 ORDER BY 子句的行。

目录存储对值敏感的游标

对于对值敏感的游标，成员资格是不敏感的，结果集的顺序和值是敏感的。

对值敏感的游标可以用于只读或可更新的游标类型。

标准
对值敏感的游标不符合 ISO/ANSI 标准定义。它们对应于 ODBC 键集决定的游标。

编程接口

接口	游标类型	注释
ODBC、ADO/OLE DB	由键集决定	
嵌入式 SQL	SCROLL	
JDBC	INSENSITIVE 和 CONCUR_UPDATABLE	使用 SQL Anywhere JDBC 驱动程序时，如果请求可更新的 INSENSITIVE 游标，则会提供对值敏感的游标。
Open Client 和 jConnect	不支持	

说明
如果应用程序读取的行是由已经更改的数据库基础行构成的，那么，就必须给应用程序提供更新后的值，而且必须向应用程序发出 SQL_ROW_UPDATED 状态。如果应用程序试图读取的行是由被删除的数据库基础行构成的，那么，就必须向应用程序发出 SQL_ROW_DELETED 状态。

对主键值的更改会从结果集中删除行（作为删除对待，后面跟插入）。当结果集中的某一行被删除（从游标中或者游标之外）并插入带有同一主键值的一个新行时，会发生特殊情况。这将会导致在旧行出现的位置由新行替换旧行。

结果集中的行并不一定会与查询的选择或顺序指定匹配。由于行成员资格在打开时是固定的，因此，即使后续的更改使行不匹配 WHERE 子句或 ORDER BY，这样的更改也并不会更改行的成员资格，同样也不会更改行的位置。

所有的值对通过游标进行的更改都是敏感的。成员资格对通过游标进行的更改的敏感性受 ODBC 选项 SQL_STATIC_SENSITIVITY 的控制。如果该选项已打开，那么通过游标的插入会将行添加到游标。否则，它们就不是结果集的组成部分。通过游标的删除将会从结果集删除行，从而可防止洞返回 SQL_ROW_DELETED 状态。

对值敏感的游标可使用*键集表*。当游标打开之后，SAP Sybase IQ 将用组成结果集的每一行的标识信息填充工作表。当在结果集中滚动时，将使用键集表标识结果集的成员资格，但在必要时从基础表获取值。

对值敏感的游标的固定成员资格属性可使您的应用程序记住游标内的行位置，并确保这些位置不会改变。

- 如果自从打开游标之后某一行被更新或者可能已经更新，那么，SAP Sybase IQ 将在读取该行时返回 SQLE_ROW_UPDATED_WARNING。警告只生成一次：再次读取同一行时不再生成警告。

  只要更新行中的列，即使游标不引用所更新的列，也会产生警告。例如，即使只有 Birthdate 列被修改，Surname 和 GivenName 上的游标也将报告更新。当行锁定被禁用时，这些更新警告和错误情况在批量操作模式（-b 数据库服务器选项）下不会发生。
- 在自上次读取后被修改的行上执行定位更新或删除的尝试将会返回 SQLE_ROW_UPDATED_SINCE_READ 错误并会取消该语句。应用程序必须再次 FETCH 该行，然后才能允许 UPDATE 或 DELETE。

  只要更新行中的列，即使游标不引用所更新的列，也会产生错误。在批量操作模式下，不会发生错误。
- 打开游标之后，如果通过游标或从另一事务中删除了某行，游标中就会出现一个*洞*。由于游标的成员资格是固定的，因此，会保留行位置，但 DELETE 操作会反映在行的更改的值中。如果您读取此洞上的行，您会收到 -197 SQLCODE 错误，指出没有当前行，并且游标仍定位在该洞上。您可以通过使用敏感游标来避免洞，因为敏感游标的成员资格会随值一起改变。

对于对值敏感的游标，行不能被预取。此要求可能会影响性能。

插入多行
通过对值敏感的游标插入多行时，新插入的行出现在该结果集的末尾处。

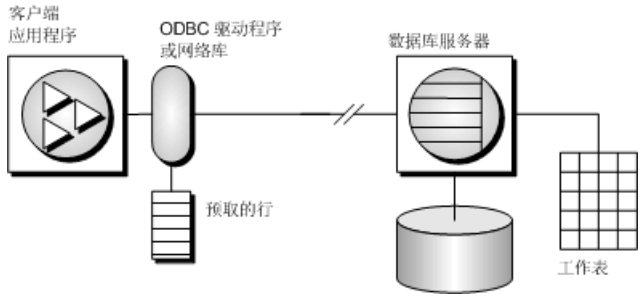
目录存储游标敏感性和性能

性能与游标的其它属性往往无法兼顾。特别是，如果使游标成为可更新游标，就会限制游标查询处理和传递，进而影响性能。此外，对游标敏感性提出要求也会约束游标性能。

要理解游标的可更新性和敏感性如何影响性能，就需要理解能通过游标看到的结果是如何从数据库传输到客户端应用程序的。

特别是，结果可能会由于性能原因存储在两个中间位置：

- **工作表** – 无论是中间还是最终结果都可以作为工作表存储。对值敏感的游标会使用由主键值构成的工作表。查询特性也可能导致优化程序在其选择的执行计划中使用工作表。
- **预取** – 通信的客户端可能会将许多行检索到客户端的缓冲区中，以避免为每一行单独将请求发送到数据库服务器。



敏感性和可更新性限制了中间位置的使用。

预取

预取和多行读取是不同的。预取可以在没有来自客户端应用程序的显式指令的情况下执行。预取会将行从服务器检索到客户端上的缓冲区中，但客户端应用程序要先读取相应的行，然后才能使用这些行。

缺省情况下，只要应用程序读取一行，SAP Sybase IQ 客户端库就会预取多行。SAP Sybase IQ 客户端库会将其余的行存储在缓冲区中。

预取会通过减少客户端/服务器端通信的往返次数而提高性能，并且不用为每一行或行块向服务器发送单独的请求，就可以让许多行可供使用，因而可提高吞吐量。

控制应用程序的预取

- prefetch 选项可以控制是否发生预取。可将单个连接的 prefetch 选项设为 Always、Conditional 或 Off。缺省情况下，它被设置为 Conditional。
- 在嵌入式 SQL 中，当您使用 BLOCK 子句打开单个 FETCH 操作上的游标时，您可以针对每个游标控制预取。

  应用程序可通过指定 BLOCK 子句来指定从服务器读取一次最多可以包含多少行。例如，如果一次读取并显示 5 行，则可以使用 BLOCK 5；如果指定 BLOCK 0，一次就会读取 1 个记录，并且会使 FETCH RELATIVE 0 总是从服务器重复读取行。

  虽然您还可以通过在应用程序上设置连接参数来将预取关闭，但是指定 BLOCK 0 比将 prefetch 选项设置为 Off 效率更高。
- 缺省情况下，预取对于值敏感型游标是禁用的。
- 在 Open Client 中，您可以在声明游标之后（但要在打开它之前）使用 ct_cursor 并使用 CS_CURSOR_ROWS 来控制预取行为。

当性能有提高余地时，预读取会动态地增加预读取的行数。这包括符合以下条件的游标：

- 使用一种支持的游标类型：
  - **ODBC 和 OLE DB** – FORWARD-ONLY 和 READ-ONLY（缺省）游标

  - **嵌入式 SQL** – DYNAMIC SCROLL（缺省）、NO SCROLL 和 INSENSITIVE 游标
  - **ADO.NET** – 所有游标
- 只执行 FETCH NEXT 操作（无绝对读取、相对读取或向后读取）。
- 应用程序不会在两次读取之间更改主机变量类型，也不使用 GET DATA 语句按块获取列数据（支持使用一个 GET DATA 语句获取值）。

更新丢失

使用可更新的游标时，防止更新丢失很重要。更新丢失是这样一种情况：两个或多个事务更新同一行，但这些事务彼此之间都不知道其它事务进行的修改，因此第二个更改会覆盖第一个修改。下面示例说明了此问题：

1. 某个应用程序在以下针对示例数据库的查询中打开一个游标。

```
SELECT ID, Quantity
FROM Products;
```

ID	Quantity
300	28
301	54
302	75
...	...

2. 应用程序通过游标读取 ID = 300 的行。
3. 另外一个事务使用下面的语句更新该行：

```
UPDATE Products
SET Quantity = Quantity - 10
WHERE ID = 300;
```

4. 然后，应用程序通过游标将该行值更新为 `(Quantity - 5)` 的值。
5. 该行的正确的最终值是 13。如果游标预读了该行，那么该行的新值是 23。另外一个事务的更新会丢失。

在数据库应用程序中，如果预先不对行值进行验证就进行更改，那么在任何一个隔离级别都有可能丢失更新。在较高隔离级别（2 和 3），可以使用锁（读取、意图和写入锁）来确保其它事务不能对应用程序已读取的行进行更改。但在隔离级别 0 和 1，更新丢失的可能性会更大：在隔离级别 0，不能获得读取锁来防止随后的数据更改；在隔离级别 1，只能锁定当前行。使用快照隔离时不会发生更新丢失，因为任何更改旧值的尝试都会导致更新冲突。同样，在隔离级别 1 使用预取也有可能导致丢失更新，因为在客户端的预取缓冲区中应用程序定位的结果集行可能与游标中服务器定位的当前行不同。

在隔离级别 1 使用游标时，为防止丢失更新，数据库服务器支持三种不同的并发控制机制，这三个机制可由应用程序指定：

1. 读取游标中的每一行时在该行上获得意图行锁。意图锁防止其它事务在同一行上获得意图锁或写入锁，从而防止并发更新。但是，意图锁不防碍读取行锁，因此意图锁不影响并发只读语句。
2. 使用对值敏感的游标。对值敏感的游标可用于跟踪基础行发生更改或删除的时间，以便应用程序可以采取相应措施。
3. 使用 FETCH FOR UPDATE，此方法可在特定行上获得意图行锁。

指定这些替代方法的方式取决于应用程序所使用的接口。对于适用于 SELECT 语句的前两个替代方法：

- 在 ODBC 中不会发生更新丢失，因为在声明可更新游标时应用程序必须为 SQLSetStmtAttr 函数指定游标并发参数。此参数是 SQL_CONCUR_LOCK、SQL_CONCUR_VALUES、SQL_CONCUR_READ_ONLY 或 SQL_CONCUR_TIMESTAMP 之一。对于 SQL_CONCUR_LOCK，数据库服务器可获取行意图锁。对于 SQL_CONCUR_VALUES 和 SQL_CONCUR_TIMESTAMP，可使用对值敏感的游标。SQL_CONCUR_READ_ONLY 用于只读游标，是缺省值。
- 在 JDBC 中语句的并发设置类似于 ODBC 中的设置。SQL Anywhere JDBC 驱动程序支持 JDBC 并发值 RESULTSET_CONCUR_READ_ONLY 和 RESULTSET_CONCUR_UPDATABLE。第一个值对应于 ODBC 并发设置 SQL_CONCUR_READ_ONLY，指定只读语句。第二个值对应于 ODBC SQL_CONCUR_LOCK 设置，因此使用行意图锁来防止更新丢失。无法在 JDBC 4.0 规范中直接指定对值敏感的游标。
- 在 jConnect 中，在 API 级别支持可更新游标，但底层实现（使用 TDS）不支持通过游标进行更新。jConnect 将单独的 UPDATE 语句发送到数据库服务器来更新特定行。为避免更新丢失，应用程序必须在隔离级别为 2 或更高的情况下运行。应用程序还可以从游标发出单独的 UPDATE 语句，但必须确保 UPDATE 语句通过在其 WHERE 子句中设置相应的条件来验证自读取该行以后该行值未变化。
- 在嵌入式 SQL 中，可通过在 SELECT 语句本身或游标声明中包括语法来设置并发说明。在 SELECT 语句中，语法 SELECT...FOR UPDATE BY LOCK 导致数据库服务器在结果集上获取意图行锁。

  或者，SELECT...FOR UPDATE BY [VALUES | TIMESTAMP] 促使数据库服务器将游标类型更改为对值敏感的游标，这样如果自上次读取特定行以后通过游标对该行进行了更改，在使用 FETCH 语句时应用程序会收到警告 (SQLE_ROW_UPDATED_WARNING)，在使用 UPDATE WHERE CURRENT OF 语句时应用程序会收到错误 (SQLE_ROW_UPDATED_SINCE_READ)。如果删除行，应用程序也会收到错误 (SQLE_NO_CURRENT_ROW)。

嵌入式 SQL 和 ODBC 接口也支持 FETCH FOR UPDATE 功能，虽然细节方面因所使用的 API 而不同。

在嵌入式 SQL 中，应用程序使用 FETCH FOR UPDATE，而非 FETCH，促使在该行上获取意图锁。在 ODBC 中，应用程序使用 API 调用 SQLSetPos，并使用操作参数 SQL_POSITION 或 SQL_REFRESH 和锁类型参数 SQL_LOCK_EXCLUSIVE，以在行上获取意图锁。在 SAP Sybase IQ 中，这些锁是长期锁，会一直保持到提交或回退事务。

目录存储游标敏感性和隔离级别

游标敏感性和隔离级别都可解决并发控制问题，但处理方式不同，并且利弊也各不相同。

通过选择事务的隔离级别（通常在连接级别），您可以确定何时在数据库中的行上放置锁以及锁的类型。锁可以防止其它事务访问或修改数据库中的行。通常，维护的锁的数目越多，并发事务之间预期的并发级别越低。

但是，锁并不会阻止同一事务的其它部分进行更新。因此，维护多个可更新游标的单个事务无法依赖锁定来防止诸如更新丢失之类的问题。

快照隔离的目的是避免使用读取锁，其方法是确保每个事务都能查看到一致的数据库视图。显而易见的好处是无需依赖于完全可序列化事务（隔离级别 3）即可查询到一致的数据库视图，并且可以避免随使用隔离级别 3 发生的并发丢失。但是，快照隔离会带来巨大的开销，因为必须维护修改行的副本才能同时满足正在执行的并发快照事务的要求，以及尚未启动的快照事务的要求。由于存在这种副本维护，因此存在大量的更新时可能不适合使用快照隔离。

但是，游标的敏感性可以确定在游标的结果中哪些更改可见（或不可见）。虽然事务的影响完全取决于指定的游标类型，但因为游标敏感性是基于游标指定的，所以游标敏感性既应用于其它事务的影响，也应用于同一事务的更新活动。通过设置游标敏感性，将不直接确定何时在数据库中的行上放置锁。而是由游标敏感性与隔离级别的组合来控制对特定应用程序可能发生的各种并发情况。

请求 SAP Sybase IQ 目录存储游标

当您从客户端应用程序请求游标类型时，SAP Sybase IQ 会提供一个游标。SAP Sybase IQ 游标不是按照编程接口中指定的类型进行定义的，而是按照结果集对基础数据中更改的敏感性来定义的。SAP Sybase IQ 会根据您请求的游标类型提供一个行为与该类型相匹配的游标。

SAP Sybase IQ 按照客户端游标类型的请求来设置游标敏感性。

ADO.NET

通过使用 SACommand.ExecuteReader 可使用只进、只读游标。SADataAdapter 对象不使用游标，而是使用客户端结果集。

ADO/OLE DB 和 ODBC

下表说明了为响应不同的 ODBC 可滚动游标类型而设置的游标敏感性。

ODBC 可滚动游标类型	SAP Sybase IQ 游标
STATIC	不敏感
KEYSET-DRIVEN	对值敏感
DYNAMIC	敏感

ODBC 可滚动游标类型	SAP Sybase IQ 游标
MIXED	对值敏感

通过将游标类型设置为 SQL_CURSOR_KEYSET_DRIVEN，然后使用 SQL_ATTR_KEYSET_SIZE 为由键集决定的游标指定键集中的行数，可获取 MIXED 游标。如果键集大小为 0（缺省值），则游标完全由键集决定。如果键集大小大于 0，则游标是混合的（在键集内由键集决定，在键集外动态变化）。键集大小的缺省值为 0。键集大小大于 0 而小于行集大小 (SQL_ATTR_ROW_ARRAY_SIZE) 是错误的。

例外情况
如果将 STATIC 游标请求为可更新，则提供的将是对值敏感的游标，而且会发出一个警告。

如果请求了 DYNAMIC 或 MIXED 游标，但若不使用工作表查询就无法执行，那么就会发出警告，并改为提供敏感性未定型游标。

JDBC
JDBC 4.0 规范支持三种游标类型：不敏感、敏感和只进敏感性未定型。SQL Anywhere JDBC 驱动程序符合这些 JDBC 规范，也支持 JDBC ResultSet 对象的三种不同的游标类型。但是，在一些情况下，数据库服务器无法使用给定游标类型所需的语义构造访问计划。在这些情况下，数据库服务器或者返回错误，或者用其它游标类型替代。

使用 jConnect 时，尽管 jConnect 支持的 API 可按照 JDBC 2.0 规范创建不同的游标类型，但在数据库服务器上底层协议 (TDS) 却仅支持只进、只读敏感性未定型游标。因为 TDS 协议将语句的结果集缓存在块中，所以所有 jConnect 游标都是敏感性未定型。应用程序需要滚动支持可滚动性的不敏感或敏感游标类型时，会滚动这些缓存结果的块。如果应用程序向后滚动时越过高速缓存的结果集的开头，就会再次执行语句。在这种情况下，如果在两次语句执行之间更改了数据，就会导致数据不一致。

嵌入式 SQL
要从嵌入式 SQL 应用程序请求游标，您可以在 DECLARE 语句上指定游标类型。下表说明了为响应不同的请求设置的游标敏感性：

游标类型	SAP Sybase IQ 游标
NO SCROLL	敏感性未定型
DYNAMIC SCROLL	敏感性未定型
SCROLL	对值敏感
INSENSITIVE	不敏感
SENSITIVE	敏感

例外情况
如果将 DYNAMIC SCROLL 或 NO SCROLL 游标请求为 UPDATABLE，就会提供敏感的或对值敏感的游标。无法保证提供两个游标中的哪一个。这种不确定性正好符合敏感性未定行为的定义。

如果将 INSENSITIVE 游标请求为 UPDATABLE，就会提供一个对值敏感的游标。

如果请求 DYNAMIC SCROLL 游标，那么在 prefetch 数据库选项被设置为 Off 而且查询执行计划未涉及工作表的情况下，可能会提供敏感的游标。同样地，这种不确定性也符合敏感性未定行为的定义。

Open Client
与 jConnect 一样，Open Client 的底层协议 (TDS) 也仅支持只进、只读、敏感性未定型游标。

结果集描述符

有些应用程序会构建无法完全在应用程序中指定的 SQL 语句。有时在应用程序准确知道要检索什么信息之前（比如当某一报告应用程序让用户选择要显示的列时），语句取决于用户的响应。

在这种情况下，应用程序需要用一种方法来检索关于*结果集*的性质及内容方面的信息。关于结果集性质的信息（被称为*描述符*）用于标识数据结构，包括希望返回的列的数量和类型。一旦应用程序确定了结果集的性质，检索内容的过程就简单了。

此*结果集元数据*（关于数据的性质和内容的信息）通过描述符来操纵。获取和管理结果集元数据的过程称为*描述*。

由于游标通常会产生结果集，所以描述符和游标会紧密链接，只不过有些接口隐藏了描述符的使用，用户看不到。通常，需要描述符的语句是返回结果集的 SELECT 语句或存储过程。

与基于游标的操作一起使用描述符的序列如下所示：

1. 分配描述符。此操作可以隐式进行，不过有些接口也允许显式分配。
2. 准备语句。
3. 描述语句。如果语句是存储过程调用或批处理，且结果集不是由过程定义中的结果子句定义的，那么描述应在打开游标之后发生。
4. 为语句声明和打开游标（嵌入式 SQL）或执行该语句。
5. 如果有必要，获取描述符并修改分配区域。此步骤通常隐式进行。
6. 读取和处理语句结果。
7. 释放描述符。
8. 关闭游标。
9. 删除语句。有些接口可以自动完成此步骤。

实现注意事项

- 在嵌入式 SQL 中，SQLDA（SQL 描述符区域）结构将保存描述符信息。
- 在 ODBC 中，使用 SQLAllocHandle 分配的描述符句柄提供了对描述符的字段的访问。您可以使用 SQLSetDescRec、SQLSetDescField、SQLGetDescRec 和 SQLGetDescField 对这些字段进行操作。

  或者，您还可以使用 SQLDescribeCol 和 SQLColAttributes 获取列信息。
- 在 Open Client 中，您可以使用 ct_dynamic 来准备语句，使用 ct_describe 来描述语句的结果集。不过，您还可以使用 ct_command 来发送 SQL 语句而不必事先准备它，并使用 ct_results 逐个地处理返回的行。这是在 Open Client 应用程序开发中比较常见的操作方式。
- 在 JDBC 中，java.sql.ResultSetMetaData 类提供了有关结果集的信息。
- 您还可以使用向数据库服务器发送数据的描述符（例如，使用 INSERT 语句）；然而，这与用于结果集的描述符是不同种类的描述符。

应用程序中的事务

事务是原子 SQL 语句集。要么执行事务中的所有语句，要么一个也不执行。本节介绍应用程序中的事务的一些方面。

自动提交和手动提交模式

数据库编程接口可以在*手动提交*模式或*自动提交*模式下运行。

- **手动提交模式** – 只有在您的应用程序执行显式提交操作时，或者，在数据库服务器执行自动提交时（例如，执行 ALTER TABLE 语句或其它数据定义语句时），才提交操作。手动提交模式有时也称为*链接模式*。

  要在应用程序中使用事务（包括嵌套事务和保存点），您必须在手动提交模式下操作。
- **自动提交模式** – 每一语句都视为单独的事务。自动提交模式相当于在每条 SQL 语句的结尾附加一条 COMMIT 语句。自动提交模式有时也称为*非链接模式*。

自动提交模式会影响应用程序的性能和行为。如果应用程序需要事务完整性，则不要使用自动提交。

如何控制自动提交行为

控制应用程序的提交行为的方式取决于您使用的编程接口。自动提交的实现可以是客户端的或服务器端的，具体情况视接口而定。

控制自动提交模式 (ADO.NET)

缺省情况下，ADO.NET 提供程序在自动提交模式下工作。若要使用显式事务，请使用 SAConnection.BeginTransaction 方法。

控制自动提交模式 (OLE DB)
缺省情况下，OLE DB 提供程序在自动提交模式下工作。若要使用显式事务，请使用 ITransactionLocal::StartTransaction、ITransaction::Commit 和 ITransaction::Abort 方法。

控制自动提交模式 (ODBC)
缺省情况下，ODBC 在自动提交模式下工作。关闭自动提交的方式取决于您是在直接使用 ODBC 还是在使用应用程序开发工具。如果要直接对 ODBC 接口编程，则请设置 SQL_ATTR_AUTOCOMMIT 连接属性。

控制自动提交模式 (JDBC)
缺省情况下，JDBC 在自动提交模式下工作。若要关闭自动提交，请使用连接对象的 setAutoCommit 方法：

```
conn.setAutoCommit( false );
```

控制自动提交模式（嵌入式 SQL）
缺省情况下，嵌入式 SQL 应用程序在手动提交模式下工作。若要启用自动提交，请使用类似如下的语句将 chained 数据库选项（服务器端选项）设置为 Off：

```
SET OPTION chained='Off';
```

控制自动提交模式 (Open Client)
缺省情况下，通过 Open Client 进行的连接在自动提交模式下工作。您可以更改此行为，方法是使用类似如下的语句在应用程序中将 chained 数据库选项（服务器端选项）设置为 On：

```
SET OPTION chained='On';
```

控制自动提交模式 (PHP)
缺省情况下，PHP 在自动提交模式下工作。要关闭自动提交，请使用 sasql_set_option 函数：

```
$result = sasql_set_option( $conn, "auto_commit", "Off" );
```

控制自动提交模式（在服务器上）
缺省情况下，数据库服务器在手动提交模式下工作。若要启用自动提交，请使用类似如下的语句将 chained 数据库选项（服务器端选项）设置为 Off：

```
SET OPTION chained='Off';
```

如果使用的是在客户端上控制提交的接口，则设置 chained 数据库选项（服务器端选项）会影响应用程序的性能和/或行为。建议不要设置服务器的链接模式。

自动提交实现细节
根据您使用的接口和提供程序以及控制自动提交行为的方式，自动提交模式的行为会略有不同。

实现自动提交模式可以采用以下两种方式之一：

- **客户端自动提交** – 当应用程序使用自动提交时，客户端库在每一个 SQL 语句执行之后发送 COMMIT 语句。

  ADO.NET、ADO/OLE DB、ODBC、PHP 和 SQL Anywhere JDBC 驱动程序应用程序控制来自客户端的提交行为。
- **服务器端自动提交** – 应用程序关闭链接模式时，数据库服务器提交每个 SQL 语句的结果。对于 Sybase jConnect JDBC 驱动程序，此行为受 chained 数据库选项控制。

  嵌入式 SQL、jConnect 驱动程序和 Open Client 应用程序操控服务器端的提交行为（例如，这些应用程序设置 chained 选项）。

对于复合语句，如存储过程或触发器，客户端和服务器端的自动提交有所不同。从客户端看，存储过程是单一语句，因此自动提交将在整个过程执行之后发送单一提交语句。从数据库服务器的角度来看，存储过程可以由许多 SQL 语句构成，因此服务器端自动提交将提交该过程内的每个 SQL 语句的结果。

注意： 不要将客户端和服务器端实现混合使用。在 SAP Sybase IQ ADO.NET、OLE DB、ODBC、PHP 或 JDBC 应用程序中，不应将设置 chained 选项与设置 autocommit 选项这两个操作组合在一起。

隔离级别设置

可以使用 isolation_level 数据库选项设置当前连接的隔离级别。

某些接口（如 ODBC）允许您在连接时设置连接的隔离级别。以后，您可以使用 isolation_level 数据库选项将此级别重置。

通过在 INSERT、UPDATE、DELETE、SELECT 和 UNION 语句中包括 OPTION 子句，可以替代 isolation_level 数据库选项在各个语句中的任何临时或公共设置。

游标和事务

通常，执行 COMMIT 时游标会关闭。但此行为也有两个例外。

- close_on_endtrans 数据库选项被设置为 Off。
- 游标打开为 WITH HOLD，这是 Open Client 和 JDBC 的缺省值。

如果这两种情况中任何一种为真，那么游标在 COMMIT 时依然打开。

ROLLBACK 和游标

如果事务回退，游标就会关闭，但那些使用 WITH HOLD 打开的游标除外。但是回退后游标的内容并不可靠。

ISO SQL3 标准草案声明：在回退时，所有游标（甚至是那些使用 WITH HOLD 打开的游标）都应关闭。您可以通过将 ansi_close_cursors_on_rollback 选项设置为 On 来获得此行为。

保存点
如果事务回退到保存点，并且如果 ansi_close_cursors_on_rollback 选项为 On，那么所有在 SAVEPOINT 后还处于打开状态的游标（甚至是那些使用 WITH HOLD 打开的游标）将关闭。

游标和隔离级别
可以使用 SET OPTION 语句来更改 isolation_level 选项，以在事务期间更改连接的隔离级别。但是，此更改不影响打开的游标。

当 WITH HOLD 子句与快照、语句快照以及只读语句快照隔离级别一起使用时，在快照启动时所提交的所有行的快照是可见的。从在其内打开游标的事务的启动开始，由当前连接所完成的所有修改也是可见的。

.NET 应用程序编程

本节介绍如何结合 .NET 使用 SAP Sybase IQ，而且包含 SAP Sybase IQ .NET 数据提供程序的 API。

SAP Sybase IQ .NET 数据提供程序

本节介绍 .NET 支持，其中包括以下方面的提示：在 Visual Studio 项目中使用 SAP Sybase IQ .NET 数据提供程序；连接到数据库；从数据库表中读取、插入、更新和删除行；调用存储过程；使用事务；以及基本错误处理。

SAP Sybase IQ .NET 支持

ADO.NET 是 Microsoft 的 ODBC、OLE DB 和 ADO 系列中最新的数据访问 API。它是 Microsoft .NET Framework 首选的数据访问组件，可用于访问关系数据库系统。

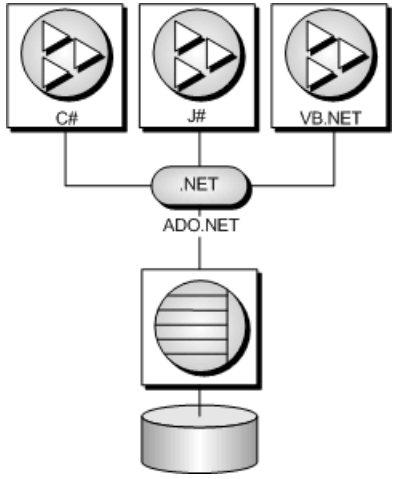
SAP Sybase IQ .NET 数据提供程序实现了 iAnywhere.Data.SQLAnywhere 命名空间，允许您使用 .NET 支持的任何语言（例如 C# 和 Visual Basic .NET）编写程序，并从 SAP Sybase IQ 数据库访问数据。

有关 .NET 数据访问的一般信息，请参见 Microsoft ".NET Data Access Architecture Guide"，网址为 http://msdn.microsoft.com/en-us/library/Ee817654%28pandp.10%29.aspx。

ADO.NET 应用程序

您可以使用面向对象的语言开发 Internet 和 Intranet 应用程序，然后使用 ADO.NET 数据提供程序将这些应用程序连接到 SAP Sybase IQ。

将此提供程序与内置的 XML 和 Web 服务功能、用于 MobiLink™ 同步的 .NET 脚本编写功能以及用于开发手持式数据库应用程序的 UltraLite.NET™ 组件组合起来，这样 SAP Sybase IQ 即可与 .NET Framework 进行集成。



SAP Sybase IQ .NET 数据提供程序功能

SAP Sybase IQ 通过三个不同的命名空间支持 Microsoft .NET Framework 版本 2.0、3.0、3.5、4.0 和 4.5。

- **iAnywhere.Data.SQLAnywhere** – ADO.NET 对象模型是通用数据访问模型。ADO.NET 组件设计用于代管数据操作中的数据访问。有两个可完成此任务的 ADO.NET 中央组件：DataSet 和 .NET Framework 数据提供程序，其中 .NET Framework 数据提供程序是一组组件，其中包括 Connection、Command、DataReader 和 DataAdapter 对象。SAP Sybase IQ 包括一个 .NET Entity Framework 数据提供程序，该数据提供程序可与 SAP Sybase IQ 数据库服务器直接通信，而不会增加 OLE DB 或 ODBC 开销。在 .NET 命名空间中，SAP Sybase IQ .NET 数据提供程序表示为 iAnywhere.Data.SQLAnywhere。

  Microsoft .NET Compact Framework 是适用于 Microsoft .NET 的智能设备开发框架。SAP Sybase IQ.NET Compact Framework 数据提供程序支持运行 Windows Mobile 的设备。支持 Compact Framework 2.0 和 3.5 版。

  SAP Sybase IQ .NET 数据提供程序命名空间在本文档中介绍。

  要进一步了解如何使用 ADO.NET 对象模型，特别是如何通过 Language Integrated Query (LINQ) to Entities 方法访问 SAP Sybase IQ 数据库中存储的数据，请参见 www.sybase.com/detail?id=1060541 上的白皮书 SQL Anywhere and the ADO.NET Entity Framework。
- **System.Data.Oledb** – 此命名空间支持 OLE DB 数据源。此命名空间是 Microsoft .NET Framework 的固有部分。System.Data.Oledb 可与 SQL Anywhere OLE DB 提供程序 SAOLEDB 一起使用，以访问 SAP Sybase IQ 数据库。
- **System.Data.Odbc** – 此命名空间支持 ODBC 数据源。此命名空间是 Microsoft .NET Framework 的固有部分。System.Data.Odbc 可与 SQL Anywhere ODBC 驱动程序一起使用，以访问 SAP Sybase IQ 数据库。

使用 SAP Sybase IQ .NET 数据提供程序具有几大优点：

- 在 .NET 环境中，SAP Sybase IQ .NET 数据提供程序提供对 SAP Sybase IQ 数据库的本地访问。与其它受支持的提供程序不同，它直接与 SAP Sybase IQ 服务器进行通信而不需要使用桥接技术。
- 因此，SAP Sybase IQ .NET 数据提供程序比 OLE DB 和 ODBC 数据提供程序速度更快。建议使用此数据提供程序访问 SAP Sybase IQ 数据库。

.NET 示例项目

SAP Sybase IQ .NET 数据提供程序中随附了几个示例项目：

- **LinqSample** – 适用于 Windows 的 .NET Framework 示例项目，演示如何使用 SAP Sybase IQ .NET 数据提供程序和 C# 进行语言集成查询、设置和转换操作。
- **SimpleWin32** – 适用于 Windows 的 .NET Framework 示例项目，该项目演示了一个简单的列表框，当单击“**Connect**”时，会用 Employees 表中的姓名填充该列表框。
- **SimpleXML** – 适用于 Windows 的 .NET Framework 示例项目，演示如何通过 ADO.NET 从 SAP Sybase IQ 获得 XML 数据。提供用于 C#、Visual Basic 和 Visual C++ 的示例。
- **SimpleViewer** – 适用于 Windows 的 .NET Framework 示例项目。
- **TableViewer** – 适用于 Windows 的 .NET Framework 示例项目，允许用户输入和执行 SQL 语句。

在 Visual Studio 项目中使用 .NET 数据提供程序

使用 SAP Sybase IQ .NET 数据提供程序和 Visual Studio 开发 .NET 应用程序，方法是在源代码中添加 SAP Sybase IQ .NET 数据提供程序的引用，以及一行引用该 SAP Sybase IQ .NET 数据提供程序类的语句。

前提条件

执行此任务没有前提条件。

过程

1. 启动 Visual Studio 并打开项目。
2. 在“**Solution Explorer**”窗口中，右击“**References**”，然后单击“**Add Reference**”。

   引用将指示要包含哪个提供程序并定位 SAP Sybase IQ .NET 数据提供程序的代码。
3. 单击“**.NET**”选项卡，然后在列表中滚动以找到以下内容之一：

   iAnywhere.Data.SQLAnywhere (.NET 2)

iAnywhere.Data.SQLAnywhere (.NET 3.5)
iAnywhere.Data.SQLAnywhere (.NET 4)

4. 单击所需的提供程序，然后单击“**OK**”。

   提供程序将添加到项目的“**Solution Explorer**”窗口的“**References**”文件夹中。

5. 向源代码指定一条指令，以帮助使用 SAP Sybase IQ .NET 数据提供程序命名空间以及在其中定义的类型。

   向项目添加下面的指令行：

   - 如果使用的是 C#，请将下行添加到源代码开始处的 `using` 指令列表中：

     ```
     using iAnywhere.Data.SQLAnywhere;
     ```

   - 如果使用的是 Visual Basic，则在源代码开头处添加下行：

     ```
     Imports iAnywhere.Data.SQLAnywhere
     ```

SAP Sybase IQ .NET 数据提供程序适用于 .NET 应用程序。

.NET 数据库连接示例

要连接到数据库，必须创建 SAConnection 对象。可在创建对象时指定连接字符串，也可在以后通过设置 ConnectionString 属性再建立连接字符串。

设计精良的应用程序应当能够处理在尝试连接数据库时出现的任何错误。

连接打开时，创建与数据库的连接；连接关闭时，释放与数据库的连接。

C# SAConnection 示例

以下 C# 代码将创建一个按钮单击处理程序，用于打开与 SAP Sybase IQ 示例数据库的连接，然后再将其关闭。异常处理程序包含在其中。

```
private void button1_Click(object sender, EventArgs e)
{
    SAConnection conn = new SAConnection("Data Source=Sybase IQ
Demo");
    try
    {
        conn.Open();

        conn.Close();
    }
    catch (SAException ex)
    {
        MessageBox.Show(ex.Errors[0].Source + " : " +
             ex.Errors[0].Message + " (" +
             ex.Errors[0].NativeError.ToString() + ")",
             "Failed to connect");
    }
}
```

Visual Basic SAConnection 示例

以下 Visual Basic 代码将创建一个按钮单击处理程序，用于打开与 SAP Sybase IQ 示例数据库的连接，然后再将其关闭。异常处理程序包含在其中。

```
Private Sub Button1_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New SAConnection("Data Source=Sybase IQ Demo")
    Try
        conn.Open()

        conn.Close()
    Catch ex As SAException
        MessageBox.Show(ex.Errors(0).Source & " : " & _
                ex.Errors(0).Message & " (" & _
                ex.Errors(0).NativeError.ToString() & ")", _
                "Failed to connect")
    End Try
End Sub
```

连接池

SAP Sybase IQ .NET 数据提供程序支持本地 .NET 连接池。连接池使应用程序可以通过将连接句柄保存到池中来重复使用现有连接，而不用重复新建与数据库的连接。连接池在缺省情况下处于启用状态。

连接池的启用和禁用可使用 `Pooling` 选项加以控制。最大池容量使用 `Max Pool Size` 选项在连接字符串中设置。最大池容量或初始池容量使用 `Min Pool Size` 选项在连接字符串中设置。最大池容量的缺省值为 100，最小池容量的缺省值为 0。

```
"Data Source=Sybase IQ Demo;Pooling=true;Max Pool Size=50;Min Pool
Size=5"
```

当应用程序第一次尝试连接数据库时，它将在池中检查是否存在使用您所指定的连接参数的现有连接。如果找到匹配的连接，将使用该连接。否则将使用新的连接。断开连接时，连接将返回池中，以便可以重复使用该连接。

SAP Sybase IQ 数据库服务器也支持连接池。通过 ConnectionPool (CPOOL) 连接参数控制该功能。然而，SAP Sybase IQ .NET 数据提供程序不使用该服务器功能，且禁用了它 (CPOOL=NO)。所有连接池都转而在 .NET 客户端应用程序上完成（客户端连接池）。

连接状态

应用程序与数据库建立连接后，便可检查连接状态，以确保在将请求发送到数据库服务器之前连接仍处于打开状态。如果连接关闭，可向用户发送相应的提示消息和/或尝试重新打开该连接。

SAConnection 类有个可用于检查连接状态的 State 属性。可能的状态值为 ConnectionState.Open 和 ConnectionState.Closed。

以下代码将检查 SAConnection 对象是否已初始化，如果已初始化，则检查连接是否处于打开状态。如果连接没有打开，则会为用户返回一条消息。

```
if ( conn == null || conn.State != ConnectionState.Open )
{
    MessageBox.Show( "Connect to a database first", "Not
connected" );
```

```
    return;
}
```

访问和操作数据

如果使用 SAP Sybase IQ .NET 数据提供程序，则有两种方法访问数据：

- **SACommand 对象** – 建议使用 SACommand 对象在 .NET 中访问和操作数据。

  SACommand 对象允许您执行直接从数据库检索或修改数据的 SQL 语句。使用 SACommand 对象可直接对数据库发出 SQL 语句以及调用存储过程。

  在 SACommand 对象中，SADataReader 用于从查询或存储过程返回只读结果集。SADataReader 每次仅返回一行，但这并不会降低性能，因为 SAP Sybase IQ 客户端的库使用预取缓冲每次预取多行。

  使用 SACommand 对象可以将更改组合成事务，而不是在自动提交模式下操作。使用 SATransaction 对象时，会将行锁定，这样其他用户便无法对其进行修改。
- **SADataAdapter 对象** – SADataAdapter 对象会将整个结果集检索到一个 DataSet 中。DataSet 是用于保存从数据库检索到的数据的断开连接的存储区。之后可以编辑 DataSet 中的数据，编辑完成后，SADataAdapter 对象利用对 DataSet 所做的更改更新数据库。使用 SADataAdapter 时，无法阻止其他用户修改 DataSet 中的行。您需要在应用程序中包含用于解决可能出现的任何冲突的逻辑。

在 SACommand 对象中使用 SADataReader 而不使用 SADataAdapter 对象从数据库读取行时，对性能没有影响。

SACommand：使用 ExecuteReader 和 ExecuteScalar 读取数据

使用 SACommand 对象可针对 SAP Sybase IQ 数据库执行 SQL 语句或调用存储过程。可以使用下列方法中的任一种检索数据库中的数据：

- **ExecuteReader** – 发出返回结果集的 SQL 查询。此方法使用只进、只读游标。可以沿一个方向快速循环遍历结果集中的行。
- **ExecuteScalar** – 发出返回单个值的 SQL 查询。可以是结果集的第一行中的第一列，或返回集合值（如 COUNT 或 AVG）的 SQL 语句。此方法使用只进、只读游标。

使用 SACommand 对象时，可使用 SADataReader 检索基于连接的结果集。但是，只能对一个表中的数据进行更改（插入、更新或删除）。不能更新基于连接的数据集。

使用 SADataReader 时，可使用多种 Get 方法来返回指定数据类型的结果。

C# ExecuteReader 示例

以下 C# 代码将打开与 SAP Sybase IQ 示例数据库的连接，然后使用 ExecuteReader 方法创建 Employees 表中包含雇员姓氏的结果集：

```
SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();
SACommand cmd = new SACommand("SELECT Surname FROM Employees", conn);
SADataReader reader = cmd.ExecuteReader();
listEmployees.BeginUpdate();
```

```
while (reader.Read())
{
    listEmployees.Items.Add(reader.GetString(0));
}
listEmployees.EndUpdate();
reader.Close();
conn.Close();
```

Visual Basic ExecuteReader 示例
以下 Visual Basic 代码将打开与 SAP Sybase IQ 示例数据库的连接，然后使用 ExecuteReader 方法创建 Employees 表中包含雇员姓氏的结果集：

```
Dim conn As New SAConnection("Data Source=Sybase IQ Demo")
Dim cmd As New SACommand("SELECT Surname FROM Employees", conn)
Dim reader As SADataReader
conn.Open()
reader = cmd.ExecuteReader()
ListEmployees.BeginUpdate()
Do While (reader.Read())
    ListEmployees.Items.Add(reader.GetString(0))
Loop
ListEmployees.EndUpdate()
conn.Close()
```

C# ExecuteScalar 示例
以下 C# 代码将打开与 SAP Sybase IQ 示例数据库的连接，然后使用 ExecuteScalar 方法获得 Employees 表中男性雇员的数量：

```
SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();
SACommand cmd = new SACommand(
    "SELECT COUNT(*) FROM Employees WHERE Sex = 'M'", conn );
int count = (int) cmd.ExecuteScalar();
textBox1.Text = count.ToString();
conn.Close();
```

SACommand：使用 GetSchemaTable 读取结果集模式
您可以获得结果集中列的模式信息。

使用 SADataReader 类的 GetSchemaTable 方法将获得当前结果集的相关信息。GetSchemaTable 方法返回标准 .NET DataTable 对象，该对象提供有关结果集中所有列的信息，包括列属性。

C# 模式信息示例
在以下示例中，通过 GetSchemaTable 方法可获得结果集的相关信息，然后将 DataTable 对象绑定到屏幕上的 datagrid。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand cmd = new SACommand("SELECT * FROM Employees", conn);
SADataReader reader = cmd.ExecuteReader();
DataTable schema = reader.GetSchemaTable();
reader.Close();
```

```
conn.Close();
dataGridView1.DataSource = schema;
```

SACommand：使用 ExecuteNonQuery 插入、删除和更新行

若要使用 SACommand 对象执行插入、更新或删除，请使用 ExecuteNonQuery 函数。ExecuteNonQuery 函数发出一个不返回结果集的查询（SQL 语句或存储过程）。

只能对一个表中的数据进行更改（插入、更新或删除）。不能更新基于连接的数据集。必须连接到数据库才能使用 SACommand 对象。

要设置 SQL 语句的隔离级别，必须将 SACommand 对象用作 SATransaction 对象的一部分。不使用 SATransaction 对象修改数据时，提供程序在自动提交模式下运行，您所做的任何更改都将立即得到应用。

C# ExecuteNonQuery DELETE 和 INSERT 示例

以下示例将打开与 SAP Sybase IQ 示例数据库的连接，并使用 ExecuteNonQuery 方法删除所有 ID 大于等于 600 的部门，然后向 Departments 表中添加两个新行。更新后的表将以 datagrid 形式显示。

```
SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();

SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600",
    conn);
deleteCmd.ExecuteNonQuery();

SACommand insertCmd = new SACommand(
    "INSERT INTO Departments(DepartmentID, DepartmentName)
VALUES( ?, ? )",
    conn );
SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
insertCmd.Parameters.Add( parm );
parm = new SAParameter();
parm.SADbType = SADbType.Char;
insertCmd.Parameters.Add( parm );

insertCmd.Parameters[0].Value = 600;
insertCmd.Parameters[1].Value = "Eastern Sales";
int recordsAffected = insertCmd.ExecuteNonQuery();

insertCmd.Parameters[0].Value = 700;
insertCmd.Parameters[1].Value = "Western Sales";
recordsAffected = insertCmd.ExecuteNonQuery();

SACommand selectCmd = new SACommand(
    "SELECT * FROM Departments", conn );
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
```

```
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();
```

C# ExecuteNonQuery UPDATE 示例

以下示例将打开与 SAP Sybase IQ 示例数据库的连接，然后使用 ExecuteNonQuery 方法将 Departments 表中 DepartmentID 为 100 的所有行的 DepartmentName 列更新为 "Engineering"。更新后的表将以 datagrid 形式显示。

```
SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();

SACommand updateCmd = new SACommand(
    "UPDATE Departments SET DepartmentName = 'Engineering' " +
    "WHERE DepartmentID = 100", conn );
int recordsAffected = updateCmd.ExecuteNonQuery();

SACommand selectCmd = new SACommand(
    "SELECT * FROM Departments", conn );
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();
```

SACommand：检索新插入行的主键值

如果正在更新的表包含自动增量主键，请使用 UUID；或者，如果主键来自主键池，则可以使用存储过程获取该数据源生成的主键值。

C# SACommand 主键示例

以下示例显示如何获得为新插入行所生成的主键。在本示例中，将使用 SACommand 对象调用 SQL 存储过程和 SAParameter 对象以对其返回的主键进行检索。出于演示的目的，本示例将创建一个示例表 (adodotnet_primarykey) 和用于插入行和返回主键值的存储过程 (sp_adodotnet_primarykey)。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();

SACommand cmd = conn.CreateCommand();

cmd.CommandText = "DROP TABLE adodotnet_primarykey";
cmd.ExecuteNonQuery();
```

```
cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey ("
+
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE or REPLACE PROCEDURE
sp_adodotnet_primarykey(" +
    "out p_id int, in p_name char(40) )" +
    "BEGIN " +
    "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
    "SELECT @@IDENTITY INTO p_id;" +
    "END";
cmd.ExecuteNonQuery();

cmd.CommandText = "sp_adodotnet_primarykey";
cmd.CommandType = CommandType.StoredProcedure;

SAParameter parmId = new SAParameter();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
cmd.Parameters.Add(parmId);

SAParameter parmName = new SAParameter();
parmName.SADbType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
cmd.Parameters.Add(parmName);

parmName.Value = "R & D --- Command";
cmd.ExecuteNonQuery();
int id1 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id1);

parmName.Value = "Marketing --- Command";
cmd.ExecuteNonQuery();
int id2 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id2);

parmName.Value = "Sales --- Command";
cmd.ExecuteNonQuery();
int id3 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id3);

parmName.Value = "Shipping --- Command";
cmd.ExecuteNonQuery();
int id4 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id4);

cmd.CommandText = "SELECT * FROM adodotnet_primarykey";
cmd.CommandType = CommandType.Text;
SADataReader dr = cmd.ExecuteReader();
conn.Close();
dataGridView1.DataSource = dr;
```

SADataAdapter：概述

SADataAdapter 会将结果集检索到一个 DataTable 中。DataSet 是表 (DataTable) 以及这些表之间的关系和约束的集合。DataSet 内置在 .NET Framework 中，与用于连接数据库的数据提供程序无关。

使用 SADataAdapter 时，必须连接到数据库以填充 DataTable，并使用对 DataTable 的更改更新数据库。不过，填充 DataTable 后，可以在 DataTable 与数据库断开的情况下对其进行修改。

如果不希望立即将更改应用到数据库，可以使用 WriteXml 方法将 DataSet（包括数据和/或模式）写入 XML 文件。这样，以后就可以通过使用 ReadXml 方法装载 DataSet 来应用更改。以下提供了两个示例。

```
ds.WriteXml("Employees.xml");
ds.WriteXml("EmployeesWithSchema.xml", XmlWriteMode.WriteSchema);
```

有关详细信息，请参见 .NET Framework 文档中的 WriteXml 和 ReadXml 部分。

调用 Update 方法将 DataSet 的更改应用于数据库时，SADataAdapter 会对已经做出的更改进行分析，然后根据需要调用相应的 INSERT、UPDATE 或 DELETE 语句。使用 DataSet 时，只能对一个表中的数据进行更改（插入、更新或删除）。不能更新基于连接的数据集。如果其他用户锁定了您试图更新的行，则将抛出异常。

警告! 对 DataSet 所做的所有更改都是在断开连接的情况下完成的。应用程序未锁定数据库中的这些行。如果在您的更改应用到数据库之前其他用户更改了您正修改的数据，则在将 DataSet 的更改应用到数据库时，会出现一些冲突。您的应用程序必须设计为能够解决这样的冲突。

解决使用 SADataAdapter 时的冲突

使用 SADataAdapter 时，不会锁定数据库的行。这意味着将来自 DataSet 的更改应用到数据库时可能会引起冲突。应用程序中应包含解决或记录产生的冲突的逻辑。

应用程序逻辑应解决的一些冲突如下：

- **唯一主键** – 如果两个用户向一个表中插入新行，则每个行都必须有一个唯一的主键。对于包含 AUTOINCREMENT 主键的表，DataSet 中的值可能与数据源中的值不同步。
- **对同一值的更新** – 如果两个用户修改同一值，则应用程序应包含用于决定哪个值正确的逻辑。
- **模式更改** – 如果某个用户修改已在 DataSet 中更新的表的模式，则在将更改应用于数据库时，更新会失败。
- **数据并发** – 并发应用程序应看到一致的数据集。SADataAdapter 不会锁定其读取的行，因此在您检索了 DataSet 并且脱机工作时，其他用户可以更新数据库中的值。

许多这样的潜在问题都可通过使用 SACommand、SADataReader 和 SATransaction 对象将更改应用到数据库来避免。建议使用 SATransaction 对象，因为它允许您设置事务的隔离级别，并将行锁定，这样其他用户便无法修改这些行。

若要简化冲突解决过程，可以自行设计 INSERT、UPDATE 或 DELETE 语句作为存储过程调用。通过在存储过程中包含 INSERT、UPDATE 和 DELETE 语句，可以捕获操作失败时的错误。除了语句外，还可以在存储过程中添加错误处理逻辑，以便在操作失败时采取适当行动，例如将错误记录到日志文件，或再次尝试操作。

SADataAdapter：使用 Fill 将数据读取到 DataTable 中

使用 SADataAdapter 可查看结果集，方法是使用 Fill 方法将查询结果填充到 DataTable 中，然后将该 DataTable 绑定到显示网格。

设置 SADataAdapter 时，可指定用于返回结果集的 SQL 语句。调用 Fill 填充 DataTable 时，所有行都是使用只进、只读游标在一次操作中读取的。读取结果中的所有行后，将关闭游标。使用 Update 方法，可将对 DataTable 行的更改反映到数据库中。

可以使用 SADataAdapter 对象检索基于连接的结果集。但是，只能对一个表中的数据进行更改（插入、更新或删除）。不能更新基于连接的数据集。

警告！ 对 DataTable 所做的任何更改均独立于原始数据库表。应用程序未锁定数据库中的这些行。如果在您的更改应用到数据库之前其他用户更改了您正修改的数据，则在将 DataTable 的更改应用到数据库时，会出现一些冲突。您的应用程序必须设计为能够解决这样的冲突。

使用 DataTable 的 C# SADataAdapter Fill 示例

以下示例显示如何使用 SADataAdapter 填充 DataTable。它将创建名为 Results 的新 DataTable 对象和新的 SADataAdapter 对象。SADataAdapter Fill 方法用于向 DataTable 填充查询结果。DataTable 随之绑定到屏幕上的网格。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
DataTable dt = new DataTable("Results");
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees",
conn);
da.Fill(dt);
conn.Close();
dataGridView1.DataSource = dt;
```

使用 DataSet 的 C# SADataAdapter Fill 示例

以下示例显示如何使用 SADataAdapter 填充 DataTable。它将创建新的 DataSet 对象和新的 SADataAdapter 对象。SADataAdapter Fill 方法用于在 DataSet 中创建名为 Results 的 DataTable 表，然后向其填充查询结果。Results DataTable 随之绑定到屏幕上的网格。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
DataSet ds = new DataSet();
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees",
conn);
da.Fill(ds, "Results");
conn.Close();
dataGridView1.DataSource = ds.Tables["Results"];
```

SADataAdapter：使用 FillSchema 设置 DataTable 的格式

SADataAdapter 允许您使用 FillSchema 方法将 DataTable 的模式配置为与特定查询的模式相匹配。DataTable 中的列属性将与 SADataAdapter 对象的 SelectCommand 属性相匹配。与 Fill 方法不同的是，DataTable 中不会存储任何行。

使用 DataTable 的 C# SADataAdapter FillSchema 示例

以下示例显示如何使用 FillSchema 方法建立与结果集模式相同的新的 DataTable 对象。Additions DataTable 随之绑定到屏幕上的网格。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees",
conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
conn.Close();
dataGridView1.DataSource = dt;
```

使用 DataSet 的 C# SADataAdapter FillSchema 示例

以下示例显示如何使用 FillSchema 方法建立与结果集模式相同的新的 DataTable 对象。可使用 Merge 方法将 DataTable 添加到 DataSet。Additions DataTable 随之绑定到屏幕上的网格。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees",
conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
DataSet ds = new DataSet();
ds.Merge(dt);
conn.Close();
dataGridView1.DataSource = ds.Tables["Additions"];
```

SADataAdapter：使用 Update 插入行

显示如何使用 SADataAdapter 的 Update 方法向表中添加行的示例。

C# SADataAdapter Insert 示例

在本示例中，将使用 SADataAdapter 的 SelectCommand 属性和 Fill 方法将 Departments 表提取到 DataTable 中。然后，向 DataTable 添加两个新行，并使用 SADataAdapter 的 InsertCommand 属性和 Update 方法更新 DataTable 中的 Departments 表。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600", conn);
deleteCmd.ExecuteNonQuery();

SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
```

```
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.InsertCommand = new SACommand(
    "INSERT INTO Departments( DepartmentID, DepartmentName ) " +
    "VALUES( ?, ? )", conn );
da.InsertCommand.UpdatedRowSource =
    UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADbType = SADbType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

DataRow row1 = dataTable.NewRow();
row1[0] = 600;
row1[1] = "Eastern Sales";
dataTable.Rows.Add( row1 );

DataRow row2 = dataTable.NewRow();
row2[0] = 700;
row2[1] = "Western Sales";
dataTable.Rows.Add( row2 );

rowCount = da.Update( dataTable );

dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

SADataAdapter：使用 Update 删除行

显示如何使用 SADataAdapter 的 Update 方法删除表中的行的示例。

C# SADataAdapter Delete 示例

在本示例中，将向 Departments 表添加两个新行，并使用 SADataAdapter 的 SelectCommand 属性和 Fill 方法将该表提取到 DataTable 中。然后，删除 DataTable 中的一些行，并使用 SADataAdapter 的 DeleteCommand 属性和 Update 方法更新 DataTable 中的 Departments 表。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand prepCmd = new SACommand("", conn);
prepCmd.CommandText =
```

```
    "DELETE FROM Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();

SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn);
da.DeleteCommand = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID = ?",
    conn);
da.DeleteCommand.UpdatedRowSource = UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.DeleteCommand.Parameters.Add(parm);

DataTable dataTable = new DataTable("Departments");
int rowCount = da.Fill(dataTable);

foreach (DataRow row in dataTable.Rows)
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row.Delete();
    }
}
rowCount = da.Update(dataTable);

dataTable.Clear();
rowCount = da.Fill(dataTable);
conn.Close();
dataGridView1.DataSource = dataTable;
```

SADataAdapter：使用 Update 更新行

显示如何使用 SADataAdapter 的 Update 方法更新表中的行的示例。

C# SADataAdapter Update 示例

在本示例中，将向 Departments 表添加两个新行，并使用 SADataAdapter 的 SelectCommand 属性和 Fill 方法将该表提取到 DataTable 中。然后，修改 DataTable 中的一些值，并使用 SADataAdapter 的 UpdateCommand 属性和 Update 方法更新 DataTable 中的 Departments 表。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand prepCmd = new SACommand("", conn);
```

```
prepCmd.CommandText =
    "DELETE FROM Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();

SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.UpdateCommand = new SACommand(
    "UPDATE Departments SET DepartmentName = ? " +
    "WHERE DepartmentID = ?",
    conn );
da.UpdateCommand.UpdatedRowSource = UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.UpdateCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.UpdateCommand.Parameters.Add( parm );

DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

foreach ( DataRow row in dataTable.Rows )
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row[1] = (string)row[1] + "_Updated";
    }
}
rowCount = da.Update( dataTable );

dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

SADataAdapter：检索新插入行的主键值

如果正在更新的表包含自动增量主键，请使用 UUID；或者，如果主键来自主键池，则可以使用存储过程获取该数据源生成的主键值。

C# SADataAdapter 主键示例

以下示例显示如何获得为新插入行所生成的主键。在本示例中，将使用 SADataAdapter 对象调用 SQL 存储过程和 SAParameter 对象以对其返回的主键进行检索。出于演示的目的，本示例将创建一个示例表 (adodotnet_primarykey) 和用于插入行和返回主键值的存储过程 (sp_adodotnet_primarykey)。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();

SACommand cmd = conn.CreateCommand();

cmd.CommandText = "DROP TABLE adodotnet_primarykey";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey ("
+
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE or REPLACE PROCEDURE
sp_adodotnet_primarykey(" +
    "out p_id int, in p_name char(40) )" +
    "BEGIN " +
    "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
    "SELECT @@IDENTITY INTO p_id;" +
    "END";
cmd.ExecuteNonQuery();

SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

da.SelectCommand = new SACommand(
    "SELECT * FROM adodotnet_primarykey", conn);

da.InsertCommand = new SACommand(
    "sp_adodotnet_primarykey", conn);
da.InsertCommand.CommandType = CommandType.StoredProcedure;
da.InsertCommand.UpdatedRowSource =
UpdateRowSource.OutputParameters;

SAParameter parmId = new SAParameter();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
parmId.SourceColumn = "ID";
parmId.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmId);
```

```
SAParameter parmName = new SAParameter();
parmName.SADbType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
parmName.SourceColumn = "Name";
parmName.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmName);

DataTable dataTable = new DataTable("Departments");
da.FillSchema(dataTable, SchemaType.Source);

DataRow row = dataTable.NewRow();
row[0] = -1;
row[1] = "R & D --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -2;
row[1] = "Marketing --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -3;
row[1] = "Sales --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -4;
row[1] = "Shipping --- Adapter";
dataTable.Rows.Add(row);

DataSet ds = new DataSet();
ds.Merge(dataTable);
da.Update(ds, "Departments");

conn.Close();
dataGridView1.DataSource = ds.Tables["Departments"];
```

BLOB

读取长字符串值或二进制数据时，可以使用一些方法分段读取数据。对于二进制数据，可使用 GetBytes 方法；而对于字符串数据，则可使用 GetChars 方法。否则，BLOB 数据的处理方法与从数据库读取的任何其它数据的处理方法相同。

C# GetChars BLOB 示例

以下示例读取结果集中的三列。前两列为整数，而第三列的类型为 LONG VARCHAR。可使用 GetChars 方法读取第三列，并以 100 字符为单位计算其长度。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand cmd = new SACommand("SELECT * FROM MarketingInformation",
conn);
SADataReader reader = cmd.ExecuteReader();

int idValue;
```

```
int productIdValue;
int length = 100;
char[] buf = new char[length];
while (reader.Read())
{
    idValue = reader.GetInt32(0);
    productIdValue = reader.GetInt32(1);
    long blobLength = 0;
    long charsRead;
    while ((charsRead = reader.GetChars(2, blobLength, buf, 0,
length))
            == (long)length)
    {
        blobLength += charsRead;
    }
    blobLength += charsRead;
}
reader.Close();
conn.Close();
```

时间值

.NET Framework 没有 Time 结构。要从 SAP Sybase IQ 读取时间值，必须使用 GetTimeSpan 方法。使用此方法会将数据作为 .NET Framework TimeSpan 对象返回。

C# TimeSpan 示例

以下示例使用 GetTimeSpan 方法将时间作为 TimeSpan 返回。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand cmd = new SACommand("SELECT 123, CURRENT TIME", conn);
SADataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    int ID = reader.GetInt32(0);
    TimeSpan time = reader.GetTimeSpan(1);
}
reader.Close();
conn.Close();
```

存储过程

可以将 SQL 存储过程用于 SAP Sybase IQ .NET 数据提供程序。

ExecuteReader 方法用于调用返回结果集的存储过程，而 ExecuteNonQuery 方法用于调用不返回结果集的存储过程。ExecuteScalar 方法用于调用仅返回单个值的存储过程。

可使用 SAParameter 对象向存储过程传递参数。

使用参数调用 C# 存储过程的示例

以下示例显示两种调用存储过程并向其传递参数的方法。本示例使用 SADataReader 提取存储过程所返回的结果集。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
bool method1 = true;

SACommand cmd = new SACommand("", conn);
if (method1)
{
    cmd.CommandText = "ShowProductInfo";
    cmd.CommandType = CommandType.StoredProcedure;
}
else
{
    cmd.CommandText = "call ShowProductInfo(?)";
    cmd.CommandType = CommandType.Text;
}

SAParameter param = cmd.CreateParameter();
param.SADbType = SADbType.Integer;
param.Direction = ParameterDirection.Input;
param.Value = 301;
cmd.Parameters.Add(param);

SADataReader reader = cmd.ExecuteReader();
reader.Read();
int ID = reader.GetInt32(0);
string name = reader.GetString(1);
string description = reader.GetString(2);
decimal price = reader.GetDecimal(6);
reader.Close();

listBox1.BeginUpdate();
listBox1.Items.Add("Name=" + name +
    " Description=" + description + " Price=" + price);
listBox1.EndUpdate();

conn.Close();
```

事务处理

利用 SAP Sybase IQ .NET 数据提供程序，可以使用 SATransaction 对象将语句组合到一起。每条语句都以 COMMIT 或 ROLLBACK 结尾，前者将使您对数据库的更改具有永久性，后者则取消事务中的所有操作。事务完成后，必须创建一个新的 SATransaction 对象以进行进一步更改。此行为有别于 ODBC 和嵌入式 SQL，使用后两者时，事务在执行 COMMIT 或 ROLLBACK 后继续存在，直到事务被关闭。

如果不创建事务，缺省情况下 SAP Sybase IQ .NET 数据提供程序将在自动提交模式下运行。每个插入、更新或删除后都有隐式 COMMIT，操作一旦完成，便已对数据库进行更改。在这种情况下，更改无法回退。

事务的隔离级别设置
缺省情况下对事务使用数据库隔离级别。可以选择在开始事务时使用 IsolationLevel 属性为事务指定隔离级别。指定的隔离级别适用于事务中执行的所有语句。SQL Anywhere .NET 数据提供程序支持快照隔离。

执行 SQL 语句时 SAP Sybase IQ 使用的锁取决于事务的隔离级别。

分布式事务处理
.NET 2.0 framework 引入了一个新命名空间 System.Transactions，其中包含用于编写事务性应用程序的类。客户端应用程序可以通过一个或多个参与者创建及参与分布式事务。客户端应用程序可以使用 TransactionScope 类隐式地创建事务。该连接对象可以检测周围存在的由 TransactionScope 创建的事务并且自动征用。客户端应用程序还可以创建一个 CommittableTransaction 并且调用 EnlistTransaction 方法进行征用。SAP Sybase IQ .NET 数据提供程序支持此功能。分布式事务具有很高的性能开销。建议对于非分布式事务使用数据库事务。

C# SATransaction 示例
以下示例显示如何将 INSERT 包装在事务中，以便提交或回退事务。可使用 SATransaction 对象创建事务，然后使用 SACommand 对象将事务链接到 SQL 语句的执行。指定隔离级别 2 (RepeatableRead) 以防止其他数据库用户更新该行。提交或回退事务后将解除对该行的锁定。如果不使用事务，SAP Sybase IQ .NET 数据提供程序将在自动提交模式下运行，您将无法回退对数据库所做的任何更改。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
string stmt = "UPDATE Products SET UnitPrice = 2000.00 " +
    "WHERE Name = 'Tee shirt'";
bool goAhead = false;

SATransaction trans =
conn.BeginTransaction(SAIsolationLevel.RepeatableRead);
SACommand cmd = new SACommand(stmt, conn, trans);
int rowsAffected = cmd.ExecuteNonQuery();
if (goAhead)
    trans.Commit();
else
    trans.Rollback();
conn.Close();
```

错误处理

应用程序应设计为可处理出现的任何错误。

只要执行中出现错误，SAP Sybase IQ .NET 数据提供程序就会创建 SAException 对象并抛出异常。每个 SAException 对象都包括 SAError 对象列表，而这些错误对象包括错误消息和代码。

错误与冲突不同。冲突发生在将更改应用于数据库时。应用程序中应包括一个用于计算正确值或在发生冲突时记录冲突的过程。

C# 错误处理示例

以下 C# 代码将创建一个按钮单击处理程序，用于打开与 SAP Sybase IQ 示例数据库的连接。如果无法建立连接，异常处理程序将显示一条或多条消息。

```
private void button1_Click(object sender, EventArgs e)
{
    SAConnection conn = new SAConnection("Data Source=Sybase IQ
Demo");
    try
    {
        conn.Open();
    }
    catch (SAException ex)
    {
        for (int i = 0; i < ex.Errors.Count; i++)
        {
            MessageBox.Show(ex.Errors[i].Source + " : " +
             ex.Errors[i].Message + " (" +
             ex.Errors[i].NativeError.ToString() + ")",
             "Failed to connect");
        }
    }
}
```

Visual Basic 错误处理示例

以下 Visual Basic 代码将创建一个按钮单击处理程序，用于打开与 SAP Sybase IQ 示例数据库的连接。如果无法建立连接，异常处理程序将显示一条或多条消息。

```
Private Sub Button1_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New SAConnection("Data Source=Sybase IQ Demo")
    Try
        conn.Open()
    Catch ex As SAException
        For i = 0 To ex.Errors.Count - 1
            MessageBox.Show(ex.Errors(i).Source & " : " & _
              ex.Errors(i).Message & " (" & _
              ex.Errors(i).NativeError.ToString() & ")", _
              "Failed to connect")
        Next i
    End Try
End Sub
```

Entity Framework 支持

SAP Sybase IQ .NET 数据提供程序支持 Entity Framework 4.3，可从 Microsoft 获得独立的安装包。要使用 Entity Framework 4.3，必须使用 Microsoft 的 NuGet Package Manager 将其添加到 Visual Studio 中。

Entity Framework 的一个新功能是 Code First。它可实现不同的开发工作流：只需写入映射到数据库对象的 C# 或 VB.NET 类即可定义数据模型对象，而不必打开设计器或

定义 XML 映射文件。此外，也可以使用数据标注或 Fluent API 执行其它配置。可使用模型生成数据库模式或映射到现有数据库。

以下是使用模型创建新的数据库对象的示例：

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLAnywhere;

namespace CodeFirstExample
{
    [Table( "EdmCategories", Schema = "DBA" )]
    public class Category
    {
        public string CategoryID { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }

        public virtual ICollection<Product> Products { get; set; }
    }

    [Table( "EdmProducts", Schema = "DBA" )]
    public class Product
    {
        public int ProductId { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
        public string CategoryID { get; set; }

        public virtual Category Category { get; set; }
    }

    [Table( "EdmSuppliers", Schema = "DBA" )]
    public class Supplier
    {
        [Key]
        public string SupplierCode { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

        protected override void OnModelCreating( DbModelBuilder
modelBuilder )
```

```
        {
            modelBuilder.Entity<Supplier>().Property( s =>
s.Name ).IsRequired();
        }
    }

    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new
SAConnectionFactory();
            Database.SetInitializer<Context>( new
DropCreateDatabaseAlways<Context>() );

            using ( var db = new Context( "DSN=Sybase IQ Demo" ) )
            {
                var query = db.Products.ToList();
            }
        }
    }
}
```

要生成并运行此示例，必须添加以下程序集引用：

```
EntityFramework
iAnywhere.Data.SQLAnywhere.v4.0
System.ComponentModel.DataAnnotations
System.Data.Entity
```

以下是有关映射到现有数据库的另一个示例：

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLAnywhere;

namespace CodeFirstExample
{
    [Table( "Customers", Schema = "GROUPO" )]
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }
```

```
        public virtual ICollection<Contact> Contacts { get; set; }
    }

    [Table( "Contacts", Schema = "GROUPO" )]
    public class Contact
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }

        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Contact> Contacts { get; set; }
        public DbSet<Customer> Customers { get; set; }
    }

    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new
SAConnectionFactory();
            Database.SetInitializer<Context>( null );

            using ( var db = new Context( "DSN=SAP Sybase IQ 16
Demo" ) )
            {
                foreach ( var customer in db.Customers.ToList() )
                {
                  Console.WriteLine( "Customer - " + string.Format(
                      "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8},
{9}",
                        customer.ID, customer.SurName,
customer.GivenName,
                    customer.Street, customer.City, customer.State,
                        customer.Country, customer.PostalCode,
                        customer.Phone, customer.CompanyName ) );

                    foreach ( var contact in customer.Contacts )
```

```
                    {
                        Console.WriteLine( "    Contact - " +
string.Format(
                       "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8},
{9}, {10}",
                            contact.ID, contact.SurName,
contact.GivenName,
                            contact.Title,
                            contact.Street, contact.City,
contact.State,
                            contact.Country, contact.PostalCode,
                            contact.Phone, contact.Fax ) );
                    }
                }
            }
        }
    }
}
```

应注意适用于 SQL Server 的 Microsoft .NET Framework 数据提供程序 (SqlClient) 与 SAP Sybase IQ .NET 数据提供程序在实现方式上存在一些细微差异。

1. 其中包括了新类 SAConnectionFactory（用于实现 IDbConnectionFactory）。创建任何数据模型之前，将 Database.DefaultConnectionFactory 设置为 SAConnectionFactory 的实例，如下所示：

   ```
   Database.DefaultConnectionFactory = new SAConnectionFactory();
   ```

2. Entity Framework Code First 的主要原则是按约定进行编码。Entity Framework 通过编码约定推算数据模型。Entity Framework 还可隐式执行多项操作。有时，开发人员可能不了解所有这些 Entity Framework 约定。不过，一些代码约定对 SAP Sybase IQ 无关紧要。SQL Server 和 SAP Sybase IQ 之间存在很大的差异。每个 SQL Server 实例可维护多个数据库，而每个 SAP Sybase IQ 数据库都是单个文件。
   - 如果用户使用无参数构造函数创建用户定义的 DbContext，SqlClient 将通过集成安全性连接到本地计算机上的 SQL Server Express。如果用户已经创建了登录映射，则 SAP Sybase IQ 提供程序将使用集成登录连接到缺省服务器。
   - 如果 Entity Framework 调用 DbDeleteDatabase 或 DbCreateDatabase（仅限于 SQL Server Express Edition），则 SqlClient 将删除现有数据库并创建新数据库。SAP Sybase IQ 提供程序从不删除或创建数据库。它只创建或删除数据库对象（表、关系、约束等）。用户必须首先创建数据库。
   - IDbConnectionFactory.CreateConnection 方法会将字符串参数 "nameOrConnectionString" 视为数据库名称（SQL Server 的初始目录）或连接字符串。如果用户不为 DbContext 提供连接字符串，SqlClient 会将用户定义的 DbContext 类的命名空间用作初始目录，自动连接到本地计算机上的 SQL Express 服务器。对于 SAP Sybase IQ，此参数只包含连接字符串。此时将忽略数据库名称，而使用集成登录。
3. SQL Server SqlClient API 会将带有数据标注属性 TimeStamp 的列映射到 SQL Server 的数据类型 timestamp/rowversion。开发人员有时会对 SQL Server timestamp/rowversion 产生一些误解。SQL Server timestamp/rowversion 数据类型与 SAP Sybase IQ 和大多数其它数据库供应商不同：

- SQL Server timestamp/rowversion 采用二进制 (8)。它不支持组合的日期和时间值。SAP Sybase IQ 支持名为 timestamp 的数据类型，该类型相当于 SQL Server 的 datetime 数据类型。
- SQL Server timestamp/rowversion 的值是唯一的。SAP Sybase IQ timestamp 的值不是唯一的。
- 每次更新行后，SQL Server 的 timestamp/rowversion 值都将发生相应的变化。

SAP Sybase IQ 提供程序不支持 TimeStamp 数据标注属性。

4. 缺省情况下，Entity Framework 4.1 会始终将模式或所有者名称设置为 dbo，它是 SQL Server 的缺省模式。不过，dbo 不适用于 SAP Sybase IQ。对于 SAP Sybase IQ，必须使用数据标注或 Fluent API 根据表名指定模式名称（例如，GROUPO）。以下提供了一个示例：

```
namespace CodeFirstTest
{
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }

       public virtual ICollection<Contact> Contacts { get; set; }
    }

    public class Contact
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }

        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }

    [Table( "Departments", Schema = "GROUPO" )]
    public class Department
```

```
    {
        [Key()]
        public int DepartmentID { get; set; }
        public string DepartmentName { get; set; }
        public int DepartmentHeadID { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Contact> Contacts { get; set; }
        public DbSet<Customer> Customers { get; set; }
        public DbSet<Department> Departments { get; set; }

        protected override void OnModelCreating( DbModelBuilder
modelBuilder )
        {
            modelBuilder.Entity<Contact>().ToTable( "Contacts",
"GROUPO" );
            modelBuilder.Entity<Customer>().ToTable( "Customers",
"GROUPO" );
        }
    }
}
```

SAP Sybase IQ .NET 数据提供程序部署

以下各节介绍如何部署 SAP Sybase IQ .NET 数据提供程序。

SAP Sybase IQ .NET 数据提供程序系统要求

要使用 SAP Sybase IQ .NET 数据提供程序，必须在计算机或手持式设备上安装以下项目：

- .NET Framework 和/或 .NET Compact Framework 2.0 版或更高版本。
- Visual Studio 2005 或更高版本，或者 .NET 语言编译器，例如 C#（仅开发时需要）。

SAP Sybase IQ .NET 数据提供程序所需文件

SAP Sybase IQ .NET 数据提供程序代码位于每个平台的 DLL 中。

Windows 所需的文件

对于 Windows，需要以下 DLL 之一：

- `%IQDIR16%\V2\Assembly\V2\iAnywhere.Data.SQLAnywhere.dll`
- `%IQDIR16%\V2\Assembly\V3.5\iAnywhere.Data.SQLAnywhere.v3.5.dll`
- `%IQDIR16%\V2\Assembly\V4\iAnywhere.Data.SQLAnywhere.v4.0.dll`

DLL 的选择取决于面向的 .NET 的版本。

提供程序的 Windows 版本还需要以下 DLL。

- **policy.16.0.iAnywhere.Data.SQLAnywhere.dll** – 策略文件可用于替换创建应用程序的提供程序版本。只要发布了关于提供程序的更新，Sybase 就会更新策略文件。也有用于 3.5 版提供程序的策略文件 (policy.16.0.iAnywhere.Data.SQLAnywhere.v3.5.dll)，以及用于 4.0 版提供程序的策略文件 (policy.16.0.iAnywhere.Data.SQLAnywhere.v4.0.dll)。
- **dblgen16.dll** – 该语言 DLL 包含了提供程序发出的英文 (en) 消息。还可使用很多其它语种，包括汉语 (zh)、法语 (fr)、德语 (de) 和日语 (jp)。
- **dbcon16.dll** – 此 DLL 包含“**连接到 SQL Anywhere**”窗口支持代码。

Visual Studio 将 .NET 数据提供程序 DLL（iAnywhere.Data.SQLAnywhere.dll 或 iAnywhere.Data.SQLAnywhere.v3.5.dll）与您的程序一同部署到设备。如果未使用 Visual Studio，则必须将数据提供程序 DLL 随应用程序一同复制到设备。它可以位于应用程序所在的目录中，也可以位于 \Windows 目录中。

SAP Sybase IQ .NET 数据提供程序的 dbdata DLL

当 SAP Sybase IQ .NET 数据提供程序第一次被某 .NET 应用程序加载时（通常是在使用 SAConnection 进行数据库连接时），它将解压缩出一个包含提供程序的非托管代码的 DLL。提供程序会将 dbdata16.dll 文件放置在使用以下策略标识的目录的子目录中。

1. 它首先尝试用于卸载的目录是以下第一个被返回的路径:

   由 TMP 环境变量标识的路径。

   由 TEMP 环境变量标识的路径。

   由 USERPROFILE 环境变量标识的路径。

   Windows 目录。
2. 如果标识的目录无法访问，提供程序将尝试使用当前的工作目录。
3. 如果当前的工作目录无法访问，提供程序将尝试使用装载应用程序自身的目录。

子目录名称将采取 GUID 形式，后缀包含版本号、DLL 的位数和用于保证唯一性的索引号。以下是允许的子目录名称的一个示例。

```
{16AA8FB8-4A98-4757-B7A5-0FF22C0A6E33}_1601.x64_1
```

SAP Sybase IQ .NET 数据提供程序的 DLL 注册

安装 SAP Sybase IQ 软件时，会在“全局程序集高速缓存”中注册 Windows 版本的 SAP Sybase IQ .NET 数据提供程序 DLL (%IQDIR%\Assembly\V2\iAnywhere.Data.SQLAnywhere.dll)。在 Windows Mobile 上无需注册该 DLL。

如果要部署 SAP Sybase IQ .NET 数据提供程序，可以使用 Microsoft SDK 附带的 gacutil 实用程序来注册它。

要在部署 SAP Sybase IQ .NET 数据提供程序时将其注册为 DbProviderFactory 实例，必须在 .NET `machine.config` 文件中添加一个条目。必须在 `<DbProviderFactories>` 部分中放入一个类似于下例的条目。

```
<add invariant="iAnywhere.Data.SQLAnywhere"
name="SAP Sybase IQ 16 Data Provider"
description=".Net Framework Data Provider for SAP Sybase IQ 16"
type="iAnywhere.Data.SQLAnywhere.SAFactory,
iAnywhere.Data.SQLAnywhere.v3.5,
    Version=16.0.0.36003, Culture=neutral,
PublicKeyToken=f222fc4333e0d400"/>
```

版本号必须与要安装的提供程序版本一致。配置文件位于 `\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG` 中。对于 64 位 Windows 系统，还必须修改 `Framework64` 树下的第二个配置文件。

.NET 跟踪支持

SAP Sybase IQ .NET 数据提供程序支持使用 .NET 跟踪功能进行跟踪。

缺省情况下，跟踪功能被禁用。要启用跟踪功能，请在应用程序的配置文件中指定跟踪源。

以下是配置文件的一个示例：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
 <source name="iAnywhere.Data.SQLAnywhere"
         switchName="SASourceSwitch"
         switchType="System.Diagnostics.SourceSwitch">
  <listeners>
   <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
   <add name="EventListener"
        type="System.Diagnostics.EventLogTraceListener"
        initializeData="MyEventLog"/>
   <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
   <remove name="Default"/>
  </listeners>
 </source>
</sources>
<switches>
 <add name="SASourceSwitch" value="All"/>
 <add name="SATraceAllSwitch" value="1" />
 <add name="SATraceExceptionSwitch" value="1" />
 <add name="SATraceFunctionSwitch" value="1" />
 <add name="SATracePoolingSwitch" value="1" />
 <add name="SATracePropertySwitch" value="1" />
</switches>
```

```
</system.diagnostics>
</configuration>
```

配置文件中引用了上述四种类型的跟踪监听器。

- **ConsoleTraceListener** – 跟踪或调试输出将被定向到标准输出或标准错误流。使用 Microsoft Visual Studio 时，输出将显示在“**输出**”窗口中。
- **DefaultTraceListener** – 此监听器将以名称 "Default" 自动添加到 Debug.Listeners 和 Trace.Listeners 集合中。跟踪或调试输出将被定向到标准输出或标准错误流。使用 Microsoft Visual Studio 时，输出将显示在“**输出**”窗口中。要避免 `ConsoleTraceListener` 产生重复输出，可删除此监听器。
- **EventLogTraceListener** – 跟踪或调试输出将被定向到在 `initializeData` 选项中标识的 `EventLog`。在本示例中，事件日志名为 `MyEventLog`。向系统事件日志写入流需要管理员权限，不推荐使用此方法调试应用程序。
- **TextWriterTraceListener** – 跟踪或调试输出将被定向到 `TextWriter`，它可将流写入到在 `initializeData` 选项中标识的文件。

要禁用跟踪上述任何跟踪监听器，请在 `<listeners>` 下删除相应的 `add` 条目。

跟踪配置信息位于应用程序的项目文件夹下的 `App.config` 文件中。如果不存在该文件，可使用 Visual Studio 进行创建并添加到项目，方法是选择“**Add**”»“**New Item**”，然后选择“**Application Configuration File**”。

可为任何监听器指定的 `traceOutputOptions` 如下：

- **Callstack** – 编写由 Environment.StackTrace 属性的返回值表示的调用堆栈。
- **DateTime** – 编写日期和时间。
- **LogicalOperationStack** – 编写由 CorrelationManager.LogicalOperationStack 属性的返回值表示的逻辑操作堆栈。
- **无** – 不编写任何元素。
- **ProcessId** – 编写由 Process.Id 属性的返回值表示的过程标识。
- **ThreadId** – 编写由当前线程的 Thread.ManagedThreadId 属性的返回值表示的线程标识。
- **Timestamp** – 编写由 System.Diagnostics.Stopwatch.GetTimeStamp 方法的返回值表示的时间戳。

之前显示的示例配置文件仅用于为 `TextWriterTraceListener` 指定跟踪输出选项。

可通过设置特定的跟踪选项来限制跟踪的内容。缺省情况下，所有值为数字的跟踪选项设置均为 0。可设置的跟踪选项包括：

- **SASourceSwitch** – SASourceSwitch 可采用以下任一值。如果设置为 `Off`，则不进行跟踪。

  Off – 不允许任何事件通过。

  Critical – 只允许重要事件通过。

Error – 允许重要事件和错误事件通过。

Warning – 允许重要事件、错误事件和警告事件通过。

Information – 允许重要事件、错误事件、警告事件和信息事件通过。

Verbose – 允许重要事件、错误事件、警告事件、信息事件和详细事件通过。

ActivityTracing – 允许停止事件、开始事件、暂停事件、传输事件和恢复事件通过。

All – 允许所有事件通过。

以下为示例设置。

```
<add name="SASourceSwitch" value="Error"/>
```

- **SATraceAllSwitch** – 将启用所有跟踪选项。您无需再设置任何其它选项，因为已选择了所有的选项。如果选择此选项，则无法禁用单独的选项。例如，以下语句将不会禁用异常跟踪。

```
<add name="SATraceAllSwitch" value="1" />
<add name="SATraceExceptionSwitch" value="0" />
```

- **SATraceExceptionSwitch** – 记录所有异常。跟踪消息具有以下形式。

```
<Type|ERR> message='message_text'[ nativeError=error_number]
```

仅当具有 SAException 对象时，才会显示 nativeError=error_number 文本。

- **SATraceFunctionSwitch** – 记录所有进入/退出函数作用域的操作。跟踪消息具有以下形式之一。

```
enter_nnn <sa.class_name.method_name|API> [object_id#]
[parameter_names]
leave_nnn
```

nnn 是一个整数，它表示范围嵌套级别 1、2、3、……可选的 parameter_names 是由空格分隔的参数名列表。

- **SATracePoolingSwitch** – 记录所有连接池。跟踪消息具有以下形式之一。

```
<sa.ConnectionPool.AllocateConnection|CPOOL>
connectionString='connection_text'
<sa.ConnectionPool.RemoveConnection|CPOOL>
connectionString='connection_text'
<sa.ConnectionPool.ReturnConnection|CPOOL>
connectionString='connection_text'
<sa.ConnectionPool.ReuseConnection|CPOOL>
connectionString='connection_text'
```

- **SATracePropertySwitch** – 记录所有属性设置和检索。跟踪消息具有以下形式之一。

```
<sa.class_name.get_property_name|API> object_id#
<sa.class_name.set_property_name|API> object_id#
```

有关详细信息，请参见 "Tracing Data Access"，网址为 http://msdn.microsoft.com/en-us/library/ms971550.aspx。

配置用于跟踪的 Windows 应用程序

对 TableViewer 示例应用程序启用跟踪所涉及的操作包括：创建引用 ConsoleTraceListener 和 TextWriterTraceListener 监听器的配置文件，删除缺省监听器以及启用所有开关（否则将设置为 0）。

前提条件

计算机上必须安装有 Visual Studio。

过程

1. 在 Visual Studio 中打开 TableViewer 示例。

   启动 Visual Studio 并打开 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\TableViewer\TableViewer.sln`。
2. 创建名为 `App.config` 的应用程序文件，并复制以下配置设置：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
 <source name="iAnywhere.Data.SQLAnywhere"
         switchName="SASourceSwitch"
         switchType="System.Diagnostics.SourceSwitch">
  <listeners>
   <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
   <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
   <remove name="Default"/>
  </listeners>
 </source>
</sources>
<switches>
 <add name="SASourceSwitch" value="All"/>
 <add name="SATraceAllSwitch" value="1" />
 <add name="SATraceExceptionSwitch" value="1" />
 <add name="SATraceFunctionSwitch" value="1" />
 <add name="SATracePoolingSwitch" value="1" />
 <add name="SATracePropertySwitch" value="1" />
</switches>
</system.diagnostics>
</configuration>
```

3. 重建应用程序。
4. 单击“**Debug**”»“**Start Debugging**”。

应用程序执行完成后，跟踪输出将记录在 `bin\Debug\myTrace.log` 文件中。

下一步

在 Visual Studio 的“**Output**”窗口中查看跟踪日志。

.NET 数据提供程序教程

Simple 和 Table Viewer 示例项目随 .NET 数据提供程序提供。

这些示例项目可与 Visual Studio 2005 或更高版本搭配使用。示例项目是用 Visual Studio 2005 开发的。如果您使用的是更高版本，可能需要运行 Visual Studio 中的“**Upgrade Wizard**”。本节还包括帮助您了解使用 Visual Studio 构建 Simple Viewer .NET 数据库应用程序的所有步骤的指南。

教程：使用 Simple 代码示例

Simple 项目使用 .NET 数据提供程序从数据库服务器中获取结果集。

前提条件

计算机上必须安装 Visual Studio 和 .NET Framework。

必须具备 SELECT ANY TABLE 系统特权。

过程

Simple 项目包括在 SAP Sybase IQ 示例中。它是一个包含了 Employees 表中的名称的简单列表框。

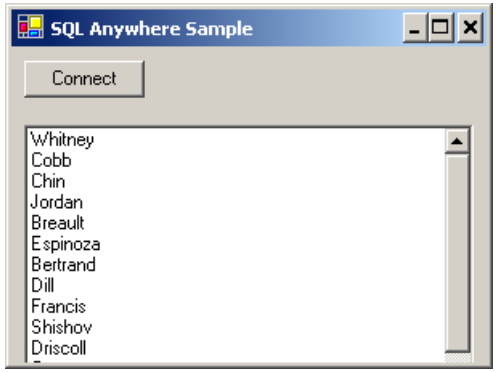
1. 启动 Visual Studio。
2. 单击“**File**”»“**Open**”»“**Project**”。
3. 浏览到 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\SimpleWin32` 并打开 `Simple.sln` 项目。
4. 当在项目中使用 SAP Sybase IQ .NET 数据提供程序时，必须添加一个对数据提供程序的引用。这在 Simple 代码示例中已经完成。要查看数据提供程序 (`iAnywhere.Data.SQLAnywhere`) 的引用，请在“**Solution Explorer**”窗口中打开“**References**”文件夹。
5. 还必须在源代码中添加一条 `using` 指令以引用数据提供程序类。这在 Simple 代码示例中已经完成。要查看 `using` 指令：
   - 打开项目的源代码。在“**Solution Explorer**”窗口中，右击 `Form1.cs`，然后单击“**View Code**”。
     在顶部的 `using` 指令中，您应看到以下行：

     ```
     using iAnywhere.Data.SQLAnywhere;
     ```

C# 项目要求使用这一行指令。如果您使用的是 Visual Basic .NET，则需要将 `Imports` 行添加到源代码中。

6. 单击“**Debug**”»“**Start Without Debugging**”或按下 Ctrl+F5 以运行 Simple 示例。
7. 在“**SQL Anywhere Sample**”窗口中单击“**Connect**”。

   应用程序连接到 SAP Sybase IQ 示例数据库并将每个职员的姓氏放在窗口中，如下图所示：



8. 关闭“**SQL Anywhere Sample**”窗口以关闭应用程序并断开与示例数据库的连接。这还会关闭数据库服务器。

您已构建并执行了一个简单 .NET 应用程序，此程序使用 SAP Sybase IQ .NET 数据提供程序从 SAP Sybase IQ 数据库服务器获取结果集。

教程：使用 Table Viewer 代码示例

TableViewer 项目使用 .NET 数据提供程序连接到数据库，执行 SQL 语句并使用 DataGrid 对象显示结果。

前提条件

计算机上必须安装 Visual Studio 和 .NET Framework。

必须具备 SELECT ANY TABLE 系统特权。

过程

TableViewer 项目包含在 SAP Sybase IQ 示例中。Table Viewer 项目比 Simple 项目更复杂。可以用其连接到数据库、选择表以及在数据库中执行 SQL 语句。

1. 启动 Visual Studio。
2. 单击“**File**”»“**Open**”»“**Project**”。
3. 浏览到 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\TableViewer` 并打开 `TableViewer.sln` 项目。

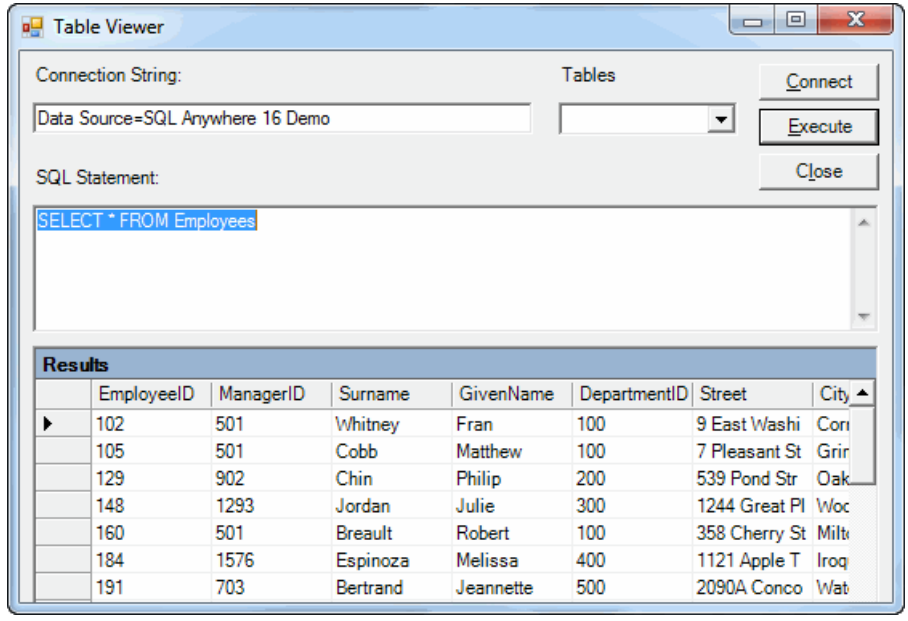
4. 要在项目中使用 SAP Sybase IQ .NET 数据提供程序，必须添加一个对数据提供程序 DLL 的引用。这在 Table Viewer 代码示例中已经完成。要查看数据提供程序 (`iAnywhere.Data.SQLAnywhere`) 的引用，请在“**Solution Explorer**”窗口中打开“**References**”文件夹。
5. 还必须在源代码中添加一条 `using` 指令以引用数据提供程序类。这在 Table Viewer 代码示例中已经完成。要查看 `using` 指令：
   - 打开项目的源代码。在“**Solution Explorer**”窗口中，右击 `TableViewer.cs`，然后单击“**View Code**”。
   - 在顶部的 `using` 指令中，您应看到以下行：

     ```
     using iAnywhere.Data.SQLAnywhere;
     ```

     C# 项目要求使用这一行指令。如果您使用的是 Visual Basic，则需要将 `Imports` 行添加到源代码中。
6. 单击“**Debug**”»“**Start Without Debugging**”或按下 Ctrl+F5 以运行 Table Viewer 示例。

   应用程序将连接到 SAP Sybase IQ 示例数据库。
7. 在“**Table Viewer**”窗口中单击“**Connect**”。
8. 在“**Table Viewer**”窗口中单击“**Execute**”。

   应用程序会检索示例数据库中的 Employees 表的数据，然后将查询结果放入“**Results**”DataGrid 中，如下图所示：



您还可以通过此应用程序执行其它 SQL 语句：在“**SQL Statement**”窗格中键入 SQL 语句，然后单击“**Execute**”。

9. 关闭“**Table Viewer**”窗口以关闭应用程序并断开与示例数据库的连接。这还会关闭数据库服务器。

您已构建并执行了一个 .NET 应用程序，此程序使用 .NET 数据提供程序连接到数据库，执行 SQL 语句并使用 DataGrid 对象显示结果。

教程：使用 Visual Studio 开发简单的 .NET 数据库应用程序

本节包括帮助您了解使用 Visual Studio 构建 Simple Viewer .NET 数据库应用程序的所有步骤的指南。

前提条件

必须具备 SELECT ANY TABLE 系统特权。

第 1 课：创建表查看器

在本课中，将使用 Microsoft Visual Studio、Server Explorer 和 SAP Sybase IQ .NET 数据提供程序创建一个访问 SAP Sybase IQ 示例数据库中某个表的应用程序，它可以检查行并执行更新。

前提条件

计算机上必须安装 Visual Studio 和 .NET Framework。

本课假定您拥有以下教程开头的“特权”部分中列出的角色和特权：教程：使用 Visual Studio 开发简单的 .NET 数据库应用程序。

过程

本教程以 Visual Studio 和 .NET Framework 为基础。完整的应用程序可在 ADO.NET 项目 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\SimpleViewer\SimpleViewer.sln` 中找到。

1. 启动 Visual Studio。
2. 单击“**File**”»“**New**”»“**Project**”。

   随即出现“**New Project**”窗口。

   a. 在“**New Project**”窗口的左侧窗格中，单击“**Visual Basic**”或“**Visual C#**”选择编程语言。
   b. 在“**Windows**”子类中，单击“**Windows Application**”(VS 2005)或“**Windows Forms Application**”(VS 2008/2010)。
   c. 在项目“**Name**”字段中，键入 `MySimpleViewer`。
   d. 单击“**OK**”创建新项目。

3. 单击“**View**”»“**Server Explorer**”。
4. 在“**Server Explorer**”窗口中，右击“**Data Connections**”，然后单击“**Add Connection**”。
5. 在“**Add Connection**”窗口中：
   a. 如果从未对其它项目使用“**Add Connection**”，那么您会看到一个数据源列表。在显示的数据源列表中单击“**SQL Anywhere**”。
      如果之前已经使用“**Add Connection**”，则单击“**Change**”以将数据源更改为“**SQL Anywhere**”。
   b. 在“**Data Source**”下面，单击“**ODBC Data Source Name**”，然后键入 `Sybase IQ Demo`。

      注意： 在 64 位 Windows 上使用 Visual Studio Add Connection 向导时，只有 64 位系统数据源名称 (DSN) 会包含在用户数据源名称中。任何 32 位系统数据源名称都不会显示。在 Visual Studio 的 32 位设计环境中，Test Connection 按钮会尝试使用 64 位系统 DSN 在 32 位环境中相应的系统 DSN 建立连接。如果 32 位系统 DSN 不存在，则测试失败。
   c. 单击“**Test Connection**”以验证您可以连接到示例数据库。
   d. 单击“**OK**”。

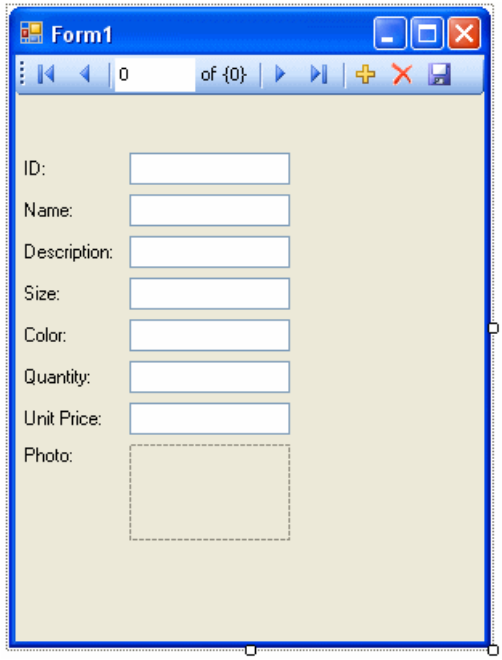
   名为 `Sybase IQ.demo` 的新连接随即出现在“**Server Explorer**”窗口中。
6. 在“**Server Explorer**”窗口中展开 `Sybase IQ.demo` 连接，直到看到表名称为止。

   （仅限 Visual Studio 2005）尝试以下操作：
   a. 右击 Products 表，然后单击“**Show Table Data**”。
      这样，窗口中会出现 Products 表的行和列。
   b. 关闭表数据窗口。
7. 单击“**Data**”»“**Add New Data Source**”。
8. 在“**Data Source Configuration Wizard**”中，执行以下操作：
   a. 在“**Data Source Type**”页面上，单击“**Database**”，然后单击“**Next**”。
   b. （仅限 Visual Studio 2010）在“**Database Model**”页面中，单击“**Dataset**”，然后单击“**Next**”。
   c. 在“**Data Connection**”页面上，单击 `Sybase IQ.demo`，然后单击“**Next**”。
   d. 在“**Save The Connection String**”页面上，确认已选择“**Yes, Save The Connection As**”，然后单击“**Next**”。
   e. 在“**Choose Your Database Objects**”页面上，单击“**Tables**”，然后单击“**Finish**”。
9. 单击“**Data**”»“**Show Data Sources**”。

   随即出现“**Data Sources**”窗口。

   在“**Data Sources**”窗口中展开 Products 表。

a. 从下拉列表中单击“Products”，然后单击“**Details**”。
b. 从下拉列表中单击“Photo”，然后单击“**Picture Box**”。
c. 单击“Products”并将其拖动到您的窗体 (Form1) 中。



数据集控件和几个带标签的文本字段出现在窗体上。

10. 在窗体上，单击“Photo”旁边的图片框。
    a. 将框形状更改为方形。
    b. 单击图片框右上角的右箭头键。
    “**Picture Box Tasks**”窗口随即打开。
    c. 从“**Size Mode**”下拉列表中，单击“**Zoom**”。
    d. 要关闭“**Picture Box Tasks**”窗口，请单击窗口外的任何位置。
11. 构建并运行项目。
    a. 单击“**Build**”»“**Build Solution**”。
    b. 单击“**Debug**”»“**Start Debugging**”。
    应用程序连接到 SAP Sybase IQ 示例数据库，并在文本框和图片框中显示 Products 表的第一行。

c. 可以使用控件上的按钮滚动浏览结果集中的行。

d. 可以通过在滚动控件中输入行号直接转到结果集中的该行。

e. 可以使用文本框更新结果集中的值并通过单击“**Save Data**”按钮进行保存。

12. 关闭应用程序，然后保存您的项目。

您现在已经使用 Visual Studio、Server Explorer 和 SAP Sybase IQ .NET 数据提供程序创建了一个简单但强大的 .NET 应用程序。

下一步

在下一课中，将向本课中开发的窗体添加 datagrid 控件。

第 2 课：添加同步数据控件

在本课中，将向在前一课中开发的窗体添加 datagrid 控件。在您浏览结果集时，此控件会自动进行更新。

前提条件

本课假定您拥有在“教程：使用数据库教程中的 Java：使用 Visual Studio 开发简单的 .NET 数据库应用程序。

过程

完整的应用程序可在 ADO.NET 项目 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\ADO.NET\SimpleViewer\SimpleViewer.sln` 中找到。

1. 启动 Visual Studio 并加载 MySimpleViewer 项目。
2. 在“**Data Sources**”窗口中右击“DataSet1”，并单击“**Edit DataSet With Designer**”。
3. 右击“**DataSet Designer**”窗口中的空白区域，并单击“**Add**”»“**TableAdapter**”。
4. 在“**TableAdapter Configuration Wizard**”中：
   a. 在“**Choose Your Data Connection**”页面上，单击“**Next**”。
   b. 在“**Choose A Command Type**”页面上，单击“**Use SQL Statements**”，然后单击“**Next**”。
   c. 在“**Enter A SQL Statement**”页面上，单击“**Query Builder**”。
   d. 在“**Add Table**”窗口上，单击“**Views**”选项卡，单击“**ViewSalesOrders**”，然后单击“**Add**”。
   e. 单击“**Close**”关闭“**Add Table**”窗口。
5. 展开“**Query Builder**”窗口，使窗口中的所有部分都可见。
   a. 展开“**ViewSalesOrders**”窗口，使所有复选框可见。
   b. 单击“**Region**”。
   c. 单击“**Quantity**”。
   d. 单击“**ProductID**”。
   e. 在“**ViewSalesOrders**”窗口下面的网格中，清除“ProductID”列的“**Output**”下面的复选框。
   f. 对于“ProductID”列，在“**Filter**”单元格中键入问号 (?)。这将为 ProductID 生成 WHERE 子句。

   此时构建出的 SQL 查询如下：

```
SELECT   Region, Quantity
FROM     GROUPO.ViewSalesOrders
WHERE    (ProductID = :Param1)
```

6. 按如下步骤修改 SQL 查询：
   a. 将 `Quantity` 更改为 `SUM(Quantity) AS TotalSales`。
   b. 将 `GROUP BY Region` 添加到 WHERE 子句后的查询末尾。

   修改后的 SQL 查询此时如下所示：

```
SELECT   Region, SUM(Quantity) as TotalSales
FROM     GROUPO.ViewSalesOrders
WHERE    (ProductID = :Param1)
GROUP BY Region
```

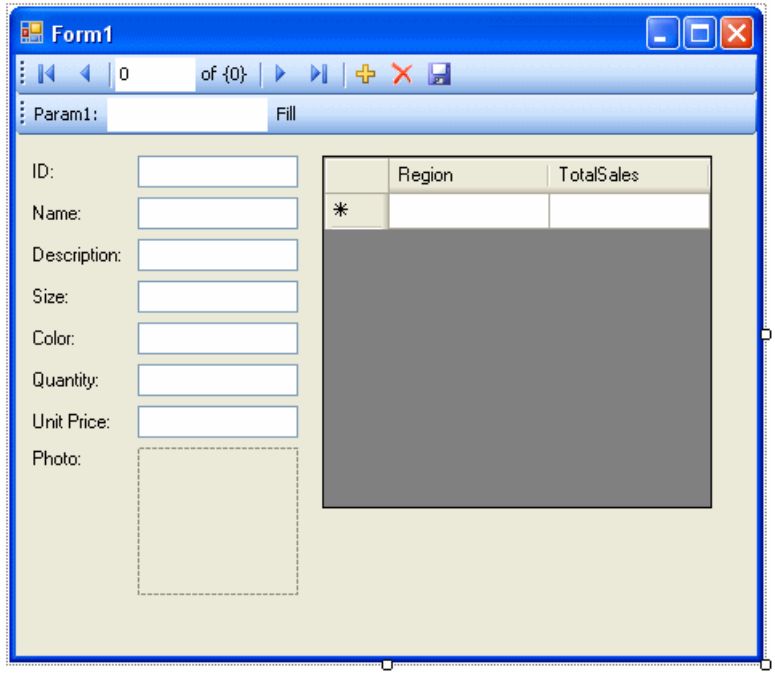
7. 单击“**OK**”。

8. 单击“**Finish**”。

   名为“**ViewSalesOrders**”的新“**TableAdapter**”已添加到“**DataSet Designer**”窗口。

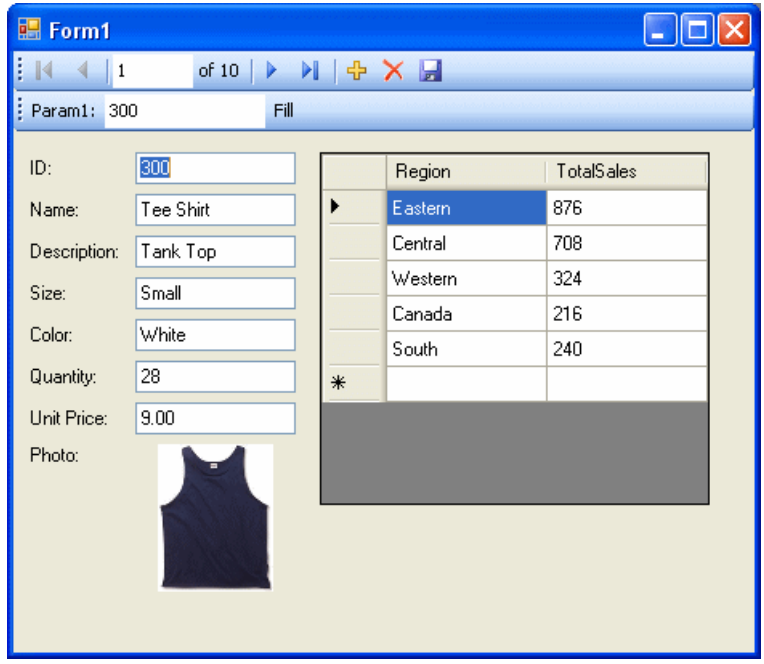
9. 单击窗体设计选项卡 (Form1)。
   - 向右拉伸窗体以为新控件留出空间。

10. 在“**Data Sources**”窗口中展开“ViewSalesOrders”。
    a. 单击“ViewSalesOrders”并从下拉列表中单击“**DataGridView**”。
    b. 单击“ViewSalesOrders”并将其拖动到您的窗体 (Form1) 中。



   datagrid 视图控件出现在窗体上。

11. 构建并运行项目。
    - 单击“**Build**”»“**Build Solution**”。
    - 单击“**Debug**”»“**Start Debugging**”。
    - 在“**Param1**”或“**ProductID**”(VS 2010) 文本框中，输入产品 ID 号（如 300），然后单击“**Fill**”。
      datagrid 视图针对所输入的产品 ID 按区域显示销售额的汇总。



还可以使用窗体上的其它控件在结果集的各行之间移动。

但如果两个控件可以互相保持同步会比较理想。以下几步说明了实现同步的方法。

12. 关闭应用程序，然后保存您的项目。

13. 删除窗体上的“Fill”条，因为您不需要它。

- 在设计窗体 (Form1) 上，右击“**Fill**”一词右侧的“Fill”条，然后单击“**Delete**”。
  “Fill”条即从窗体上删除。

14. 按如下所示同步这两个控件。

a. 在设计窗体 (Form1) 上，右击“ID”文本框，然后单击“**Properties**”。

b. 单击“**Events**”按钮（它显示为一道闪电）。

c. 向下滚动直到找到“**TextChanged**”事件。

d. 单击“**TextChanged**”，然后从下拉列表中单击“**fillToolStripButton_Click**”。如果您在使用 Visual Basic，则事件称为“**FillToolStripButton_Click**”。

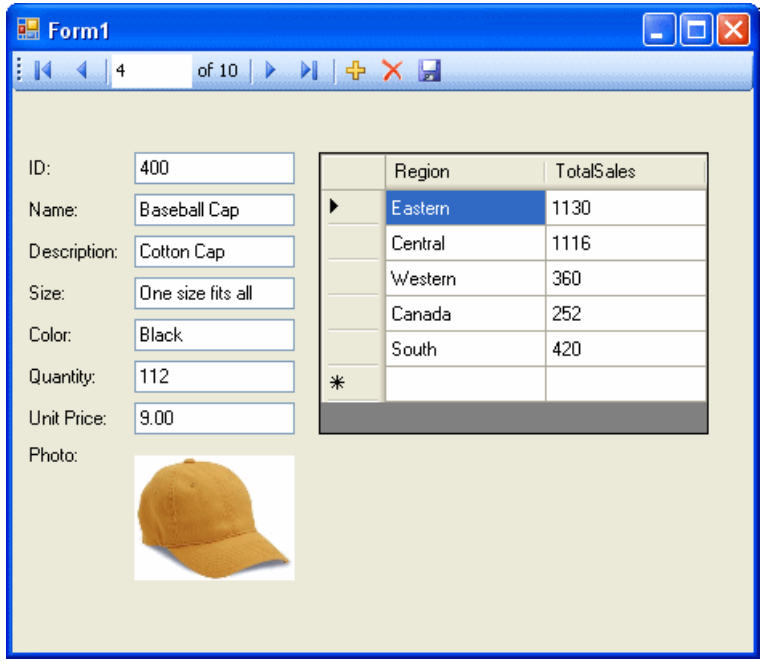
e. 双击“**fillToolStripButton_Click**”，窗体的代码窗口即在 `fillToolStripButton_Click` 事件处理程序上打开。

f. 找到对 `param1ToolStripTextBox` 或 `productIDToolStripTextBox` (VS 2010) 的引用，将其更改为 `iDTextBox`。如果您在使用 Visual Basic，则文本框称为 `IDTextBox`。

g. 重建并运行项目。

15. 应用程序窗体现在带有一个导航控件。

- 当您在结果集中移动时，datagrid 视图会针对当前产品按区域显示更新的销售额汇总。



16. 关闭应用程序，然后保存您的项目。

您现在已添加了会随着您对结果集的浏览自动更新的控件。

在本教程中，您学到了如何利用 Microsoft Visual Studio、Server Explorer 和 SAP Sybase IQ .NET 数据提供程序的强大组合来创建数据库应用程序。

.NET API 参考

命名空间为 `iAnywhere.Data.SQLAnywhere`。

OLE DB 和 ADO 开发

SAP Sybase IQ 中附带了一个供 OLE DB 和 ADO 使用的 OLE DB 提供程序。

OLE DB 是 Microsoft 开发的一组组件对象模型（Component Object Model，简称 COM）接口，它们为应用程序访问不同信息源中存储的数据提供了统一访问接口，并且还提供了实现其它数据库服务的能力。这些接口所支持的 DBMS 功能数与数据存储相符，从而使数据存储能够共享它的数据。

ADO 是通过 OLE DB 系统接口以编程方式访问、编辑以及更新多种数据源的对象模型。ADO 也是由 Microsoft 开发的。大多数使用 OLE DB 编程接口的开发人员在使用该编程接口时都是编写 ADO API 代码，而不是直接编写 OLE DB API 代码。

不要将 ADO 接口同 ADO.NET 混淆。ADO.NET 是一个单独的接口。

有关 OLE DB 和 ADO 编程的文档，请参见 Microsoft Developer Network。有关 OLE DB 和 ADO 开发的特定于 SAP Sybase IQ 的信息，请参阅本文档。

OLE DB

OLE DB 是 Microsoft 的数据访问模型。它使用组件对象模型 (COM) 接口，与 ODBC 不同的是，OLE DB 假定数据源不使用 SQL 查询处理器。

SAP Sybase IQ 中附带了一个名为 `SAOLEDB` 的 *OLE DB 提供程序*。该提供程序可用于当前 Windows 平台。该提供程序不可用于 Windows Mobile 平台。

还可以将用于 ODBC 的 Microsoft OLE DB 提供程序 (MSDASQL) 和 SQL Anywhere ODBC 驱动程序结合使用来访问 SAP Sybase IQ。

使用 SAP Sybase IQ OLE DB 提供程序具有以下几个优点：

- 使用 OLE DB/ODBC Bridge 无法利用某些功能，例如，通过游标更新。
- 如果使用 SAP Sybase IQ OLE DB 提供程序，则不需要在部署中使用 ODBC。
- MSDASQL 允许 OLE DB 客户端使用任何 ODBC 驱动程序，但不保证您能使用每个 ODBC 驱动程序的全部功能。而使用 SAP Sybase IQ 提供程序则可以从 OLE DB 编程环境访问 SAP Sybase IQ 的全部功能。

使用 OLE DB 连接

SAP Sybase IQ 提供了 OLE DB 提供程序作为 ODBC 的备选项。OLE DB 是 Microsoft 的数据访问模型，其采用组件对象模型 (COM) 接口。与 ODBC 不同的是，OLE DB 假定数据源不使用 SQL 查询处理器。虽然 OLE DB 需要 Windows 客户端，但可以使用 OLE DB 来访问 Windows 和 UNIX 服务器。

SAP Sybase IQ OLE DB 支持与 SQL Anywhere 支持不同。SAP Sybase IQ 支持动态（动态滚动）、静态（不敏感）和只进（无滚动）游标，但不支持键集（滚动）游标。在 SAP Sybase IQ 中，无论如何指定，隔离级别始终为 3。

SAP Sybase IQ 支持动态（动态滚动）、静态（不敏感）和只进（无滚动）游标，但不支持键集（滚动）游标。在 SAP Sybase IQ 中，无论如何指定，隔离级别始终为 3。

SAP Sybase IQ 不支持 Windows CE 或通过游标进行远程更新。

其它信息
«编程» >“OLE DB 和 ADO 开发”>“OLE DB 连接参数”

支持的平台

SAP Sybase IQ OLE DB 提供程序设计为使用 Microsoft 数据访问组件（Microsoft Data Access Components，简称 MDAC）2.8 和更高版本。

OLE DB 中的分布式事务

OLE DB 驱动程序可在分布式事务环境中用作资源管理器。

利用 SAP Sybase IQ 进行 ADO 编程

ADO（ActiveX 数据对象）是一种通过 Automation 接口公开的数据访问对象模型，它允许客户端应用程序在事先不了解对象的情况下，在运行时发现对象的方法和属性。Automation 接口允许脚本语言（如 Visual Basic）使用标准的数据访问对象模型。ADO 使用 OLE DB 提供数据访问。

使用 SAP Sybase IQ OLE DB 提供程序，可以从 ADO 编程环境访问 SAP Sybase IQ 的全部功能。

本节介绍使用 Visual Basic 的 ADO 时如何执行基本任务。它不是使用 ADO 进行编程的完整指南。

本节中的代码示例位于 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\VBSampler\vbsampler.sln` 项目文件中。

有关在 ADO 中进行编程的信息，请参见开发工具的文档。

如何使用 Connection 对象连接到数据库

本节介绍一个用于连接到数据库的简单的 Visual Basic 例程。

示例代码
通过将名为 cmdTestConnection 的命令按钮放置在窗体上，并将该例程粘贴到它的 Click 事件中，可以尝试执行该例程。运行程序并单击该按钮，即可建立连接然后断开连接。

```
Private Sub cmdTestConnection_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdTestConnection.Click

    ' Declare variables
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim cAffected As Integer

    On Error GoTo HandleError

    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.Open()
    MsgBox("Connection succeeded")
    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

注释
该示例执行下列任务：

- 声明例程中使用的变量。
- 使用 SAP Sybase IQ OLE DB 提供程序与示例数据库建立连接。
- 使用 Command 对象执行一条简单的语句，用以在数据库服务器消息窗口中显示消息。
- 关闭连接。

如何使用 Command 对象执行语句

本节介绍一个向数据库发送简单 SQL 语句的简单例程。

示例代码
通过将名为 cmdUpdate 的命令按钮放置在窗体上，并将该例程粘贴到它的 Click 事件中，可以尝试执行该例程。运行程序并单击该按钮，即可建立连接，在数据库服务器消息窗口中显示消息，然后断开连接。

```
Private Sub cmdUpdate_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdUpdate.Click

    ' Declare variables
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim cAffected As Integer
```

```
    On Error GoTo HandleError

    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.Open()

    'Execute a command
    myCommand.CommandText = _
        "UPDATE Customers SET GivenName='Liz' WHERE ID=102"
    myCommand.ActiveConnection = myConn
    myCommand.Execute(cAffected)
    MsgBox(CStr(cAffected) & " rows affected.", _
        MsgBoxStyle.Information)

    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

注释
建立连接之后，示例代码创建一个 Command 对象，将其 CommandText 属性设置为更新语句，并将其 ActiveConnection 属性设置为当前连接。然后执行更新语句，并在窗口中显示受更新操作影响的行数。

在本示例中，更新将在执行之后被发送到数据库并被提交。

还可以通过游标执行更新。

如何使用 Recordset 对象获取结果集

ADO Recordset 对象表示查询的结果集。可以使用它查看数据库中的数据。

示例代码
通过将名为 cmdQuery 的命令按钮放置到窗体上，并将该例程粘贴到它的 Click 事件中，可以尝试执行该例程。运行程序并单击该按钮，即可建立连接，在数据库服务器消息窗口中显示消息，执行查询并在窗口中显示前几行，然后断开连接。

```
Private Sub cmdQuery_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdQuery.Click

    ' Declare variables
    Dim i As Integer
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim myRS As New ADODB.Recordset
```

```
    On Error GoTo ErrorHandler

    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.CursorLocation = _
        ADODB.CursorLocationEnum.adUseServer
    myConn.Mode = _
        ADODB.ConnectModeEnum.adModeReadWrite
    myConn.IsolationLevel = _
        ADODB.IsolationLevelEnum.adXactCursorStability
    myConn.Open()

    'Execute a query
    myRS = New ADODB.Recordset
    myRS.CacheSize = 50
    myRS.let_Source("SELECT * FROM Customers")
    myRS.let_ActiveConnection(myConn)
    myRS.CursorType = ADODB.CursorTypeEnum.adOpenKeyset
    myRS.LockType = ADODB.LockTypeEnum.adLockOptimistic
    myRS.Open()

    'Scroll through the first few results
    myRS.MoveFirst()
    For i = 1 To 5
        MsgBox(myRS.Fields("CompanyName").Value, _
            MsgBoxStyle.Information)
        myRS.MoveNext()
    Next

    myRS.Close()
    myConn.Close()
    Exit Sub

ErrorHandler:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

注释

在本例中，Recordset 对象保存对 Customers 表执行查询的结果。For 循环滚动浏览前几行并显示每一行的 CompanyName 值。

这是一个从 ADO 使用游标的简单示例。

Recordset 对象

使用 SAP Sybase IQ 时，ADO Recordset 代表一个游标。可以通过在打开 Recordset 对象之前声明 Recordset 对象的 CursorType 属性来选择游标类型。所选游标类型控制可对 Recordset 执行的操作，并会对性能产生影响。

游标类型
ADO 有其自己对游标类型的命名约定。

下面列出可用的游标类型、相应的游标类型常量以及与它们等价的 SQL Anywhere 类型：

ADO 游标类型	ADO 常量	SAP Sybase IQ 类型
动态游标	adOpenDynamic	动态滚动游标
键集游标	adOpenKeyset	滚动游标
静态游标	adOpenStatic	不敏感游标
只进游标	adOpenForwardOnly	非滚动游标

示例代码
下列代码集为 ADO Recordset 对象设置游标类型：

```
Dim myRS As New ADODB.Recordset
myRS.CursorType = ADODB.CursorTypeEnum.adOpenDynamic
```

使用 Recordset 对象通过游标对行进行更新

SAP Sybase IQ OLE DB 提供程序允许通过游标更新结果集。该功能不能通过 MSDASQL 提供程序实现。

更新记录集
可以通过 Recordset 来更新数据库。

```
Private Sub cmdUpdateThroughCursor_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdUpdateThroughCursor.Click

    ' Declare variables
    Dim i As Integer
    Dim myConn As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim SQLString As String

    On Error GoTo HandleError

    ' Connect
    myConn.Provider = "SAOLEDB"
```

```
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.Open()
    myConn.BeginTrans()
    SQLString = "SELECT * FROM Customers"
    myRS.Open(SQLString, myConn, _
        ADODB.CursorTypeEnum.adOpenDynamic, _
        ADODB.LockTypeEnum.adLockBatchOptimistic)

    If myRS.BOF And myRS.EOF Then
        MsgBox("Recordset is empty!", 16, "Empty Recordset")
    Else
        MsgBox("Cursor type: " & CStr(myRS.CursorType), _
            MsgBoxStyle.Information)
        myRS.MoveFirst()
        For i = 1 To 3
            MsgBox("Row: " & CStr(myRS.Fields("ID").Value), _
                MsgBoxStyle.Information)
            If i = 2 Then
                myRS.Update("City", "Toronto")
                myRS.UpdateBatch()
            End If
            myRS.MoveNext()
        Next i
        myRS.Close()
    End If
    myConn.CommitTrans()
    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub

End Sub
```

注释

如果对 Recordset 使用了 adLockBatchOptimistic 设置，则 myRS.Update 方法不会对数据库本身做任何更改。而是会更新 Recordset 的本地副本。

myRS.UpdateBatch 方法对数据库服务器进行更新，但是不提交更改，因为它位于事务内部。如果 UpdateBatch 方法是在事务外部调用的，则更改将被提交。

myConn.CommitTrans 方法提交所做的更改。Recordset 对象在此之前已被关闭，因此不存在数据的本地副本是否发生更改的问题。

ADO 事务

缺省情况下，使用 ADO 对数据库进行的任何更改都会在执行时被提交。这包括显式更新以及对 Recordset 执行的 UpdateBatch 方法。但是，上一节内容却说明了可以对 Connection 对象使用 BeginTrans、RollbackTrans 或 CommitTrans 方法以使用事务。

事务隔离级别作为 Connection 对象的属性进行设置。IsolationLevel 属性可具有下列值:

ADO 隔离级别	常量	SAP Sybase IQ 级别
未指定	adXactUnspecified	不适用。设置为 0
混沌	adXactChaos	不支持。设置为 0
浏览	adXactBrowse	0
未提交的读操作	adXactReadUncommitted	0
游标稳定性	adXactCursorStability	1
已提交的读操作	adXactReadCommitted	1
可重复的读操作	adXactRepeatableRead	2
已隔离	adXactIsolated	3
可序列化	adXactSerializable	3
快照	2097152	4
语句快照	4194304	5
只读语句快照	8388608	6

OLE DB 连接参数

OLE DB 连接参数由 Microsoft 定义。SAP Sybase IQ OLE DB 提供程序支持这些连接参数的子集。典型的连接字符串如下所示:

```
"Provider=SAOLEDB;Data Source=myDsn;Initial Catalog=myDbn;
    User ID=myUid;Password=myPwd"
```

以下是提供程序支持的 OLE DB 连接参数。某些情况下，OLE DB 连接参数与 SAP Sybase IQ 连接参数完全相同（如 Password）或相似（如 User ID）。请注意，其中的很多连接参数都使用了空格。

- **Provider** – 此参数用于标识 SQL Anywhere OLE DB 提供程序 (SAOLEDB)。
- **User ID** – 此连接参数直接映射到 SAP Sybase IQ UserID (UID) 连接参数。
- **Password** – 此连接参数直接映射到 SAP Sybase IQ Password (PWD) 连接参数。
- **Data Source** – 此连接参数直接映射到 SAP Sybase IQ DataSourceName (DSN) 连接参数。例如: `Data Source=Sybase IQ Demo`.
- **Initial Catalog** – 此连接参数直接映射到 SAP Sybase IQ DatabaseName (DBN) 连接参数。例如: `Initial Catalog=demo`.
- **Location** – 此连接参数直接映射到 SAP Sybase IQ Host 连接参数。该参数值与 Host 参数值具有相同的格式。例如: `Location=localhost:4444`.

- **Extended Properties** – OLE DB 使用此连接参数传入所有 SAP Sybase IQ 特定的连接参数。例如: `Extended Properties="UserID=DBA;DBKEY=V3moj3952B;DBF=demo.db".`

  ADO 使用此参数收集和传入所有它不识别的连接参数。

  某些 Microsoft 连接窗口包含名为“**Prov String**”或“**Provider String**”的字段。该字段的内容作为值传递给 Extended Properties。
- **OLE DB Services** – 此连接参数不经 SAP Sybase IQ OLE DB 提供程序直接处理。它控制 ADO 中的连接池。
- **Prompt** – 此连接参数控制连接尝试处理错误的方式。可能的提示值为 1、2、3 或 4。分别表示 DBPROMPT_PROMPT (1)、DBPROMPT_COMPLETE (2)、DBPROMPT_COMPLETEREQUIRED (3) 和 DBPROMPT_NOPROMPT (4)。

  缺省提示值为 4，表示提供程序不显示连接窗口。将提示值设置为 1 会始终显示连接窗口。将提示值设置为 2 会在初始连接尝试失败时显示连接窗口。如果将提示值设置为 3，则在初始连接尝试失败但提供程序禁用对连接到数据源所不需要的任何信息的控制时显示连接窗口。
- **Window Handle** – 应用程序可以在父窗口句柄适用时传递父窗口句柄，或者在窗口句柄不适用或提供程序不显示任何窗口时传递空指针。窗口句柄值通常为 0 (NULL)。

还可以指定其它 OLE DB 连接参数，但 OLE DB 提供程序会将其忽略。

调用 SAP Sybase IQ OLE DB 提供程序时，它会获取 OLE DB 连接参数的属性值。以下是从 Microsoft 的 RowsetViewer 应用程序获取的一组典型属性值。

```
User ID '<user_id>'
Password '<password>'
Location 'localhost:4444'
Initial Catalog 'demo'
Data Source 'testds'
Extended Properties 'appinfo=api=oledb'
Prompt 2
Window Handle 0
```

提供程序通过此组参数值构造的连接字符串为:

```
'DSN=testds;HOST=localhost:
4444;DBN=demo;UID=<user_id>;PWD=<password>;appinfo=api=oledb'
```

SAP Sybase IQ OLE DB 提供程序使用连接字符串、Window Handle 和 Prompt 值作为它所执行的数据库服务器连接调用的参数。

以下是一个简单的 ADO 连接字符串示例。

```
connection.Open
"Provider=SAOLEDB;UserID=<user_id>;Location=localhost:
4444;Pwd=<password>"
```

ADO 对连接字符串进行分析并将所有无法识别的连接参数传入 Extended Properties 中。调用 SAP Sybase IQ OLE DB 提供程序时，它会获取 OLE DB 连接参数的属性值。以下是一组从使用上述连接字符串的 ADO 应用程序中获取的属性值。

```
User ID ''
Password ''
Location 'localhost:4444'
Initial Catalog ''
Data Source ''
Extended Properties 'UserID=<user_id>;Pwd=<password>'
Prompt 4
Window Handle 0
```

提供程序通过此组参数值构造的连接字符串为：

```
'HOST=localhost:4444; UserID=<user_id>;Pwd=<password>'
```

提供程序使用连接字符串、Window Handle 和 Prompt 值作为它所执行的数据库服务器连接调用的参数。

OLE DB 连接池

适用于 OLE DB 的 .NET Framework 数据提供程序会使用 OLE DB 会话池自动将连接放入连接池中。

当应用程序断开连接时，连接并未真正断开，而是会保持一段时间。当应用程序重新打开一个连接时，ADO/OLE DB 会认出该应用程序使用了完全相同的连接字符串并将重新使用打开的连接。例如，如果应用程序执行“打开/执行/关闭”100 次的操作，则其中只有一次真正的打开和一次真正的关闭。最终的关闭发生在空闲时间达到 1 分钟后。

如果连接被外部手段终止（例如被管理工具强行断开），ADO/OLE DB 要到下一次与服务器交互时才会得知连接断开了。在实施强行断开之前应当仔细考虑。

控制连接池的标志是 DBPROPVAL_OS_RESOURCEPOOLING (1)。可以在连接字符串中使用一个连接参数来将此标志关闭。

如果在连接字符串中指定 `OLE DB Services=-2`，连接池将被禁用。下面是示例连接字符串：

```
Provider=SAOLEDB;OLE DB Services=-2;...
```

如果在连接字符串中指定 `OLE DB Services=-4`，连接池和事务征用将被禁用。下面是示例连接字符串：

```
Provider=SAOLEDB;OLE DB Services=-4;...
```

在禁用连接池的情况下，若应用程序频繁地用相同的连接字符串打开和关闭连接，性能将大受影响。

Microsoft 链接服务器

可创建一个能使用 SAP Sybase IQ OLE DB 提供程序来获得对 SAP Sybase IQ 数据库的访问权限的 Microsoft 链接服务器。可使用 Microsoft 的表引用语法（由四部分组

成）或 Microsoft 的 OPENQUERY SQL 函数来发出 SQL 查询。下面就是一个由四部分组成的语法的示例。

```
SELECT * FROM SADATABASE.demo.GROUPO.Customers
```

在本例中，SADATABASE 是链接服务器的名称，demo 是目录或数据库名，GROUPO 是 SAP Sybase IQ 数据库中的表所有者，而 Customers 是 SAP Sybase IQ 数据库中的表名。

另一种形式是使用 Microsoft 的 OPENQUERY 函数。

```
SELECT * FROM OPENQUERY( SADATABASE, 'SELECT * FROM Customers' )
```

在 OPENQUERY 语法中，第二个 SELECT 语句 ('SELECT * FROM Customers') 被传递到 SAP Sybase IQ 服务器中执行。

对于复杂查询，由于整个查询是在 SAP Sybase IQ 服务器上进行计算的，所以可能更适合使用 OPENQUERY。通过由四部分组成的语法，SQL Server 可能会先检索该查询所引用的所有表的内容，然后才能对查询进行计算（如包含 WHERE、JOIN 的查询以及嵌套查询等）。如果查询中涉及到非常大的表，则使用由四部分组成的语法时，处理时间可能会很长。在以下由四部分组成的查询示例中，SQL Server 通过 OLE DB 提供程序向 SAP Sybase IQ 数据库服务器传递一条对整个表的简单 SELECT 子句（无 WHERE 子句），然后再计算 WHERE 条件本身。

```
SELECT ID, Surname, GivenName FROM [SADATABASE].[demo].[GROUPO].
[Customers]
WHERE Surname = 'Elkins'
```

将返回结果集中的所有行，然后由 SQL Server 将此结果集减至一行，而不是向 SQL Server 返回结果集中的一行。以下示例产生的结果相同，但仅向 SQL Server 返回一行。

```
SELECT * FROM OPENQUERY( SADATABASE,
    'SELECT ID, Surname, GivenName FROM [GROUPO].[Customers]
    WHERE Surname = ''Elkins''' )
```

可以用 Microsoft SQL Server 交互应用程序或 SQL Server 脚本来设置使用 SAP Sybase IQ OLE DB 提供程序的链接服务器。

注意： 若使用 Windows Vista 或更高版本的 Windows，则设置链接服务器之前需要注意几个事项。SQL Server 作为系统上的服务运行。根据服务在 Windows Vista 或更高版本上的设置方式，服务可能无法使用共享内存连接，无法启动服务器，以及无法访问用户数据源定义。例如，作为**“网络服务”**登录的服务无法启动服务器，无法通过共享内存连接，也无法访问用户数据源。这些情况下，SAP Sybase IQ 服务器必须提前启动并且必须使用 TCPIP 通信协议。此外，如果要使用数据源，则该数据源必须是系统数据源。

使用交互应用程序设置链接服务器

使用 Microsoft SQL Server 交互应用程序可创建 Microsoft 链接服务器，该服务器使用 SAP Sybase IQ OLE DB 提供程序来获取 SAP Sybase IQ 数据库的访问权限。

前提条件

SQL Server 2000 或更高版本。

过程

1. 对于 Microsoft SQL Server 2005/2008，启动 SQL Server Management Studio。对于其它版本的 SQL Server，此应用程序的名称和链接服务器的设置步骤可能会有所不同。

   在“**Object Explorer**”窗格中，展开“**Server Objects**”»“**Linked Servers**”。右击“**Linked Servers**”并单击“**New Linked Server**”。
2. 填充“**General**”页。

   “**General**”页上的“**Linked Server**”字段应包含“**Linked Server**”的名称（如上例中的 SADATABASE）。

   应选择“**Other Data Source**”选项，并从“**Provider**”列表中选择“**SQL Anywhere OLE DB Provider 16**”。

   “**Product Name**”字段可以填入任意名称（例如 SAP Sybase IQ 或您的应用程序名称）。

   “**Data Source**”字段可包含 ODBC 数据源名 (DSN)。这是一个便捷选项，不一定非得包含数据源名。如果使用系统 DSN，则必须是适用于 32 位 SQL Server 版本的 32 位 DSN，或者是适用于 64 位 SQL Server 版本的 64 位 DSN。

   ```
   Data Source: SAP Sybase IQ 16 Demo
   ```

   “**Provider String**”字段可以包含其它连接参数，例如 UserID (UID)、ServerName (Server) 和 DatabaseFile (DBF)。

   ```
   Provider string: Server=myserver;DBF=sample.db
   ```

   “**Location**”字段可以包含与 SAP Sybase IQ Host 连接参数等效的内容（例如，localhost:4444 或 10.25.99.253:2638）。

   ```
   Location: AppServer-pc:2639
   ```

   “**Initial Catalog**”字段可以包含要连接的数据库的名称（例如，demo）。此数据库必须已经处于打开状态。

   ```
   Initial Catalog: demo
   ```

   上述最后四个字段与“**Security**”页面中的用户 ID 和口令的组合必须包含足够多的信息才能成功连接到数据库服务器。

3. 不要在“**Provider String**”字段中指定数据库用户 ID 和口令作为连接参数（因为它是以纯文本显示的），而应该在“**Security**”页面中填写。

   在 SQL Server 2005/2008 中，单击“**Be made using this security context**”选项并填写“**Remote login**”和“**With password**”字段（口令以星号显示）。
4. 转到“**Server Options**”页面。

   启用“**RPC**”和“**RPC Out**”选项。

   完成该操作的方法因 Microsoft SQL Server 的版本而异。在 SQL Server 2000 中，必须为这两个选项选中两个复选框。在 SQL Server 2005/2008 中，这些选项具有 True/False 设置。确保将它们设置为 True。要在 SAP Sybase IQ 数据库中执行存储过程/函数调用，并成功地传递参数（传入和传出），则必须设置“**Remote Procedure Call**”（“**RPC**”）选项。
5. 选择“**Allow Inprocess**”提供程序选项。

   完成该操作的方法因 Microsoft SQL Server 的版本而异。在 SQL Server 2000 中有一个“**Provider Options**”按钮，可将您引至能选择此选项的页面。对于 SQL Server 2005/2008，在“**Linked Servers**”»“**Providers**”下右击 SAOLEDB.16 提供程序并单击“**Properties**”。确保选中“**Allow Inprocess**”复选框。如果未选择“**Inprocess**”选项，则查询将失败。
6. 可忽略其它提供程序选项。这些选项中有些与 SQL Server 的向后兼容性有关，并且不会影响 SQL Server 与 SAP Sybase IQ OLE DB 提供程序的交互方式。例如，“**Nested queries**”和“**Supports LIKE operator**”。选择其它选项可能会产生语法错误或导致性能降低。

Microsoft 链接服务器已配置完成。

使用脚本设置链接服务器

可使用 SQL Server 脚本设置链接服务器定义。

前提条件

SQL Server 2005 或更高版本。

过程

在 SQL Server 上运行以下脚本前，使用下列步骤对该脚本进行相应更改。

```
USE [master]
GO
EXEC master.dbo.sp_addlinkedserver @server=N'SADATABASE',
     @srvproduct=N'SAP Sybase IQ', @provider=N'SAOLEDB.16',
     @datasrc=datasrc=N'Sybase IQ Demo',
     @provstr=N'host=localhost:4444;server=myserver;dbn=demo'
GO
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
     @optname=N'rpc', @optvalue=N'true'
GO
```

```
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
     @optname=N'rpc out', @optvalue=N'true'
GO
-- Set remote login
EXEC master.dbo.sp_addlinkedsrvlogin @rmtsrvname = N'SADATABASE',
     @locallogin = NULL , @useself = N'False',
     @rmtuser = N'DBA', @rmtpassword = N'sql'
GO
-- Set global provider "allow in process" flag
EXEC master.dbo.sp_MSset_oledb_prop N'SAOLEDB.16',
N'AllowInProcess', 1
```

1. 选择新的链接服务器名称（本例中使用 SADATABASE）。
2. 选择可选的数据源名称（本例中使用 SAP Sybase IQ 16 Demo）。
3. 选择可选的提供程序字符串（本例中使用 `N'host=localhost:4444;server=myserver;dbn=demo'`）。
4. 选择远程用户 ID 和口令（本例中使用 N'DBA' 和 N'sql'）。

可在 Microsoft SQL Server 下运行修改后的脚本，以创建新的链接服务器。

支持的 OLE DB 接口

OLE DB API 由一组接口组成。下表介绍 SQL Anywhere OLE DB 驱动程序中对每个接口的支持。

接口	作用	限制
IAccessor	定义客户端内存和数据存储值之间的捆绑。	不支持 DBACCESSOR_PASSBYREF。不支持 DBACCESSOR_OPTIMIZED。
IAlterIndex IAlterTable	变更表、索引和列。	不支持。
IChapteredRowset	分段的行集允许以单独的段访问行集中的行。	不支持。SAP Sybase IQ 不支持分段的行集。
IColumnsInfo	获得有关行集的列的简单信息。	支持。
IColumnsRowset	获得有关行集内可选元数据列的信息，并获得列元数据的行集。	支持。
ICommand	执行 SQL 语句。	不支持调用。ICommandProperties：带有 DBPROPSET_PROPERTIESINERROR 的 GetProperties，用于查找尚未设置的属性。

接口	作用	限制
ICommandPersist	保持命令对象（而非任何活动行集）的状态。随后可使用 PROCEDURES 或 VIEWS 行集枚举这些持久性命令对象。	支持。
ICommandPrepare	准备命令。	支持。
ICommandProperties	为由命令创建的行集设置“行集”属性。最常用于指定行集应当支持的接口。	支持。
ICommandText	为 ICommand 设置 SQL 语句文本。	只支持 DBGUID_DEFAULT SQL 方言。
ICommandWithParameters	为命令设置或获取参数信息。	不支持以标量值的矢量形式存储的参数。 不支持 BLOB 参数。
IConvertType		支持。
IDBAsynchNotify IDBAsynchStatus	异步处理。 在数据源初始化、填充行集等的异步处理时，向客户端通知事件。	不支持。
IDBCreateCommand	从会话创建命令。	支持。
IDBCreateSession	从数据源对象创建会话。	支持。
IDBDataSourceAdmin	创建/破坏/修改数据源对象：由客户端使用的 COM 对象。该接口不用于管理数据存储区（数据库）。	不支持。
IDBInfo	查找该提供程序所特有的关键字的信息（即，查找非标准的 SQL 关键字）。 还查找有关文字、文本匹配查询中使用的特殊字符、以及其它文字信息。	支持。
IDBInitialize	初始化数据源对象和枚举器。	支持。

接口	作用	限制
IDBProperties	管理数据源对象或枚举器的属性。	支持。
IDBSchemaRowset	以标准格式（行集）获取有关系统表的信息。	支持。
IErrorInfo IErrorLookup IErrorRecords	支持 ActiveX 错误对象。	支持。
IGetDataSource	将接口指针返回会话的数据源对象。	支持。
IIndexDefinition	在数据存储区创建或删除索引。	不支持。
IMultipleResults	从命令检索多个结果（行集或行计数）。	支持。
IOpenRowset	按照数据库表的名称访问数据库表的非 SQL 方式。	支持。 支持按照表名打开表，不支持按照 GUID 打开表。
IParentRowset	访问分段/分层行集。	不支持。
IRowset	访问行集。	支持。
IRowsetChange	允许更改行集数据，并将所做更改反映回数据存储区。 未实现用于 BLOB 的 InsertRow/SetData。	支持。
IRowsetChapterMember	访问分段/分层行集。	不支持。
IRowsetCurrentIndex	动态更改行集的索引。	不支持。
IRowsetFind	在行集中查找与指定值匹配的行。	不支持。
IRowsetIdentity	比较行的句柄。	不支持。
IRowsetIndex	访问数据库索引。	不支持。

接口	作用	限制
IRowsetInfo	查找有关行集属性的信息或者查找创建了行集的对象。	支持。
IRowsetLocate	使用书签定位到行集的行上。	支持。
IRowsetNotify	为行集事件提供 COM 回调接口。	支持。
IRowsetRefresh	获取事务可看到的数据的最新值。	不支持。
IRowsetResynch	旧的 OLE DB 1.x 接口，被 IRowsetRefresh 替代。	不支持。
IRowsetScroll	滚动浏览行集以抓取行数据。	不支持。
IRowsetUpdate	延迟对行集数据的更改，直到 Update 被调用。	支持。
IRowsetView	使用现有行集上的视图。	不支持。
ISequentialStream	检索 BLOB 列。	只支持读取。 不支持 对该接口执行 SetData。
ISessionProperties	获得会话属性信息。	支持。
ISourcesRowset	获得数据源对象和枚举器的行集。	支持。
ISQLErrorInfo ISupportErrorInfo	支持 ActiveX 错误对象。	支持。
ITableDefinition ITableDefinitionWith-Constraints	用约束来创建、删除和变更表。	支持。
ITransaction	提交或中止事务。	并非所有的标志都受支持。
ITransactionJoin	支持分布式事务。	并非所有的标志都受支持。
ITransactionLocal	处理会话事务。 并非所有的标志都受支持。	支持。

接口	作用	限制
ITransactionOptions	获得或设置事务的选项。	支持。
IViewChapter	使用现有行集上的视图，专用于对行应用后处理过滤/排序。	不支持。
IViewFilter	将行集的内容限制为与条件集相匹配的行。	不支持。
IViewRowset	在打开行集时，将行集的内容限制为与条件集相匹配的行。	不支持。
IViewSort	对视图应用排序顺序。	不支持。

注册 OLE DB 提供程序

使用 SAP Sybase IQ 安装程序安装 SAOLEDB 提供程序时，提供程序会自行注册。注册过程包括在注册表的 COM 部分创建注册表条目，以便 ADO 可以在 SAOLEDB 提供程序被调用时找到 DLL。如果更改 DLL 的位置，则必须重新注册。

示例

SAP Sybase IQ OLE DB 提供程序从其安装目录运行时，以下命令将对该提供程序进行注册：

```
regsvr32 dboledb16.dll
regsvr32 dboledba16.dll
```

ODBC CLI

ODBC（开放式数据库连接，Open Database Connectivity）是 Microsoft Corporation 开发的一个标准调用层接口（Call Level Interface，简称 CLI)。它基于“SQL 访问组 CLI”规范。可以对任何提供 ODBC 驱动程序的数据源运行 ODBC 应用程序。如果您希望应用程序能够向具有 ODBC 驱动程序的其它数据源移植，将 ODBC 作为编程接口是很好的选择。

对 ODBC 的支持

SAP Sybase IQ 提供对 ODBC 3.5 的支持，这是作为 Microsoft Data Access Kit 2.7 的一部分提供的。

ODBC 支持的级别

ODBC 功能是根据符合的级别进行组织的。这些功能为**核心**、**级别 1** 或**级别 2**，级别 2 是 ODBC 支持的最完整级别。这些功能在 Microsoft 的 ODBC 程序员参考中列出，网址为 http://msdn.microsoft.com/zh-cn/library/ms714177.aspx。

SAP Sybase IQ 支持的功能

SAP Sybase IQ 对 ODBC 3.5 规范的支持如下所示：

- **核心支持** – SAP Sybase IQ 支持所有 [核心] 级别功能。
- **级别 1 支持** – 除了 ODBC 函数的异步执行以外，SAP Sybase IQ 支持所有 [级别 1] 功能。

  SAP Sybase IQ 支持多个线程共享一个连接。SAP Sybase IQ 序列化来自不同线程的请求。
- **级别 2 支持** – SAP Sybase IQ 支持除了以下几项以外的所有其它 [级别 2] 功能：
  - 表和视图的由三个部分构成的名称。这不适用于 SAP Sybase IQ。
  - 指定的个别语句的 ODBC 函数异步执行。
  - 使登录请求和 SQL 查询超时的能力。

ODBC 应用程序开发

每个调用 ODBC 函数的 C/C++ 源文件都必须包括一个平台特定的 ODBC 头文件。每个特定于平台的头文件都包括主要的 ODBC 头文件 `odbc.h`，该头文件定义了编写 ODBC 程序所需的所有函数、数据类型和常量定义。

执行下列任务以将 ODBC 头文件包括在 C/C++ 源文件中：

1. 向源文件添加 include 行，该行引用相应的平台特定的头文件。要使用的行如下所示：

操作系统	Include 行
Windows	#include "ntodbc.h"
Unix	#include "unixodbc.h"
Windows Mobile	#include "ntodbc.h"

2. 将包含头文件的目录添加到您的编译器的包括路径中。
   特定于平台的头文件和 `odbc.h` 都安装在 SAP Sybase IQ 安装目录的 `SDK\Include` 子目录下。
3. 为 Unix 创建 ODBC 应用程序时，可能需要为 32 位应用程序定义宏 "UNIX" 或为 64 位应用程序定义宏 "UNIX64"，以便获得正确的数据对齐方式和大小。如果您使用的是以下受支持的编译器之一，则不需要此步骤：

   任何受支持平台上的 GNU C/C++ 编译器
   适用于 Linux 的 Intel C/C++ 编译器 (icc)
   适用于 Linux 或 Solaris 的 SunPro C/C++ 编译器
   适用于 AIX 的 VisualAge C/C++ 编译器
   适用于 HP-UX 的 C/C++ 编译器 (cc/aCC)

源代码编写完成后，可以随时对应用程序进行编译和链接。以下各节介绍如何创建可执行应用程序。

Windows 上的 ODBC 应用程序

链接应用程序时，必须链接到适当的导入库文件，才能访问 ODBC 函数。

导入库为 ODBC 驱动程序管理器 `odbc32.dll` 定义入口点。驱动程序管理器进而装载 SAP Sybase IQ ODBC 驱动程序 `dbodbc16.dll`。

通常，导入库存储在 Microsoft 平台 SDK 的 `Lib` 目录结构下：

操作系统	导入库
Windows（32 位）	`\Lib\X86\odbc32.lib`
Windows（64 位）	`\Lib\X86\odbc32.lib`

示例

以下命令说明如何将包含特定于平台的导入库的目录添加到 LIB 环境变量的库目录列表中。

```
set LIB=%LIB%;c:\mssdk\v7.0\lib
```

以下命令说明如何使用 Microsoft 编译和链接工具对存储在 `odbc.c` 中的应用程序进行编译和链接：

```
cl odbc.c /I"%IQDIR16%\SDK\Lib\X86\Include" odbc32.lib
```

Unix 上的 ODBC 应用程序

用于 Unix 的 ODBC 驱动程序管理器随 SAP Sybase IQ 提供，并且有第三方驱动程序管理器可供使用。本节说明介绍创建不使用 ODBC 驱动程序管理器的 ODBC 应用程序。

ODBC 驱动程序

ODBC 驱动程序是一个共享对象或者共享库。针对单线程应用程序和多线程应用程序提供了几种 SAP Sybase IQ ODBC 驱动程序版本。此外，还提供了通用的 SAP Sybase IQ ODBC 驱动程序，用于检测到正在使用的线程模型，以及对适当单线程库或多线程库的直接调用。

ODBC 驱动程序就是下面的文件：

操作系统	线程模型	ODBC 驱动程序
（除 HP-UX 以外的所有 Unix）	通用	`libdbodbc16.so (libdbodbc16.so.1)`
（除 HP-UX 以外的所有 Unix）	单线程	`libdbodbc16_n.so (libdbodbc16_n.so.1)`
（除 HP-UX 以外的所有 Unix）	多线程	`libdbodbc16_r.so (libdbodbc16_r.so.1)`
HP-UX	通用	`libdbodbc16.sl (libdbodbc16.sl.1)`
HP-UX	单线程	`libdbodbc16_n.sl (libdbodbc16_n.sl.1)`
HP-UX	多线程	`libdbodbc16_r.sl (libdbodbc16_r.sl.1)`

库是作为符号链接进行安装的，这些链接指向带有版本号（显示在括号中）的共享库。

在 Unix 上链接 ODBC 应用程序时，请将应用程序链接到通用 ODBC 驱动程序 `libdbodbc16`。部署应用程序时，请确保用户的库路径中有适当的（或所有）ODBC 驱动程序版本（非线程或线程）。

数据源信息

如果 SAP Sybase IQ 未检测到 ODBC 驱动程序管理器，它会将系统信息文件用于数据源信息。

unixODBC 驱动程序管理器

unixODBC 在版本 2.2.14 之前发行的版本错误地实现了由 Microsoft 所定义的 64 位 ODBC 规范中的某些方面。将 unixODBC 驱动程序管理器与 SAP Sybase IQ 64 位 ODBC 驱动程序一起使用时，这些差异会导致问题。

要避免这些问题，您应了解这些差异。差异之一是 SQLLEN 和 SQLULEN 的定义。在 Microsoft 64 位 ODBC 规范中，这些定义是 64 位类型的，并且是 SAP Sybase IQ 64 位 ODBC 驱动程序应使用的 64 位类型。而 unixODBC 的某些实现方式将这两种类型定义为 32 位，在连接到 SAP Sybase IQ 64 位 ODBC 驱动程序时，这会导致问题。

要避免在 64 位平台上出现问题，您必须做三件事。

1. 您应包括 SAP Sybase IQ ODBC 头文件 `unixodbc.h`，而不是包括 unixODBC 标头（例如 `sql.h` 和 `sqlext.h`）。这将保证您有正确的 SQLLEN 和 SQLULEN 定义。在 unixODBC 2.2.14 或更高版本中，通过头文件更正了该问题。
2. 您必须确保已使用了所有参数的正确类型。正确头文件的使用以及 C/C++ 编译器强大的类型检查功能会在这方面有所帮助。还必须确保您已为所有由 SAP Sybase IQ 驱动程序通过指针间接设置的变量使用了正确的类型。
3. 不要使用 2.2.14 发行版以前的 unixODBC 驱动程序管理器版本，而是直接链接到 SAP Sybase IQ ODBC 驱动程序。例如，确保 libodbc 共享对象链接到 SAP Sybase IQ 驱动程序。

   ```
   libodbc.so.1 -> libdbodbc16_r.so.1
   ```

   或者，也可以在平台上使用 SAP Sybase IQ 驱动程序管理器（如果该平台具有该程序）。

适用于 Unix 的 UTF-32 ODBC 驱动管理器

将 SQLWCHAR 定义为 32 位 (UTF-32) 的 ODBC 驱动程序管理器版本不能同支持宽调用的 SAP Sybase IQ ODBC 驱动程序一起使用，因为该驱动程序是为 16 位 SQLWCHAR 构建的。针对这些情况，特地提供了纯 ANSI 版本的 SAP Sybase IQ ODBC 驱动程序。此版本的 ODBC 驱动程序不支持宽调用接口（如 SQLConnectW）。

此驱动程序的共享对象名为 `libdbodbcansi16_r`。只提供一个驱动程序的线程变体。某些框架，如 Real Basic，不能使用 dyliband，必需使用打包形式。

常规 ODBC 驱动程序将 SQLWCHAR 字符串视作 UTF-16 字符串。该驱动程序不能和某些 ODBC 驱动程序管理器一起使用（如 iODBC），因为这些驱动程序会将 SQLWCHAR 字符串视为 UTF-32 字符串。处理启用了 Unicode 的驱动程序时，这些驱动程序管理器将来自应用程序的窄调用转换成到驱动程序的宽调用。纯 ANSI 驱动程序避开了这种行为，只要应用程序不作出任何宽调用，驱动程序就可以与这些驱动程序管理器一同使用。通过 iODBC 进行宽调用或其它任何有类似语义的驱动程序管理器，仍然是不受支持的。

ODBC 示例

SAP Sybase IQ 中包含几个 ODBC 示例。可在 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C 目录 (Windows) 和 $SYBASE/IQ-16_0/samples/sqlanywhere/c 目录 (UNIX) 中找到这些示例。

目录中这些以 ODBC 开头的示例说明单独和简单的 ODBC 任务，例如连接到数据库和执行语句。odbc.c 文件中提供了一个完整的 ODBC 示例程序。此程序执行的操作与同一目录中嵌入式 SQL 动态游标示例程序执行的操作相同。

构建适用于 Windows 的 ODBC 示例程序

通过构建 ODBC 示例程序，可以运行该程序，并了解该程序如何执行 ODBC 任务，例如连接到数据库和执行语句。

前提条件

对于 x64 平台版本，您可能需要设置正确的编译和链接环境。以下示例构建了一个适用于 x64 平台的示例程序。

```
set mssdk=c:\mssdk\v7.0
build64
```

过程

位于 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C 目录中的批处理文件可用于对所有示例应用程序进行编译和链接。

1. 打开命令提示符，将目录更改为 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C 目录。
2. 运行 build.bat 或 build64.bat 批处理文件。

示例 ODBC 程序已构建完成。

构建适用于 Unix 的 ODBC 示例程序

通过构建 ODBC 示例程序，可以运行该程序，并了解该程序如何执行 ODBC 任务，例如连接到数据库和执行语句。

前提条件

执行此任务没有前提条件。

过程

位于 $SYBASE/IQ-16_0/samples/sqlanywhere/c 目录中的 shell 脚本可用于对所有示例应用程序进行编译和链接。

1. 打开命令 shell，将目录更改为 `$SYBASE/IQ-16_0/samples/sqlanywhere/c` 目录。
2. 运行 `build.sh` shell 脚本。

示例 ODBC 程序已构建完成。

ODBC 示例程序

可通过在相应的平台中运行以下文件来装载 ODBC 示例程序。

- 对于 32 位 Windows，运行 `%ALLUSERSPROFILE%\SybaseIQ\samples\sqlanywhere\C\odbcwin.exe`。
- 对于 64 位 Windows，运行 `%ALLUSERSPROFILE%\SybaseIQ\samples\sqlanywhere\C\odbcx64.exe`。
- 对于 Unix，运行 `$SYBASE/IQ-16_0/samples/sqlanywhere/C/odbc`。

文件运行后，选择示例数据库中的一个表。例如，可以输入 `Customers` 或 `Employees`。

ODBC 句柄

ODBC 应用程序使用一小组**句柄**来定义基本功能，例如数据库连接和 SQL 语句。句柄是一个 32 位的值。

下面的句柄基本上用于所有 ODBC 应用程序：

- **环境** – 环境句柄提供一个在其中访问数据的全局上下文。每个 ODBC 应用程序必须在启动时分配一个（只一个）环境句柄，在结束时必须将其释放。

  下面的代码说明如何分配环境句柄：

  ```
  SQLRETURN rc;
  SQLHENV env;
  rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
  ```
- **连接** – 连接是由 ODBC 驱动程序和数据源指定的。应用程序可以具有几个与它的环境相关联的连接。分配连接句柄并不会建立连接；必须首先分配连接句柄，然后才能在建立连接时使用它。

  下面的代码说明如何分配连接句柄：

  ```
  SQLRETURN rc;
  SQLHDBC  dbc;
  rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
  ```
- **语句** – 语句句柄提供对 SQL 语句以及与之相关联的任何信息（如结果集和参数）的访问。每个连接可以有多个语句。在游标操作（读取数据）和单个语句执行（例如 INSERT、UPDATE 和 DELETE）中都使用语句。

  下面的代码说明如何分配语句句柄：

```
SQLRETURN rc;
SQLHSTMT stmt;
rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

如何分配 ODBC 句柄

ODBC 程序所需的句柄类型现列示如下：

项	句柄类型
环境	SQLHENV
连接	SQLHDBC
语句	SQLHSTMT
描述符	SQLHDESC

要使用 ODBC 句柄，请执行以下任务：

1. 调用 SQLAllocHandle 函数。
2. 在随后的函数调用中使用该句柄。
3. 使用 SQLFreeHandle 释放对象。

SQLAllocHandle 采用以下参数：

- 要分配的项所属类型的标识符
- 父项的句柄
- 要分配的句柄的位置指针
  有关信息，请参见 Microsoft 的 ODBC API Reference 中的 SQLAllocHandle，网址为 http://msdn.microsoft.com/zh-cn/library/ms712455.aspx。

SQLFreeHandle 采用以下参数：

- 要释放的项所属类型的标识符
- 要释放的项的句柄
  有关信息，请参见 Microsoft 的 ODBC API Reference 中的 SQLFreeHandle，网址为 http://msdn.microsoft.com/zh-cn/library/ms710123.aspx。

示例

下面的代码段将分配和释放一个环境句柄：

```
SQLRETURN rc;
SQLHENV env;
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
if( rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO )
{
    .
    .
    .
}
SQLFreeHandle( SQL_HANDLE_ENV, env );
```

ODBC 示例

有关连接 SAP Sybase IQ 示例数据库并立即断开的简单 ODBC 程序，可以访问 %IQDIRSAMP16%\SQLAnywhere\ODBCConnect\odbcconnect.cpp。此示例显示了设置与数据库服务器连接的环境所需的步骤，以及断开服务器和释放资源所需的步骤。

ODBC 连接函数

ODBC 提供了一组连接函数。使用哪一个连接函数，取决于您希望如何部署和使用应用程序：

- **SQLConnect** – 最简单的连接函数。

  SQLConnect 接收数据源名，以及用户 ID 和口令（可选）。如果您将数据源名硬编码到应用程序中，则可能想要使用 SQLConnect。

  有关更多信息，请参见 Microsoft ODBC API 参考 中的 SQLConnect (http://msdn.microsoft.com/en-us/library/ms711810.aspx)。
- **SQLDriverConnect** – 使用连接字符串连接数据源。

  SQLDriverConnect 允许应用程序使用数据源外部的特定于 SAP Sybase IQ 的连接信息。另外，您可以使用 SQLDriverConnect 来请求 Sybase IQ ODBC 驱动程序提示用户提供连接信息。

  SQLDriverConnect 还可用来在不指定数据源的情况下进行连接。Sybase IQ ODBC 驱动程序名是指定的。以下示例连接到一台已在运行的数据库服务器上。

  ```
  SQLSMALLINT cso;
  SQLCHAR     scso[2048];

  SQLDriverConnect( hdbc, NULL,
     "Driver=Sybase IQ;UID=<user_id>;PWD=<password>", SQL_NTS,
     scso, sizeof(scso)-1,
     &cso, SQL_DRIVER_NOPROMPT );
  ```

  有关更多信息，请参见 Microsoft ODBC API 参考中的 SQLDriverConnect (http://msdn.microsoft.com/en-us/library/ms715433.aspx)。
- **SQLBrowseConnect** – 使用连接字符串（如 SQLDriverConnect）连接到数据源。

  SQLBrowseConnect 允许您的应用程序创建一些自己的、具有以下功能的窗口：提示用户提供连接信息，以及浏览由特定驱动程序（这里是 Sybase IQ ODBC 驱动程序）使用的数据源。

  有关更多信息，请参见 Microsoft ODBC API 参考中的 SQLBrowseConnect (http://msdn.microsoft.com/en-us/library/ms714565.aspx)。

建立 ODBC 连接

在应用程序中建立 ODBC 连接以执行任何数据库操作。

前提条件

执行此任务没有前提条件。

过程

可在 `%ALLUSERSPROFILE%\SybaseIQ\samples\ODBCConnect\odbcconnect.cpp` 下找到完整示例。

1. 分配 ODBC 环境。

   例如：

   ```
   SQLRETURN rc;
   SQLHENV   env;
   rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
   ```

2. 声明 ODBC 版本。

   通过声明该应用程序遵循 ODBC 版本 3，SQLSTATE 值和某些其它与版本相关的功能都将被设置为适当的行为。例如：

   ```
   rc = SQLSetEnvAttr( env, SQL_ATTR_ODBC_VERSION,
   (void*)SQL_OV_ODBC3, 0 );
   ```

3. 分配 ODBC 连接项。

   例如：

   ```
   rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
   ```

4. 设置任何必须在连接前设置的连接属性。

   有些连接特性必须在建立连接前或建立连接后进行设置，而其它连接特性则既可在之前设置也可在之后设置。SQL_AUTOCOMMIT 特性是一个既可在之前设置也可在之后设置的特性：

   ```
   rc = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0 );
   ```

   缺省情况下，ODBC 在自动提交模式下工作。通过将 SQL_AUTOCOMMIT 设置为 false 可关闭此模式。

5. 如果需要，可以汇编这些数据源或连接字符串。

   根据您的应用程序的情况，您可以对数据源或连接字符串进行硬编码，也可以将它们存储在外部以获得更大的灵活性。

6. 调用 ODBC 连接函数。

   例如：

   ```
   if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
   {
   ```

```
printf( "dbc allocated\n" );
rc = SQLConnect( dbc,
   (SQLCHAR *) "Sybase IQ Demo", SQL_NTS,
   (SQLCHAR *) "<user_id>", SQL_NTS,
   (SQLCHAR *) "<password>", SQL_NTS );
if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
{
    // Successfully connected.
```

每个传递到 ODBC 的字符串都有相应的长度。如果长度未知，则可传递 SQL_NTS，表示它是一个结尾标记为空值字符 (\0) 的*以 NULL 终止的字符串*。

应用程序在构建和运行时会建立 ODBC 连接。

ODBC 更改的服务器选项

连接到 SAP Sybase IQ 数据库时，SAP Sybase IQ ODBC 驱动程序会设置一些临时服务器选项。以下选项设置如下。

- **date_format** – yyyy-mm-dd
- **date_order** – ymd
- **isolation_level** – 基于 SQLSetConnectAttr 的 SQL_ATTR_TXN_ISOLATION/ SA_SQL_ATTR_TXN_ISOLATION 属性设置。提供以下选项。

```
SQL_TXN_READ_UNCOMMITTED
SQL_TXN_READ_COMMITTED
SQL_TXN_REPEATABLE_READ
SQL_TXN_SERIALIZABLE
SA_SQL_TXN_SNAPSHOT
SA_SQL_TXN_STATEMENT_SNAPSHOT
SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT
```

- **time_format** – hh:nn:ss
- **timestamp_format** – yyyy-mm-dd hh:nn:ss.ssssss
- **timestamp_with_time_zone_format** – yyyy-mm-dd hh:nn:ss.ssssss +hh:nn

要恢复缺省选项设置，请执行 SET 语句.这是一个重置 timestamp_format 选项的语句的示例。

```
set temporary option timestamp_format =
```

SQLSetConnectAttr 扩展连接属性

SAP Sybase IQ ODBC 驱动程序支持某些扩展的连接属性。

- **SA_REGISTER_MESSAGE_CALLBACK** – 可使用 SQL MESSAGE 语句将消息从数据库服务器发送到客户端应用程序。长时间运行的数据库服务器语句也可以生成消息。

  可创建消息处理程序例程来拦截这些消息。消息处理程序的回调原型如下:

```
void SQL_CALLBACK message_handler(
SQLHDBC sqlany_dbc,
unsigned char  msg_type,
long code,
unsigned short  length,
char * message
);
```

msg_type 的以下可能值在 sqldef.h 中定义。

- **MESSAGE_TYPE_INFO** – 消息类型为 INFO。
- **MESSAGE_TYPE_WARNING** – 消息类型为 WARNING。
- **MESSAGE_TYPE_ACTION** – 消息类型为 ACTION。
- **MESSAGE_TYPE_STATUS** – 消息类型为 STATUS。
- **MESSAGE_TYPE_PROGRESS** – 消息类型为 PROGRESS。此类消息是由长时间运行的数据库服务器语句生成的，如 BACKUP DATABASE 和 LOAD TABLE。

与消息关联的 SQLCODE 可以在 *code* 中提供。如果不可用，则 *code* 参数值为 0。

消息的长度包含在 *length* 中。

消息的指针包含在 *message* 中。消息字符串不是以空值终止的。必须设计您的应用程序以处理这一问题。以下是一个示例。

```
memcpy( mybuff, msg, len );
mybuff[ len ] = '\0';
```

要在 ODBC 中注册消息处理程序，请按如下方式调用 SQLSetConnectAttr 函数：

```
rc = SQLSetConnectAttr(
   hdbc,
   SA_REGISTER_MESSAGE_CALLBACK,
   (SQLPOINTER) &message_handler, SQL_IS_POINTER );
```

要在 ODBC 中注销消息处理程序，请按如下方式调用 SQLSetConnectAttr 函数：

```
rc = SQLSetConnectAttr(
   hdbc,
   SA_REGISTER_MESSAGE_CALLBACK,
   NULL, SQL_IS_POINTER );
```

- **SA_GET_MESSAGE_CALLBACK_PARM** – 要检索将传递给消息处理程序回调例程的 SQLHDBC 连接句柄的值，请将 SQLGetConnectAttr 与 SA_GET_MESSAGE_CALLBACK_PARM 参数配合使用。

```
SQLHDBC callback_hdbc = NULL;
rc = SQLGetConnectAttr(
   hdbc,
   SA_GET_MESSAGE_CALLBACK_PARM,
   (SQLPOINTER) &callback_hdbc, 0, 0 );
```

返回值将与传递给消息处理程序回调例程的参数值相同。

- **SA_REGISTER_VALIDATE_FILE_TRANSFER_CALLBACK** – 用于注册文件传输校验回调函数。在允许进行任何传输前，ODBC 驱动程序会调用校验回调函

数（如果存在）。如果在执行间接语句（从存储过程内部）期间，请求进行客户端数据传输，则除非客户端应用程序注册了校验回调函数，否则 ODBC 驱动程序将不允许进行传输。下面更详尽地介绍了进行校验调用的条件。

回调原型如下：

```
int SQL_CALLBACK file_transfer_callback(
void * sqlca,
char * file_name,
int is_write
);
```

file_name 参数是要读取或写入的文件的名称。如果请求读取（从客户端传输到服务器），则 *is_write* 参数为 0，如果请求写入，则该参数为非零值。如果不允许进行文件传输，则回调函数应返回 0，否则返回非零值。

为确保数据安全，服务器会跟踪请求文件传输的语句的源。服务器会确定语句是否是从客户端应用程序直接接收的。从客户端启动数据传输时，服务器会将语句源的相关信息发送到客户端软件。对于 ODBC 驱动程序而言，仅当是由于执行客户端应用程序直接发送的语句而请求数据传输时，它才会允许无条件传输数据。否则，应用程序必须注册上文所述的校验回调函数，如果未注册该函数，则传输会被拒绝，而且语句失败并出现一个错误。如果客户端语句调用的某个存储过程在数据库中已经存在，则对该存储过程本身的执行不被视为是客户端启动的语句所完成的。但是，如果客户端应用程序显式地创建一个临时存储过程，则服务器会将对该存储过程的执行视为是由客户端启动的。同样，如果客户端应用程序执行一个批处理语句，则该批处理语句的执行被视为是直接由客户端应用程序完成的。

- **SA_SQL_ATTR_TXN_ISOLATION** – 该属性用于设置扩展的事务隔离级别。以下示例设置一个快照隔离级别：

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SA_SQL_ATTR_TXN_ISOLATION,
      SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
```

64 位 ODBC 注意事项

使用 SQLBindCol、SQLBindParameter 或 SQLGetData 这类的 ODBC 函数时，某些参数的类型在函数原型中被设置为 SQLLEN 或 SQLULEN。视您所查看的 Microsoft ODBC API 参考文档的日期而定，您可能会看到被描述为 SQLINTEGER 或 SQLUINTEGER 的相同参数。

SQLLEN 和 SQLULEN 数据项在 64 位 ODBC 应用程序中为 64 位，在 32 位 ODBC 应用程序中为 32 位。SQLINTEGER 和 SQLUINTEGER 数据项在所有平台上均为 32 位。

为说明该问题，从旧版 Microsoft ODBC API 参考中摘选了以下 ODBC 函数原型。

```
SQLRETURN SQLGetData(
     SQLHSTMT     StatementHandle,
```

```
     SQLUSMALLINT ColumnNumber,
     SQLSMALLINT  TargetType,
     SQLPOINTER   TargetValuePtr,
     SQLINTEGER   BufferLength,
     SQLINTEGER  *StrLen_or_IndPtr);
```

将该函数原型与在 Microsoft Visual Studio 版本 8 的 `sql.h` 中找到的实际函数原型相比较。

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

正如您所见，BufferLength 和 StrLen_or_Ind 参数的类型现在被设置为 SQLLEN，而不是 SQLINTEGER。对于 64 位平台，它们是 64 位，而不是 Microsoft 文档中所述的 32 位。

为避免出现跨平台编译问题，SAP Sybase IQ 提供了自己的 ODBC 头文件。对于 Windows 平台，您应包括 `ntodbc.h` 头文件。对于 Unix 平台（如 Linux），您应包括 `unixodbc.h` 头文件。使用这些头文件可确保与目标平台上的相应 SAP Sybase IQ ODBC 驱动程序兼容。

下表列出了一些在 64 位和 32 位平台上具有相同或不同存储大小的常见 ODBC 类型。

ODBC API	64 位平台	32 位平台
SQLINTEGER	32 位	32 位
SQLUINTEGER	32 位	32 位
SQLLEN	64 位	32 位
SQLULEN	64 位	32 位
SQLSETPOSIROW	64 位	16 位
SQL_C_BOOKMARK	64 位	32 位
BOOKMARK	64 位	32 位

如果对数据变量和参数的声明不正确，则您可能会遇到不正确的软件行为。

下表汇总了引入 64 位支持以来发生更改的 ODBC API 函数原型。已对受影响的参数进行注释。Microsoft 文档中的参数名与函数原型中使用的实际参数名不同时，前者将显示在括号中。参数名是指 Microsoft Visual Studio 版本 8 的头文件中使用的参数名。

ODBC API	参数（文档中的参数名）
SQLBindCol	SQLLEN BufferLength SQLLEN *Strlen_or_Ind
SQLBindParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind
SQLBindParameter	SQLULEN cbColDef (ColumnSize) SQLLEN cbValueMax (BufferLength) SQLLEN *pcbValue (Strlen_or_IndPtr)
SQLColAttribute	SQLLEN *NumericAttribute
SQLColAttributes	SQLLEN *pfDesc
SQLDescribeCol	SQLULEN *ColumnSize (ColumnSizePtr)
SQLDescribeParam	SQLULEN *pcbParamDef (ParameterSizePtr)
SQLExtendedFetch	SQLLEN irow (FetchOffset) SQLULEN *pcrow (RowCountPtr)
SQLFetchScroll	SQLLEN FetchOffset
SQLGetData	SQLLEN BufferLength SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)
SQLGetDescRec	SQLLEN *Length (LengthPtr)
SQLParamOptions	SQLULEN crow, SQLULEN *pirow
SQLPutData	SQLLEN Strlen_or_Ind
SQLRowCount	SQLLEN *RowCount (RowCountPtr)
SQLSetConnectOption	SQLULEN Value
SQLSetDescRec	SQLLEN Length SQLLEN *StringLength (StringLengthPtr) SQLLEN *Indicator (IndicatorPtr)
SQLSetParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)

ODBC API	参数（文档中的参数名）
SQLSetPos	SQLSETPOSIROW irow (RowNumber)
SQLSetScrollOptions	SQLLEN crowKeyset
SQLSetStmtOption	SQLULEN Value

某些通过指针传递到 ODBC API 并从其返回的值已经更改，以符合 64 位应用程序的需要。例如，以下的 SQLSetStmtAttr 和 SQLSetDescField 函数值不再是 SQLINTEGER/SQLUINTEGER。此规则同样适用于 SQLGetStmtAttr 和 SQLGetDescField 函数的相应参数。

ODBC API	Value/ValuePtr 变量的类型
SQLSetStmtAttr(SQL_ATTR_FETCH_BOOKMARK_PTR)	SQLLEN * value
SQLSetStmtAttr(SQL_ATTR_KEYSET_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_MAX_LENGTH)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_MAX_ROWS)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_PARAM_BIND_OFFSET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARAMS_PROCESSED_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARAMSET_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROW_ARRAY_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROW_BIND_OFFSET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_ROW_NUMBER)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROWS_FETCHED_PTR)	SQLULEN * value
SQLSetDescField(SQL_DESC_ARRAY_SIZE)	SQLULEN value
SQLSetDescField(SQL_DESC_BIND_OFFSET_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_ROWS_PROCESSED_PTR)	SQLULEN * value
SQLSetDescField(SQL_DESC_DISPLAY_SIZE)	SQLLEN value
SQLSetDescField(SQL_DESC_INDICATOR_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH_PTR)	SQLLEN * value

有关详细信息，请参见 Microsoft 文章 "在 MDAC 2.7 信息: ODBC 64 位 API 更改"，网址为 http://support.microsoft.com/kb/298678。

数据对齐要求

使用 SQLBindCol、SQLBindParameter 或 SQLGetData 时，将为列或参数指定 C 数据类型。在某些平台上，提供给每列的存储（内存）必须正确对齐，才能读取或存储指定类型的值。ODBC 驱动程序检查数据是否正确对齐。当对象没有正确对齐时，ODBC 驱动程序会发布 **“无效的字符串或缓冲区长度”** 消息 (**“SQLSTATE”** HY090 或 S1090)。

下表列出了诸如 Sun Sparc、Itanium-IA64 和基于 ARM 的设备之类的处理器的内存对齐要求。数据值内存地址必须是指示值的倍数。

C 数据类型	需要的对齐方式
SQL_C_CHAR	无
SQL_C_BINARY	无
SQL_C_GUID	无
SQL_C_BIT	无
SQL_C_STINYINT	无
SQL_C_UTINYINT	无
SQL_C_TINYINT	无
SQL_C_NUMERIC	无
SQL_C_DEFAULT	无
SQL_C_SSHORT	2
SQL_C_USHORT	2
SQL_C_SHORT	2
SQL_C_DATE	2
SQL_C_TIME	2
SQL_C_TIMESTAMP	2
SQL_C_TYPE_DATE	2
SQL_C_TYPE_TIME	2
SQL_C_TYPE_TIMESTAMP	2
SQL_C_WCHAR	2（在所有平台上，缓冲区大小必须是 2 的倍数）

C 数据类型	需要的对齐方式
SQL_C_SLONG	4
SQL_C_ULONG	4
SQL_C_LONG	4
SQL_C_FLOAT	4
SQL_C_DOUBLE	8（对于 ARM，采用 4）
SQL_C_SBIGINT	8
SQL_C_UBIGINT	8

x86、x64 和 PowerPC 平台不需要内存对齐。x64 平台包括 Advanced Micro Devices (AMD) AMD64 处理器和 Intel Extended Memory 64 Technology (EM64T) 处理器。

ODBC 应用程序中的结果集

ODBC 应用程序使用游标来操纵和更新结果集。SAP Sybase IQ 为不同种类的游标和游标操作提供广泛的支持。

ODBC 事务隔离级别

可使用 SQLSetConnectAttr 为连接设置事务隔离级别。确定 SAP Sybase IQ 提供的事务隔离级别的特性包括以下几种：

- **SQL_TXN_READ_UNCOMMITTED** – 将隔离级别设置为 0。在设置此特性值后，它会隔离任何从其他用户所做的更改读取的数据，且其他用户所做的更改将不可见。其他用户重新执行读取语句时将会受到影响。它不支持可重复的读取。这是隔离级别的缺省值。
- **SQL_TXN_READ_COMMITTED** – 将隔离级别设置为 1。在设置此特性值后，它不会隔离任何从其他用户所做的更改读取的数据，且其他用户所做的更改将可见。其他用户重新执行读取语句时将会受到影响。它不支持可重复的读取。
- **SQL_TXN_REPEATABLE_READ** – 将隔离级别设置为 2。在设置此特性值后，它会隔离任何从其他用户所做的更改读取的数据，且其他用户所做的更改将不可见。其他用户重新执行读取语句时将会受到影响。它支持可重复的读取。
- **SQL_TXN_SERIALIZABLE** – 将隔离级别设置为 3。在设置此特性值后，它会隔离任何从其他用户所做的更改读取的数据，且其他用户所做的更改将不可见。其他用户重新执行读取语句时将不会受到影响。它支持可重复的读取。
- **SA_SQL_TXN_SNAPSHOT** – 将隔离级别设置为快照。设置此特性后，它将为整个事务提供数据库的单个视图。

- **SA_SQL_TXN_STATEMENT_SNAPSHOT** – 将隔离级别设置为语句快照。设置此属性值后，它所提供的一致性要比快照隔离差，但是当出现由于长时间运行事务而导致临时文件中版本存储所用的空间过大的情况时，它将非常有用。
- **SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT** – 将隔离级别设置为只读语句快照。设置此属性值后，它所提供的一致性要比语句快照隔离差，但可避免出现更新冲突的可能性。因此，它最适用于移植那些最初打算在不同隔离级别下运行的应用程序。

allow_snapshot_isolation 数据库选项必须设置为 On 才能使用快照、语句快照或只读语句快照设置。

有关详细信息，请参见 Microsoft 的 ODBC API Reference 中的 SQLSetConnectAttr，网址为 http://msdn.microsoft.com/zh-cn/library/ms713605.aspx。

示例

下列片段将隔离级别设置为快照：

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SQL_ATTR_TXN_ISOLATION,
      SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
```

ODBC 游标特性

执行语句和操纵结果集的 ODBC 函数使用游标执行它们的任务。应用程序每次执行 SQLExecute 或 SQLExecDirect 函数时，都会隐式打开游标。

如果游标在结果集中只正向移动而不更新结果集，对于这种应用情况，游标行为相对比较简单。缺省情况下，ODBC 应用程序会请求此行为。ODBC 定义一个只读的只进游标，SAP Sybase IQ 针对这种情况提供了已优化性能的游标。

如果游标需要在结果集中向前和向后双向滚动（例如在很多图形用户界面应用程序中），对于这种应用情况，游标行为将会更加复杂。当应用程序返回到一个由某种别的应用程序更新的行时，它会怎样？ODBC 定义了多种**可滚动游标**，从而使您可以内置适合您的应用程序的行为。SAP Sybase IQ 提供了一整套游标，可以满足各种 ODBC 可滚动游标类型的要求。

通过调用定义语句属性的 SQLSetStmtAttr 函数，您可以设置所需的 ODBC 游标特性。您必须先调用 SQLSetStmtAttr，然后才能执行创建结果集的语句。

您可以使用 SQLSetStmtAttr 来设置很多游标特性。确定 SAP Sybase IQ 提供的游标类型的特性包括以下几种：

- **SQL_ATTR_CURSOR_SCROLLABLE** – 对于可滚动游标，设置为 SQL_SCROLLABLE；对于只进游标，设置为 SQL_NONSCROLLABLE。SQL_NONSCROLLABLE 是缺省设置。
- **SQL_ATTR_CONCURRENCY** – 设置为下列值之一：
  - **SQL_CONCUR_READ_ONLY** – 不允许更新。SQL_CONCUR_READ_ONLY 是缺省设置。

- **SQL_CONCUR_LOCK** – 使用足以确保该行能够得到更新的最低锁定级别。
- **SQL_CONCUR_ROWVER** – 使用优化的并发控制，比较数据行的版本（例如，SQLBase ROWID 或 Sybase TIMESTAMP）。
- **SQL_CONCUR_VALUES** – 使用优化的并发控制，比较值。

有关详细信息，请参见 Microsoft 的 ODBC API Reference 中的 SQLSetStmtAttr，网址为 http://msdn.microsoft.com/zh-cn/library/ms712631.aspx。

示例

下面的代码段请求一个只读的、可滚动游标：

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CURSOR_SCROLLABLE,
      SQL_SCROLLABLE, SQL_IS_UINTEGER );
```

数据检索

若要从数据库检索行，可使用 SQLExecute 或 SQLExecDirect 执行 SELECT 语句。这将为该语句打开一个游标。

然后，使用 SQLFetch 或 SQLFetchScroll 通过游标读取行。这些函数从结果集读取下一个数据行集，并为所有绑定列返回数据。使用 SQLFetchScroll，可在绝对位置或相对位置，或者通过书签来指定行集。SQLFetchScroll 替换 ODBC 2.0 规范中旧的 SQLExtendedFetch。

应用程序使用 SQLFreeHandle 释放语句后，它将关闭游标。

若要从游标读取值，您的应用程序可以使用 SQLBindCol 或 SQLGetData。如果使用 SQLBindCol，则会在每次读取时自动检索值。如果使用 SQLGetData，您必须在每次读取后为每一列调用它。

SQLGetData 用来逐段读取 LONG VARCHAR 或 LONG BINARY 之类的列的值。另一种方法，还可以将 SQL_ATTR_MAX_LENGTH 语句属性设置为一个很大的值，使之能够包含列的整个值。SQL_ATTR_MAX_LENGTH 的缺省值为 256 KB。

SAP Sybase IQ ODBC 驱动程序实现 SQL_ATTR_MAX_LENGTH 的方式与 ODBC 规范的目标方式不同。SQL_ATTR_MAX_LENGTH 的用意是作为截断较大读取的机制来使用。它可以 "预览" 模式实现，其中仅显示数据的第一部分。例如：不是从服务器向客户端应用程序传输 4 MB blob，而是只传输前 500 字节（通过设置 SQL_ATTR_MAX_LENGTH 为 500）。SAP Sybase IQ ODBC 驱动程序不支持此实现。

以下代码段在查询时打开游标，并通过游标检索数据。为了使示例更容易读，这里省略了错误检查。该代码段节选自一个完整示例，可在 `%IQDIRSAMP16%\SQLAnywhere\ODBCSelect\odbcselect.cpp` 下找到。

```
SQLINTEGER cbDeptID = 0, cbDeptName = SQL_NTS, cbManagerID = 0;
SQLCHAR deptName[ DEPT_NAME_LEN + 1 ];
SQLSMALLINT deptID, managerID;
SQLHENV  env;
SQLHDBC  dbc;
```

```
SQLHSTMT stmt;
SQLRETURN rc;

SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
SQLSetEnvAttr( env,
            SQL_ATTR_ODBC_VERSION,
            (void *)SQL_OV_ODBC3, 0 );
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLConnect( dbc,
            (SQLCHAR *) "SAP Sybase IQ 16 Demo", SQL_NTS,
            (SQLCHAR *) "DBA", SQL_NTS,
            (SQLCHAR *) "sql", SQL_NTS );
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLBindCol( stmt, 1,
            SQL_C_SSHORT, &deptID, 0, &cbDeptID );
SQLBindCol( stmt, 2,
            SQL_C_CHAR, deptName,
            sizeof(deptName), &cbDeptName );
SQLBindCol( stmt, 3,
            SQL_C_SSHORT, &managerID, 0, &cbManagerID );
SQLExecDirect( stmt, (SQLCHAR * )
    "SELECT DepartmentID, DepartmentName, DepartmentHeadID "
    "FROM Departments "
    "ORDER BY DepartmentID", SQL_NTS );
while( ( rc = SQLFetch( stmt ) ) != SQL_NO_DATA )
{
   printf( "%d %20s %d\n", deptID, deptName, managerID );
}
SQLFreeHandle( SQL_HANDLE_STMT, stmt );
SQLDisconnect( dbc );
SQLFreeHandle( SQL_HANDLE_DBC, dbc );
SQLFreeHandle( SQL_HANDLE_ENV, env );
```

在游标中可读取的行位置编号受整数的大小制约。您最多可以读取到第 2147483646 行，这个数字比可以在 32 位整数中保存的值小 1。在使用负数（从末尾开始计算行数）时，您最多可以读取到的行数比整数中可保存的最大负值大 1。

通过游标更新和删除行

Microsoft 的 ODBC 程序员参考建议使用 SELECT ...FOR UPDATE 来指示查询可以使用定位操作进行更新。您不必在 SAP Sybase IQ 中使用 FOR UPDATE 子句：只要满足以下条件，即可自动更新 SELECT 语句：

- 基础的查询支持更新。
  就是说，只要结果中各列内的数据操作语句有意义，定位的数据操作语句就可在游标上执行。
  ansi_update_constraints 数据库选项将查询的类型限制为可更新的类型。
- 游标类型支持更新。
  如果您使用的是只读游标，则不能更新结果集。

ODBC 提供两种方法来执行定位的更新和删除：

- 使用 SQLSetPos 函数。
  根据提供的参数（SQL_POSITION、SQL_REFRESH、SQL_UPDATE 和 SQL_DELETE），SQLSetPos 设置游标位置，让应用程序刷新、更新或删除结果集中的数据。
  这是 SAP Sybase IQ 中使用的方法。
- 使用 SQLExecute 发送定位的 UPDATE 和 DELETE 语句。这种方法不得用于 SAP Sybase IQ 中。

书签

ODBC 提供**书签**，书签是用于游标中的行的值。对于对值敏感的游标和不敏感游标，SAP Sybase IQ 支持书签。例如，ODBC 游标类型 SQL_CURSOR_STATIC 和 SQL_CURSOR_KEYSET_DRIVEN 支持书签，而游标类型 SQL_CURSOR_DYNAMIC 和 SQL_CURSOR_FORWARD_ONLY 不支持书签。

在 ODBC 3.0 之前，数据库只能指定它是否支持书签：没有为每个游标类型提供此信息的接口。数据库服务器没有办法指示支持的游标书签类型。对于 ODBC 2 应用程序，SAP Sybase IQ 会返回信息说明它的确支持书签。这样，虽然您尝试将书签用于动态游标时不会遇到什么阻碍，但是您不应当这样组合使用。

存储过程注意事项

本节介绍如何创建和调用存储过程以及如何处理来自 ODBC 应用程序的结果。

过程和结果集
有两种类型的过程：返回结果集的过程和不返回结果集的过程。您可以使用 SQLNumResultCols 来指出差异：如果过程不返回结果集，则结果列数为零。如果有结果集，则可像任何其它游标一样使用 SQLFetch 或 SQLExtendedFetch 来读取值。

过程的参数应当使用参数标记（问号）来传递。使用 SQLBindParameter 可为每个参数标记指派存储区域，无论是 INPUT、OUTPUT 还是 INOUT 参数都可以。

要处理多个结果集，ODBC 必须描述当前正在执行的游标，而不是描述由过程定义的结果集。因此，ODBC 并不总是按照存储过程定义中的 RESULT 子句中的定义来描述列名称。要避免出现此问题，您可以在过程结果集游标中使用列别名。

示例 1

此示例创建和调用不返回结果集的过程。该过程采用一个 INOUT 参数，并会增加它的值。在此示例中，变量 [num_columns] 的值为零，这是因为该过程不返回结果集。为了使示例更容易读，这里省略了错误检查。

```
HDBC dbc;
SQLHSTMT stmt;
SQLINTEGER I;
SQLSMALLINT num_columns;

SQLAllocStmt( dbc, &stmt );
```

```
SQLExecDirect( stmt,
    "CREATE PROCEDURE Increment( INOUT a INT )"
    "BEGIN "
    "    SET a = a + 1 "
    "END", SQL_NTS );

/* Call the procedure to increment 'I' */
I = 1;
SQLBindParameter( stmt, 1, SQL_C_LONG, SQL_INTEGER, 0, 0, &I, NULL );
SQLExecDirect( stmt, "CALL Increment( ? )", SQL_NTS );
SQLNumResultCols( stmt, &num_columns );
```

示例 2

此示例调用一个返回结果集的过程。在此示例中，变量 [num_columns] 的值为 2，因为该过程返回的结果集具有两列。同样，错误检查已被省略，这样该示例便更容易阅读。

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLSMALLINT num_columns;

SQLCHAR ID[ 10 ];
SQLCHAR Surname[ 20 ];

SQLExecDirect( stmt,
      "CREATE PROCEDURE EmployeeList() "
      "RESULT( ID CHAR(10), Surname CHAR(20) ) "
      "BEGIN "
      "    SELECT EmployeeID, Surname FROM Employees "
      "END", SQL_NTS );

/* Call the procedure - print the results */
SQLExecDirect( stmt, "CALL EmployeeList()", SQL_NTS );
SQLNumResultCols( stmt, &num_columns );
SQLBindCol( stmt, 1, SQL_C_CHAR, &ID, sizeof(ID), NULL );
SQLBindCol( stmt, 2, SQL_C_CHAR, &Surname, sizeof(Surname), NULL );

for( ;; )
{
   rc = SQLFetch( stmt );
   if( rc == SQL_NO_DATA_FOUND )
   {
      rc = SQLMoreResults( stmt );
      if( rc == SQL_NO_DATA_FOUND ) break;
   }
   else
   {
      do_something( ID, Surname );
   }
}
```

ODBC 转义语法

您可以使用任意 ODBC 应用程序的 ODBC 转义语法。此转义语法用于调用一组普通函数，无论您正在使用哪种数据库管理系统。转义语法的一般格式为

{ keyword *parameters* }

这组关键字包括：

- **{d date-string}** – 日期字符串是 SAP Sybase IQ 接受的任意日期值。
- **{t time-string}** – 时间字符串是 SAP Sybase IQ 接受的任意时间值。
- **{ts date-string time-string}** – 日期/时间字符串是 SAP Sybase IQ 接受的任意时间戳值。
- **{guid uuid-string}** – uuid-string 是任意有效的 GUID 字符串，例如 41dfe9ef-db91-11d2-8c43-006008d26a6f。
- **{oj outer-join-expr}** – outer-join-expr 是 SAP Sybase IQ 接受的有效 OUTER JOIN 表达式。
- **{? = call func(p1,...)}** – 此函数是 SAP Sybase IQ 接受的任意有效函数调用。
- **{call proc(p1,...)}** – 此过程是 SAP Sybase IQ 接受的任意有效存储过程调用。
- **{fn func(p1,...)}** – 此函数为下文所列函数库中的任一函数。

您可以使用转义语法访问由 ODBC 驱动程序实现的函数库，这些函数包括数字、字符串、时间、日期和系统函数。

例如，要以与数据库管理系统无关的方式获得当前日期，您需要执行以下语句：

```
SELECT { FN CURDATE() }
```

下面的表格列出了 SAP Sybase IQ ODBC 驱动程序支持的函数。

SAP Sybase IQ ODBC 驱动程序支持的函数

数字函数	字符串函数	系统函数	时间/日期函数
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH	CONVERT	CURRENT_TIMESTAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH

数字函数	字符串函数	系统函数	时间/日期函数
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOUR
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		WEEK
ROUND	RTRIM		YEAR
SIGN	SOUNDEX		
SIN	SPACE		
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

ODBC 转义函数与 JDBC 转义函数完全相同。在使用 JDBC 的 Interactive SQL 中，大括号*必须*成对使用。连续的括号之间不得有空格：“{{” 是允许的，但 “{ {” 是不允许的。同样，您不能在语句中使用新行字符。由于转义语法不是由 Interactive SQL 解析，所以在存储过程中不能使用转义语法。

例如，要使用 SQL 转义语法通过 sa_db_info 过程获得数据库属性，您将在 Interactive SQL 中执行以下语句：

```
{{CALL sa_db_info( 0 ) }}
```

ODBC 中的错误处理

ODBC 中的错误是使用来自每个 ODBC 函数调用的返回值和 SQLError 函数或 SQLGetDiagRec 函数的返回值进行报告的。SQLError 函数用于 ODBC 版本 3 之前的版本（但不包括版本 3）。自版本 3 起，已不建议使用 SQLError 函数，此函数已被 SQLGetDiagRec 函数取代。

每个 ODBC 函数都返回一个 SQLRETURN，它是以下状态代码之一：

状态代码	说明
SQL_SUCCESS	无错误。
SQL_SUCCESS_WITH_INFO	该函数完成，但是对 SQLError 的调用将显示警告。 这种状态最常见的情况是：返回的值太长，应用程序提供的缓冲区不够用。
SQL_ERROR	函数未完成，因为出现了错误。调用 SQLError 可获取有关此问题的详细信息。
SQL_INVALID_HANDLE	作为参数传递的环境、连接或语句句柄无效。 如果在释放句柄后再使用该句柄，或者句柄为空值指针，则通常会发生这种情况。
SQL_NO_DATA_FOUND	没有可用信息。 使用这种状态的最常见情况是在从游标进行读取时，这种状态表示游标中没有更多行。
SQL_NEED_DATA	参数需要数据。 这是一项高级特性，SQLParamData 和 SQLPutData 下面的 ODBC SDK 文档对其作了说明。

每个环境、连接和语句句柄都可能有与之相关联的一个或多个错误或警告。每个对 SQLError 或 SQLGetDiagRec 的调用都会返回有关一个错误的信息，然后删除有关该错误的信息。如果您不调用 SQLError 或 SQLGetDiagRec 来删除所有错误，则会在执行下一个将同一句柄作为参数来传递的函数调用时，删除这些错误。

每个对 SQLError 的调用都会传递三个句柄，这些句柄分别用于环境、连接和语句。第一个调用使用 SQL_NULL_HSTMT 获取与连接相关联的错误。同样，使用 SQL_NULL_DBC 和 SQL_NULL_HSTMT 的调用将获取任何与环境句柄相关联的错误。

每个对 SQLGetDiagRec 的调用将传递环境、连接或语句句柄。第一个调用传递句柄类型 SQL_HANDLE_DBC 以获取与连接关联的错误。第二个调用传递句柄类型 SQL_HANDLE_STMT，以获取与刚执行的语句相关联的错误。

如果有要报告的错误（*不是* SQL_ERROR），SQLError 和 SQLGetDiagRec 将返回 SQL_SUCCESS；如果没有其它要报告的错误，则将返回 SQL_NO_DATA_FOUND。

示例 1

以下代码段使用了 SQLError，并返回代码：

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
UCHAR errmsg[100];
```

```
rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
   SQLError( env, dbc, SQL_NULL_HSTMT, NULL, NULL,
         errmsg, sizeof(errmsg), NULL );
   print_error( "Allocation failed", errmsg );
   return;
}

/* Delete items for order 2015 */
rc = SQLExecDirect( stmt,
       "DELETE FROM SalesOrderItems WHERE ID=2015",
       SQL_NTS );
if( rc == SQL_ERROR )
{
   SQLError( env, dbc, stmt, NULL, NULL,
            errmsg, sizeof(errmsg), NULL );
   print_error( "Failed to delete items", errmsg );
   return;
}
```

示例 2

以下代码段使用了 SQLGetDiagRec，并返回代码:

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLSMALLINT errmsglen;
SQLINTEGER errnative;
SQLCHAR errmsg[255];
SQLCHAR errstate[5];

rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
   SQLGetDiagRec( SQL_HANDLE_DBC, dbc, 1, errstate,
       &errnative, errmsg, sizeof(errmsg), &errmsglen );
   print_error( "Allocation failed", errstate, errnative, errmsg );
   return;
}

rc = SQLExecDirect( stmt,
       "DELETE FROM SalesOrderItems WHERE ID=2015",
       SQL_NTS );
if( rc == SQL_ERROR )
{
   SQLGetDiagRec( SQL_HANDLE_STMT, stmt, 1, errstate,
       &errnative, errmsg, sizeof(errmsg), &errmsglen );
   print_error( "Failed to delete items", errstate, errnative,
errmsg );
   return;
}
```

数据库中的 Java

SAP Sybase IQ 提供了用于在数据库服务器环境中执行 Java 类的机制。在数据库服务器中使用 Java 方法为向数据库添加编程逻辑提供了有效方式。

数据库中的 Java 支持具有下列优点：

- 在应用程序的不同层级（客户端、中间层或服务器）中重复使用 Java 组件，哪个层级最适用，就将它们用于哪个层级。SAP Sybase IQ 成为用于分布式计算的平台。
- 与 SQL 存储过程语言相比，Java 为在数据库中内置逻辑提供了一种功能更强大的语言。
- Java 可用于数据库服务器中，但不会危害数据库和服务器的完整性、安全性或可靠性。

SQLJ 标准
数据库中的 Java 基于 SQLJ 第 1 部分建议的标准 (ANSI/INCITS 331.1-1999) 构建而成。SQLJ 第 1 部分提供了有关将 Java 静态方法作为 SQL 存储过程和函数来调用的规范。

数据库中的 Java 常见问题解答

本节介绍数据库中的 Java 的主要功能。

数据库中的 Java 有哪些主要功能？

下面所有要点的详细解释将在后面的几节中提供。

- **可以在数据库中运行 Java** – 外部 Java VM 代表数据库服务器来运行 Java 代码。
- **可以从 Java 访问数据** – SAP Sybase IQ 让您通过 Java 来访问数据。
- **保留 SQL** – 使用 Java 不会改变现有 SQL 语句的行为，也不会改变非 Java 关系数据库行为的其它方面。

如何在数据库中使用自己的 Java 类？

Java 语言比 SQL 更强大。Java 是一种面向对象的语言，因此它的指令（源代码）采用类的形式。要在数据库中执行 Java，应在数据库外编写 Java 指令并在数据库外将它们编译为已编译的类（*字节代码*），这些类是包含 Java 指令的二进制文件。

已编译的类和存储过程一样可从客户端应用程序轻松调用，其调用方式也相同。Java 类可以同时包含有关主题的信息和某些计算逻辑。例如：您能设计、编写并编译 Java 代码来创建一个 Employees 类。此类包括对雇员表的各种操作方法。将 Java 类作为对象安装到数据库中，同时编写 SQL 覆盖函数或过程调用 Java 类中的方法。

安装之后，即可使用存储过程从数据库服务器执行这些类。例如，下面的语句将创建到 Java 过程的接口：

```
CREATE PROCEDURE MyMethod()
EXTERNAL NAME 'JDBCExample.MyMethod()V'
LANGUAGE JAVA;
```

SAP Sybase IQ 是 Java 类的运行时环境，而不是 Java 开发环境。您需要一个 Java 开发环境来编写和编译 Java，如 Java 开发工具包（Java Development Kit，简称 JDK）。您还需要具备 Java 运行时环境才能执行 Java 类。

您可以使用 Java 开发工具包附带的 Java API 中的许多类。您还可以使用 Java 开发人员创建和编译的类。

Java 在数据库中是如何执行的？

SAP Sybase IQ 启动 Java VM。Java VM 将解释已编译的 Java 指令并代表数据库服务器来运行它们。数据库服务器会在需要时自动启动 Java VM：您不必执行任何显式操作来启动或停止 Java VM。

数据库服务器中的 SQL 请求处理器已经进行了扩展，因此它可以向 Java VM 中发出调用来执行 Java 指令。另外，它还可以处理来自 Java VM 的请求以便能够从 Java 进行数据访问。

Java 错误处理

Java 应用程序中的错误会生成一个表示错误的异常对象（称作*抛出异常*）。如果未能在应用程序的某个层级捕获并适当处理抛出的异常，则该异常将终止 Java 程序。

无论是 Java API 类还是自定义创建的类都有可能抛出异常。事实上，用户可以创建自己的异常类，这些类会抛出其自定义创建的错误类。

如果发生异常的方法的主体中没有异常处理程序，则会继续沿着调用堆栈向上搜索异常处理程序。如果到达调用堆栈的顶部仍未找到异常处理程序，则会调用运行该应用程序的 Java 解释器的缺省异常处理程序，同时程序会终止。

在 SAP Sybase IQ 中，如果 SQL 语句调用 Java 方法，并抛出了未处理的异常，则会生成一个 SQL 错误。Java 异常以及 Java 堆栈跟踪的全文将在服务器消息窗口中显示。

如何将 Java 类安装到数据库中

可将 Java 类作为单个类或 JAR 安装到数据库中。

- **单一类** – 可从已编译的类文件中将单个类安装到数据库中。类文件的扩展名通常为 `.class`。

- **JAR** – 如果某组类位于已压缩或未压缩的 JAR 文件中，您可以一次性地安装这组中的所有类。JAR 文件的扩展名通常为 `.jar` 或 `.zip`。SAP Sybase IQ 支持所有用 JAR 实用程序创建的 JAR 压缩文件以及其它一些 JAR 压缩模式。

类文件创建

您在创建自己的类时会涉及许多步骤，每一步的具体情况会因您是否使用 Java 开发工具而有所不同，但一般都会包括以下步骤：

1. 定义类。
   编写定义类的 Java 代码。
2. 命名和保存类。
   将类声明（Java 代码）保存在扩展名为 `.java` 的文件中。确保文件名与类名相同，并且这两个名称的大小写一致。
   例如，名为 Utility 的类应保存在名为 `Utility.java` 的文件中。
3. 编译类。
   此步骤会将包含 Java 代码的类声明转化为一个不同的包含字节代码的新文件。新文件的名称与 Java 代码文件的名称相同，但是扩展名为 `.class`。您可以在 Java 运行时环境中运行已编译的 Java 类，而不必考虑编译它时所使用的平台或运行时环境的操作系统是什么。

数据库中 Java 类的特殊功能

本节介绍在数据库中使用的 Java 类的功能。

如何调用 Main 方法

通常都是通过对具有 main 方法的类运行 Java VM 来启动 Java 应用程序（在数据库外部）。

例如，文件 `%ALLUSERSPROFILE%\SybaseIQ\samples\JavaInvoice\Invoice.java` 中的 Invoice 类具有一个 main 方法。当使用如下所示的命令从命令行执行该类时，执行的是 main 方法。

```
java Invoice
```

Java 应用程序中的线程

利用 java.lang.Thread 包的功能，可以在 Java 应用程序中使用多个线程。

您可在 Java 应用程序中同步、挂起、重新开始、中断或停止线程。

无此类方法例外

如果在调用 Java 方法时所提供的参数数量不正确，或者所使用的数据类型不正确，Java VM 将用 `java.lang.NoSuchMethodException` 错误做出响应。检查参数的数量和类型。

如何从 Java 方法返回结果集

编写一个向调用环境返回结果集的 Java 方法，并将此方法包装在一个被声明为 LANGUAGE JAVA 的 EXTERNAL NAME 的 SQL 存储过程中。

执行以下任务以从 Java 方法返回结果集：

1. 确保在一个公共类中将 Java 方法声明为公共的和静态的。
2. 对于您期望该方法返回的每个结果集，要确保该方法有一个类型为 java.sql.ResultSet[] 的参数。这些结果集参数都必须出现在参数列表的末尾处。
3. 在该方法中，首先创建一个 java.sql.ResultSet 实例，然后将其指派给其中一个 ResultSet[] 参数。
4. 创建一个类型为 EXTERNAL NAME LANGUAGE JAVA 的 SQL 存储过程。该类型的过程是 Java 方法的包装。您可以像对其它任何返回结果集的过程一样对 SQL 过程结果集使用游标。

示例

下面的简单类有一个方法，该方法执行查询并将结果集返回给调用环境。

```
import java.sql.*;

public class MyResultSet
{
  public static void return_rset( ResultSet[] rset1 )
       throws SQLException
  {
    Connection conn = DriverManager.getConnection(
                        "jdbc:default:connection" );
    Statement stmt = conn.createStatement();
    ResultSet rset =
      stmt.executeQuery (
                        "SELECT Surname " +
                        "FROM Customers" );
    rset1[0] = rset;
  }
}
```

您可以使用 CREATE PROCEDURE 语句公开结果集，该语句将指明从过程返回的结果集的数量以及 Java 方法的签名。

指明结果集的 CREATE PROCEDURE 语句可以定义如下：

```
CREATE PROCEDURE result_set()
  RESULT (SurName person_name_t)
  DYNAMIC RESULT SETS 1
  EXTERNAL NAME
```

```
    'MyResultSet.return_rset([Ljava/sql/ResultSet;)V'
  LANGUAGE JAVA;
```

您可以对此过程打开游标，就像对任何返回结果集的 SAP Sybase IQ 过程一样。

字符串 `([Ljava/sql/ResultSet;)V` 是一个 Java 方法签名，它是参数和返回值的数量及类型的精简字符表示形式。

通过存储过程从 Java 返回的值

您可以将使用 EXTERNAL NAME LANGUAGE JAVA 创建的存储过程用作 Java 方法的包装。本节介绍如何编写 Java 方法以在存储过程中利用 OUT 或 INOUT 参数。

Java 不显式支持 INOUT 或 OUT 参数。相反，您可以使用参数数组。例如，要使用整型 OUT 参数，请创建一个刚好由一个整数组成的数组：

```
public class Invoice
{
  public static boolean testOut( int[] param )
  {
    param[0] = 123;
    return true;
  }
}
```

以下过程使用 testOut 方法：

```
CREATE PROCEDURE testOut( OUT p INTEGER )
EXTERNAL NAME 'Invoice.testOut([I)Z'
LANGUAGE JAVA;
```

字符串 `([I)Z` 是一个 Java 方法签名，用于指明该方法有一个单个参数（是一个由整数构成的数组）并会返回一个布尔值。定义该方法，以使要用作 OUT 或 INOUT 参数的方法参数成为一个对应于 OUT 或 INOUT 参数的 SQL 数据类型的 Java 数据类型的数组。

要对此进行测试，请使用未初始化的变量调用该存储过程。

```
CREATE VARIABLE zap INTEGER;
CALL testOut( zap );
SELECT zap;
```

结果集是 `123`。

Java 的安全管理

Java 提供了安全管理器，您可以使用它们控制用户对应用程序的安全敏感功能的访问，如文件访问和网络访问。您应利用 Java VM 所支持的安全管理功能。

如何启动和停止 Java VM

Java VM 将在执行第一个 Java 操作时自动装载。要显式装载它以便为执行 Java 操作做好准备，可通过执行以下语句来完成此任务：

```
START JAVA;
```

可在不使用 Java 时使用 STOP JAVA 语句卸载 Java VM。语法是：

```
STOP JAVA;
```

Java VM 中的关闭挂接

用于提供数据库中的 JAVA 支持的 SAP Sybase IQ JAVA VM ClassLoader 允许应用程序安装关闭挂接。这些关闭挂接类似于应用程序随 JVM Runtime 一同安装的关闭挂接。

对于在数据库支持中使用 JAVA 的连接，当其执行 STOP JAVA 语句或断开连接时，用于该连接的 ClassLoader 会在卸载之前运行已为该特定连接安装的所有关闭挂接。对于安装了数据库中所有 JAVA 类的数据库应用程序中的常规 Java，不必安装关闭挂接。使用 ClassLoader 关闭挂接时应格外小心，它只能用来清理系统范围内的任何资源，这些资源是为用于停止 Java 的特定连接而分配的。此外，在关闭挂接中不允许 jdbc:default JDBC 请求，因为在调用 ClassLoader 关闭挂接之前，jdbc:default 连接已关闭。

要使用 SQL Anywhere Java VM ClassLoader 安装关闭挂接，应用程序必须将 `sajvm.jar` 包括在 Java 编译器类路径内，它还必须执行如下代码：

```
SDHookThread hook = new SDHookThread( ... );
ClassLoader classLoader =
Thread.currentThread().getContextClassLoader();
((ianywhere.sa.jvm.SAClassLoader)classLoader).addShutdownHook( hook
 );
```

SDHookThread 类扩展了标准 Thread 类，并且上述代码必须由 ClassLoader 为当前连接所装载的类来执行。在数据库内安装且随后通过外部环境调用而调用的任何类都由正确的 SQL Anywhere Java VM ClassLoader 自动执行。

要从 SQL Anywhere Java VM ClassLoader 列表中删除关闭挂接，应用程序需要执行类似于下例的代码：

```
ClassLoader classLoader =
Thread.currentThread().getContextClassLoader();
((ianywhere.sa.jvm.SAClassLoader)classLoader).removeShutdownHook( h
ook );
```

以上代码必须由 ClassLoader 为当前连接所装载的类来执行。

JDBC CLI

JDBC 是 Java 应用程序的调用层接口。JDBC 提供了与各种关系数据库的统一接口，并且为创建各种更高级别的工具和接口提供了一个公共基础。JDBC 现在是 Java 的标准组成部分并被包括在 JDK 中。

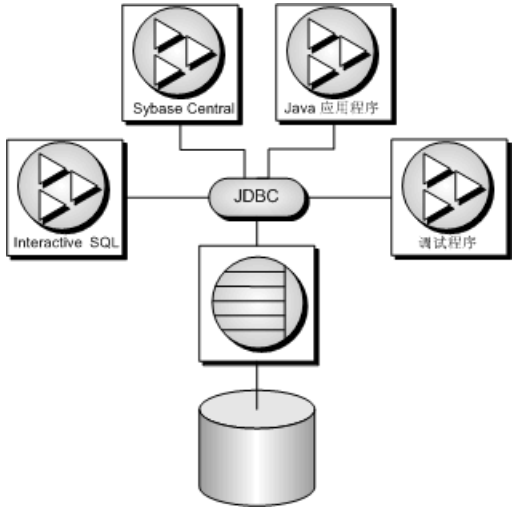
SAP Sybase IQ 包含 4.0 驱动程序（2 类驱动程序）。

SAP Sybase IQ 也支持纯 Java JDBC 驱动程序（称作 jConnect），可以从 SAP 获取。

除了将 JDBC 用作客户端应用程序编程接口外，您还可以通过在数据库中使用 Java 来在数据库服务器内使用 JDBC 访问数据。

JDBC 应用程序

您可以开发使用 JDBC API 连接到 SAP Sybase IQ 的 Java 应用程序。随 SAP Sybase IQ 提供的多个应用程序都使用 JDBC，例如 Interactive SQL。



Java 和 JDBC 也是用于开发 UltraLite® 应用程序的重要编程语言。

您既可从客户端应用程序使用 JDBC，也可从数据库内部使用 JDBC。在数据库中加入编程逻辑时，使用 JDBC 的 Java 类也可以起到 SQL 存储过程的作用，而且功能比 SQL 存储过程更强。

JDBC 为 Java 应用程序提供了 SQL 接口：要从 Java 访问关系数据，可利用 JDBC 调用进行此访问。

短语*客户端应用程序*指在用户计算机上运行的应用程序和在中层应用程序服务器上运行的逻辑。

这些示例说明在 SAP Sybase IQ 中使用 JDBC 时的一些独特功能。有关 JDBC 编程的详细信息，请参见讲解 JDBC 编程的书籍。

可通过以下方式配合使用 JDBC 和 SAP Sybase IQ：

- **客户端的 JDBC** – Java 客户端应用程序可以对 SAP Sybase IQ 进行 JDBC 调用。通过 JDBC 驱动程序建立连接。

  SAP Sybase IQ 包含 JDBC 4.0 驱动程序（2 类 JDBC 驱动程序）；并且还支持纯 Java 应用程序的 jConnect 驱动程序（4 类 JDBC 驱动程序）。
- **数据库中的 JDBC** – 数据库中安装的 Java 类可使用内部 JDBC 驱动程序进行 JDBC 调用，以此来访问和修改数据库中的数据。

JDBC 资源

- **示例源代码** – 本节示例的源代码位于 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` 目录下。
- **JDBC 规范** – 要获得 JDBC 数据访问 API 的详细信息，请访问 http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html。
- **必需的软件** – 使用 jConnect 驱动程序需要 TCP/IP。

  jConnect 驱动程序可以从 http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect 获得。

JDBC 驱动程序

SAP Sybase IQ 支持以下 JDBC 驱动程序：

- **SQL Anywhere 16 JDBC 4.0 驱动程序** – 此驱动程序使用 Command Sequence 客户端/服务器协议与 SAP Sybase IQ 进行通信。它的行为与 ODBC、嵌入式 SQL 和 OLE DB 应用程序是一致的。建议使用 SQL Anywhere 16 JDBC 4.0 驱动程序连接 SAP Sybase IQ 数据库。JDBC 4.0 驱动程序只能和 JRE 1.6 或更高版本一同使用。

  JDBC 4.0 驱动程序利用了新 JDBC 驱动程序自动注册。因此，如果应用程序要使用 JDBC 4.0 驱动程序，它不再需要执行 Class.forName 调用来加载 JDBC 驱动程序。而只要在类文件路径中放入一个 `sajdbc4.jar` 文件，并简单地用以 jdbc:sqlanywhere 开头的 URL 调用 DriverManager.getConnection() 即可。

  JDBC 4.0 驱动程序包含清单信息以便将其作为开放服务网关协议（Open Services Gateway initiative，简称 OSGi）软件包进行装载。

  使用 JDBC 4.0 驱动程序后，NCHAR 数据的元数据将以 java.sql.Types.NCHAR、NVARCHAR 或 LONGNVARCHAR 返回列的类型。此外，应用程序现在还可以使用

Get/SetNString 或 Get/SetNClob 方法替代 Get/SetString 和 Get/SetClob 方法来获取数据。

- **jConnect** – 此驱动程序是 100% 纯 Java 驱动程序。它使用 TDS 客户端/服务器协议与 SAP Sybase IQ 进行通信。

  jConnect 和 jConnect 文档可以从 http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect 获得。

在选择使用哪个驱动程序时，应当考虑下列因素：

- **功能** – SQL Anywhere 16 JDBC 4.0 驱动程序和 jConnect 与 JDBC 4.0 兼容。在连接到 SAP Sybase IQ 数据库时，SQL Anywhere 16 JDBC 驱动程序提供可完全滚动的游标。当连接到 SAP Sybase IQ 数据库服务器时，jConnect JDBC 驱动程序提供可滚动游标，但结果集在客户端上高速缓存。当连接到 SAP Adaptive Server® Enterprise 数据库时，jConnect JDBC 驱动程序提供可完全滚动的游标。

  JDBC 4.0 API 文档可以从 http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html 获得。
- **纯 Java** – jConnect 驱动程序是纯 Java 解决方案。SQL Anywhere 16 JDBC 驱动程序基于 SQL Anywhere 16 ODBC 驱动程序，并非纯粹的 Java 解决方案。
- **性能** – 在多数情况下，SQL Anywhere 16 JDBC 驱动程序所提供的性能要比 jConnect 驱动程序好一些。
- **兼容性** – jConnect 驱动程序使用的 TDS 协议可与 Adaptive Server 共享。该驱动程序行为的某些方面受此协议的控制，并被配置为与 Adaptive Server 兼容。

有关 SQL Anywhere 16 JDBC 驱动程序和 jConnect 的平台可用性的详细信息，请参见 http://www.sybase.com/detail?id=1061806。

JDBC 程序结构

JDBC 应用程序中通常会发生以下事件序列：

- **创建连接对象** – 调用 DriverManager 类的 getConnection 类方法即可创建 Connection 对象，并与数据库建立连接。
- **生成语句对象** – Connection 对象会生成 Statement 对象。
- **传递 SQL 语句** – 将在数据库环境内部执行的 SQL 语句传递到 Statement 对象。如果该语句是一个查询，则此操作会返回 ResultSet 对象。

  ResultSet 对象包含 SQL 语句返回的数据，但一次只显示一行（与游标的工作方式类似）。
- **遍历结果集的行** – ResultSet 对象的 next 方法可执行两种操作：
  - 当前行（通过 ResultSet 对象显示的结果集内的行）前进一行。
  - 返回布尔值，指示要前进到的目标行是否存在。

- **为每行检索值** – 通过识别 ResultSet 对象中每一列的名称或位置的方式来检索每一列的值。可以使用 getData 方法来获取当前行中某列的值。

Java 对象可使用 JDBC 对象与数据库进行交互，并获取数据以供自身使用。

客户端与服务器端 JDBC 连接的区别

客户端 JDBC 与数据库服务器中的 JDBC 之间的区别是它们与数据库环境建立连接的方式不同。

- **客户端** – 在客户端 JDBC 中，建立连接需要 SQL Anywhere JDBC 驱动程序或 jConnect JDBC 驱动程序。将参数传递给 DriverManager.getConnection 就建立了连接。从客户端应用程序的角度看，数据库环境是外部应用程序。
- **服务器端** – 在数据库服务器中使用 JDBC 时，连接已经存在。字符串 "jdbc:default:connection" 会传递给 DriverManager.getConnection，这可使 JDBC 应用程序在当前用户连接中工作。这是一个快速、有效而安全的操作，因为客户端应用程序已通过了数据库安全检查建立了连接。用户 ID 和口令一经提供，便不需要再次提供。服务器端 JDBC 驱动程序只能连接到当前连接的数据库。

您可以编写 JDBC 类，以便使用一个条件语句构造 URL，让 JDBC 类既能在客户端运行，又能在服务器端运行。外部连接需要主机名和端口号，而内部连接需要 "jdbc:default:connection"。

SQL Anywhere JDBC 驱动程序

与纯 Java jConnect JDBC 驱动程序相比，SQL Anywhere JDBC 4.0 驱动程序提供了一些性能优势和功能优点，但此驱动程序不能提供纯 Java 解决方案。建议使用 SQL Anywhere JDBC 4.0 驱动程序。

如何装载 SQL Anywhere JDBC 4.0 驱动程序

确保 SQL Anywhere JDBC 4.0 驱动程序在类文件路径下。

```
set classpath=%IQDIR%\java\sajdbc4.jar;%classpath%
```

JDBC 4.0 驱动程序利用了新 JDBC 驱动程序自动注册。如果驱动程序在类文件路径中，在执行启动时它会被自动装载。

所需文件

SQL Anywhere JDBC 4.0 驱动程序的 Java 组件包括在 `sajdbc4.jar` 文件中，该文件安装在 SAP Sybase IQ 安装目录的 `Java` 子目录中。对于 Windows，本机组件是 `dbjdbc16.dll`，它位于 SAP Sybase IQ 安装目录的 `bin32` 或 `bin64` 子目录中；对于 Unix，本机组件是 `libdbjdbc16.so`。该组件必须位于系统路径中。

SQL Anywhere 16 JDBC 驱动程序连接字符串

要通过 SQL Anywhere 16 JDBC 驱动程序连接到某个数据库，您需要提供该数据库的 URL。例如：

```
Connection con = DriverManager.getConnection(
      "jdbc:sqlanywhere:DSN=Sybase IQ Demo" );
```

```
Connection con =
DriverManager.getConnection("jdbc:sqlanywhere:DSN=Sybase IQ Demo" );
```

该 URL 包含 `jdbc:sqlanywhere:`，后跟一个连接字符串。如果 `sajdbc4.jar` 文件在类文件路径下，则 JDBC 4.0 驱动程序已自动被装载并将处理该 URL。如示例中所示，可能为了方便指定了 ODBC 数据源 (`DSN`)，但也可以用通过分号隔开的显式连接参数来辅助或代替数据源连接参数。

如果不使用数据源，则必须在连接字符串中指定所有要求的连接参数：

```
Connection con = DriverManager.getConnection(

"jdbc:sqlanywhere:UserID=<user_id>;Password=<password>;Start=..." )
;
```

由于既没有用 ODBC 驱动程序也没有用 ODBC 驱动程序管理器，所以不需要 `Driver` 连接参数。如果它出现，将被忽略。

jConnect JDBC 驱动程序

jConnect 驱动程序可以单独下载。要在小程序中使用 JDBC，则必须使用 jConnect JDBC 驱动程序连接到 SAP Sybase IQ 数据库。

可在 http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect 中下载 jConnect 驱动程序。jConnect 文档也位于此页面上。

jConnect 驱动程序文件

jConnect 作为名为 `jconn4.jar` 的 JAR 文件提供。此文件位于 jConnect 安装位置。

为 jConnect 设置类文件路径

要想让应用程序使用 jConnect，则在编译和运行时，jConnect 类必须位于类文件路径中，以便 Java 编译器和 Java 运行时能够找到必要的文件。

以下命令可将 jConnect 驱动程序添加到现有的 CLASSPATH 环境变量中，其中 *jconnect-path* 是 jConnect 安装目录。

```
set classpath=jconnect-path\classes\jconn4.jar;%classpath%
```

导入 jConnect 类

jConnect 中的类全部位于 com.sybase.jdbc4.jdbc 中。您必须在每个源文件的开头导入这些类：

```
import com.sybase.jdbc4.jdbc.*
```

加密口令
SAP Sybase IQ 支持对 jConnect 连接进行口令加密。

在数据库中安装 jConnect 系统对象

要用 jConnect 访问系统表信息（数据库元数据），必须将 jConnect 系统对象添加到数据库中。

前提条件

您必须拥有 ALTER DATABASE 系统特权，而且必须是数据库的唯一连接用户。

升级前备份数据库文件。如果尝试升级数据库并且尝试失败，则该数据库变得不可用。

过程

使用 iqinit 实用程序时，jConnect 系统对象在缺省情况下安装到 SAP Sybase IQ 数据库中。您可以在创建数据库时将 jConnect 系统对象添加到数据库，也可以在以后升级数据库时再添加。

如何装载 jConnect 驱动程序

确保 jConnect 驱动程序在类文件路径中。驱动程序文件 `jconn4.jar` 位于 jConnect 安装目录的 `classes` 子目录中。

```
set classpath=.;c:\jConnect-7_0\classes\jconn4.jar;%classpath%
```

jConnect 驱动程序利用了新 JDBC 驱动程序自动注册。如果驱动程序在类文件路径中，在执行启动时它会被自动装载。

jConnect 驱动程序连接字符串

要通过 jConnect 连接到某个数据库，您需要提供该数据库的 URL。例如：

```
Connection con = DriverManager.getConnection(
      "jdbc:sybase:Tds:localhost:2638", "<user_id>", "<password>");
```

URL 按照以下方式组成：

```
jdbc:sybase:Tds:host:port
```

各组成部分如下：

- **jdbc:sybase:Tds** – 使用 TDS 应用程序协议的 jConnect JDBC 驱动程序。
- **host** – 运行服务器的计算机的 IP 地址或名称。如果要建立同一主机连接，则可使用 localhost，它表示您登录的计算机系统。
- **port** – 数据库服务器监听时使用的端口号。分配给 SAP Sybase IQ 的端口号是 2638。如果没有特殊原因，请使用该端口号。

连接字符串长度必须小于 253 个字符。

如果使用 SAP Sybase IQ 个人服务器，确保启动服务器时包含 TCP/IP 支持选项。

如何使用 jConnect 连接字符串指定数据库

每个 SAP Sybase IQ 数据库服务器一次能装载一个或多个数据库。如果通过 jConnect 连接时提供的 URL 只指定了服务器，而没有指定数据库，则尝试连接的数据库是服务器上的缺省数据库。

您可以通过下列任一方式提供 URL 的扩展形式，由此而指定特定数据库。

使用 ServiceName 参数

```
jdbc:sybase:Tds:host:port?ServiceName=database
```

使用问号，后面跟随一系列赋值，这是向 URL 提供参数的标准方式。ServiceName 是否大写并不重要，但 = 号两边不能有空格。*database* 参数是数据库名，而不是服务器名。数据库名不得包括路径或文件后缀。例如：

```
Connection con = DriverManager.getConnection(
      "jdbc:sybase:Tds:localhost:2638?ServiceName=demo", "DBA",
"sql");
```

使用 RemotePWD 参数

存在一个用于将其它连接参数传递到服务器的解决方案。

这一技术允许您使用 RemotePWD 字段来提供其它连接参数，如数据库名或数据库文件。您可用 put 方法将 RemotePWD 设置成一个“属性”字段。

以下代码对如何使用该字段进行了说明。

```
import java.util.Properties;
.
.
.
Properties props = new Properties();
props.put( "User", "DBA" );
props.put( "Password", "sql" );
props.put( "RemotePWD", ",DatabaseFile=mydb.db" );

Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638", props );
```

如示例中所示，在 DatabaseFile 连接参数前必须加上一个逗号。利用 DatabaseFile 参数，您可使用 jConnect 启动服务器上的数据库。缺省情况下，数据库启动时 AutoStop=YES。如果向 DatabaseFile (DBF) 或 DatabaseName (DBN) 连接参数指定 utility_db（例如，DBN=utility_db），则会自动启动实用程序数据库。

用于 jConnect 连接的数据库选项集

应用程序使用 jConnect 驱动程序连接数据库时，将调用 sp_tsql_environment 存储过程：sp_tsql_environment 过程将一些数据库选项设置为与 Adaptive Server Enterprise 行为兼容。

从 JDBC 客户端应用程序连接

使用 SQL Anywhere JDBC 驱动程序时，数据库元数据始终可用。

要从使用 jConnect 的 JDBC 应用程序访问数据库系统表（数据库元数据），必须向数据库添加一组 jConnect 系统对象。这些过程会缺省安装到所有数据库。iqinit -i 选项会阻止此安装。

以下完整的 Java 应用程序是一个命令行程序，它连接到正在运行的数据库，在命令行中输出一组信息，然后终止。

对于任何 JDBC 应用程序，要使用数据库数据，第一步都必须建立连接。

此示例中所示的外部连接是常规的客户端/服务器连接。

连接示例代码

以下示例缺省使用 JDBC 4.0 版本的 SQL Anywhere JDBC 驱动程序连接到数据库。要使用不同的驱动程序，可以在命令行中传入驱动程序名称（jdbc4、jConnect）。使用 JDBC 4.0 驱动程序和 jConnect 的示例包含在代码中。本示例假设一数据库服务器已经使用示例数据库启动。源代码可在 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` 目录的 `JDBCConnect.java` 文件中找到。

```
import java.io.*;
import java.sql.*;

public class JDBCConnect
{
    public static void main( String args[] )
    {
        try
        {
            String arg;
            Connection con;

            // Select the JDBC driver and create a connection.
            // May throw a SQLException.
            // Choices are:
            // 1. jConnect driver
            // 2. SQL Anywhere JDBC 4.0 driver
            arg = "jdbc4";
            if( args.length > 0 ) arg = args[0];
            if( arg.compareToIgnoreCase( "jconnect" ) == 0 )
            {
                con = DriverManager.getConnection(
                    "jdbc:sybase:Tds:localhost:2638", "<user_id>",
"<password>");
            }
            else
            {
                con = DriverManager.getConnection(
```

```
                    "jdbc:sqlanywhere:uid=<user_id>;pwd=<password>" );
            }

            System.out.println("Using "+arg+" driver");

            // Create a statement object, the container for the SQL
            // statement. May throw a SQLException.
            Statement stmt = con.createStatement();

            // Create a result set object by executing the query.
            // May throw a SQLException.
            ResultSet rs = stmt.executeQuery(
                "SELECT ID, GivenName, Surname FROM Customers");

            // Process the result set.
            while (rs.next())
            {
                int value = rs.getInt(1);
                String FirstName = rs.getString(2);
                String LastName = rs.getString(3);
               System.out.println(value+" "+FirstName+" "+LastName);
            }
            rs.close();
            stmt.close();
            con.close();
        }
        catch (SQLException sqe)
        {
            System.out.println("Unexpected exception : " +
                               sqe.toString() + ", sqlstate = " +
                               sqe.getSQLState());
            System.exit(1);
        }
        catch (Exception e)
        {
            e.printStackTrace();
            System.exit(1);
        }

        System.exit(0);
    }
}
```

连接示例如何工作

外部连接示例是一个 Java 命令行程序。

导入包

该应用程序需要两个程序包，这两个程序包将在 `JDBCConnect.java` 的前几行中导入：

- java.io 包中含有输出到命令提示窗口时所需的 Java input/output 类。
- java.sql 包中包含所有 JDBC 应用程序都需要的 JDBC 类。

main 方法
每个 Java 应用程序都需要一个包含 main 方法的类，该方法是在程序启动时调用的方法。在以上的简单示例中，JDBCConnect.main 是应用程序中唯一的公共方法。

JDBCConnect.main 方法执行以下任务：

1. 根据命令行参数决定需要装载的驱动程序。如果 SQL Anywhere JDBC 4.0 和 jConnect 7.0 驱动程序在类文件路径下，启动时将自动装载它们。
2. 使用 SQL Anywhere JDBC 驱动程序 URL 与缺省的运行数据库建立连接。getConnection 方法使用指定的 URL 建立连接。
3. 创建包含 SQL 语句的语句对象。
4. 通过执行 SQL 查询创建结果集对象。
5. 迭代通过结果集，输出列信息。
6. 结束每个结果集、语句和连接对象。

运行连接示例

使用示例来显示创建和执行 JDBC 应用程序时所涉及的步骤。

前提条件

必须安装 Java 开发工具包（Java Development Kit，简称 JDK）。

过程

可以建立两种使用 JDBC 的不同类型连接。一种是客户端连接，另一种是服务器端连接。以下示例使用客户端连接。

1. 在命令提示符下，转到 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` 目录。
2. 使用本地计算机上的 iqdemo.db 数据库启动数据库服务器。
3. 设置 CLASSPATH 环境变量。本例使用了包含在 `sajdbc4.jar` 中的 SQL Anywhere JDBC 4.0 驱动程序。

   ```
   set classpath=.;%IQDIR%\java\sajdbc4.jar
   ```

   如果要改为使用 jConnect 驱动程序，则使用以下命令（其中 *path* 是 jConnect 的安装目录）：

   ```
   set classpath=.;jconnect-path\classes\jconn4.jar
   ```
4. 运行以下命令编译示例：

   ```
   javac JDBCConnect.java
   ```
5. 运行以下命令执行示例：

   ```
   java JDBCConnect
   ```

   添加命令行参数（如 `jconnect`）以装载不同的 JDBC 驱动程序。

```
java JDBCConnect jconnect
```

6. 确认命令提示符下出现一列带有客户名称的标识号。

   如果连接尝试失败，则显示的是错误消息。请确认是否已执行了所需的全部步骤。请检查类文件路径是否正确。不正确的设置可能会导致无法找到类。

显示一列带有客户名称的标识号。

如何从服务器端的 JDBC 类建立连接

JDBC 中的 SQL 语句是使用 Connection 对象的 createStatement 方法构建的。即使是在服务器内部运行的类，也需要建立连接以创建 Connection 对象。

从服务器端的 JDBC 类建立连接比建立外部连接更为直接。由于用户已经连接到数据库，所以类只使用当前连接。

服务器端连接示例代码

以下是服务器端连接示例的源代码。它是 JDBCConnect.java 示例的修改版本，位于 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC\JDBCConnect2.java 中。

```
import java.io.*;
import java.sql.*;

public class JDBCConnect2
{
  public static void main( String args[] )
  {
    try
    {
      // Open the connection. May throw a SQLException.
      Connection con = DriverManager.getConnection(
          "jdbc:default:connection" );

      // Create a statement object, the container for the SQL
      // statement. May throw a SQLException.
      Statement stmt = con.createStatement();
      // Create a result set object by executing the query.
      // May throw a SQLException.
      ResultSet rs = stmt.executeQuery(
        "SELECT ID, GivenName, Surname FROM Customers");

      // Process the result set.
      while (rs.next())
      {
        int value = rs.getInt(1);
        String FirstName = rs.getString(2);
        String LastName = rs.getString(3);
        System.out.println(value+" "+FirstName+" "+LastName);
      }
      rs.close();
```

```
      stmt.close();
      con.close();
    }
    catch (SQLException sqe)
    {
      System.out.println("Unexpected exception : " +
                sqe.toString() + ", sqlstate = " +
                sqe.getSQLState());
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

服务器端连接示例的不同之处

服务器端连接示例与客户端连接示例几乎是相同的，但有以下几个例外：

1. 不需要预装载 JDBC 驱动程序。
2. 它使用当前连接与缺省的运行数据库建立连接。已将 getConnection 调用中的 URL 更改为：

   ```
   Connection con = DriverManager.getConnection(
       "jdbc:default:connection" );
   ```
3. `System.exit()` 语句已删除。

运行服务器端连接示例

使用示例来显示创建和执行 JDBC 服务器端应用程序时所涉及的步骤。

前提条件

必须安装 Java 开发工具包（Java Development Kit，简称 JDK）。

过程

可以建立两种使用 JDBC 的不同类型连接。一种是客户端连接，另一种是服务器端连接。以下示例使用服务器端连接。

1. 在命令提示符下，转到 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` 目录。

   ```
   cd %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC
   ```
2. 对于服务器端 JDBC，除非服务器将从其它当前工作目录启动，否则无需设置 CLASSPATH 环境变量。

   ```
   set classpath=.;%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere
   \JDBC
   ```
3. 使用本地计算机上的 iqdemo 数据库启动数据库服务器。

4. 输入以下命令编译示例：

```
javac JDBCConnect2.java
```

5. 使用 Interactive SQL 将类安装到示例数据库。执行以下语句（可能需要类文件的路径）：

```
INSTALL JAVA NEW
FROM FILE 'JDBCConnect2.class';
```

6. 定义一个名为 JDBCConnect 的存储过程，该存储过程将充当类中 JDBCConnect2.main 方法的包装：

```
CREATE PROCEDURE JDBCConnect(OUT args LONG VARCHAR)
  EXTERNAL NAME 'JDBCConnect2.main([Ljava/lang/String;)V'
  LANGUAGE JAVA;
```

7. 如下所示调用 JDBCConnect2.main 方法：

```
CALL JDBCConnect();
```

   在会话中首次调用某个 Java 类时，必须装载 Java 虚拟机。这可能需要几秒钟的时间。

8. 确认在数据库服务器消息窗口中出现一列带有客户名称的标识号。

   如果连接尝试失败，则显示的是错误消息。请确认是否已执行了所需的全部步骤。

数据库服务器消息窗口中显示一列带有客户名称的标识号。

有关 JDBC 连接的说明

熟悉自动提交行为、事务隔离级别以及连接缺省值。

- **自动提交行为** – JDBC 规范要求在每个数据操作语句后缺省执行 COMMIT。当前，客户端 JDBC 行为是提交（自动提交为 true），服务器端行为是不提交（自动提交为 false）。要在客户端和服务器端应用程序中均获得相同的行为，您可以使用诸如以下的语句：

```
con.setAutoCommit( false );
```

  在此语句中，con 是当前连接对象。您也可以将自动提交设置为 true。
- **设置事务隔离级别** – 要设置事务隔离级别，应用程序必须使用以下值之一来调用 Connection.setTransactionIsolation 方法。

  SQL Anywhere JDBC 4.0 驱动程序：

  TRANSACTION_NONE
  TRANSACTION_READ_COMMITTED
  TRANSACTION_READ_UNCOMMITTED
  TRANSACTION_REPEATABLE_READ
  TRANSACTION_SERIALIZABLE
  sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT

sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_SNAPSHOT
sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_READONLY_SNAPSHOT

下例使用 JDBC 4.0 驱动程序将事务隔离级别设置为 SNAPSHOT。

```
try
{
    con.setTransactionIsolation(

sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT
    );
}
catch( Exception e )
{
    System.err.println( "Error! Could not set isolation level" );
    System.err.println( e.getMessage() );
    printExceptions( (SQLException)e );
}
```

有关 getTransactionIsolation 和 setTransactionIsolation 的详细信息，请参见关于 java.sql.Connection 接口的文档 (http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/)。

- **连接缺省值** – 从服务器端的 JDBC 对 `getConnection( "jdbc:default:connection" )` 的调用中，只有第一个调用使用缺省值创建新连接。后续调用返回当前连接的包装，所有连接属性均保持不变。如果在初始连接中将自动提交设置为 false，则同一 Java 代码中，任何后续 getConnection 调用所返回的连接中的自动提交均会设置为 false。

  您可能希望确保关闭连接时会使连接属性恢复为缺省值，这样，所获得的后续连接就会采用标准的 JDBC 值。以下代码可实现这一点：

```
Connection con =
    DriverManager.getConnection("jdbc:default:connection");

boolean oldAutoCommit = con.getAutoCommit();
try
{
     // main body of code here
}
finally
{
    con.setAutoCommit( oldAutoCommit );
}
```

  此处的讨论不仅适用于自动提交，也适用于其它连接属性，如事务隔离级别和只读模式。

  有关 getTransactionIsolation、setTransactionIsolation 和 isReadOnly 方法的详细信息，请参见关于 java.sql.Connection 接口的文档 (http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/)。

使用 JDBC 访问数据

相对于传统的 SQL 存储过程，用于保存数据库中的部分或全部类的 Java 应用程序具有更大的优点。不过，在入门阶段，最好并行使用 SQL 存储过程来证实 JDBC 的功能，这样或许会有所帮助。在下面的示例中，您编写了向 Departments 表中插入行的 Java 类。

与其它接口一样，JDBC 中的 SQL 语句可以为*静态*，也可以为*动态*。静态 SQL 语句在 Java 应用程序中构造，然后会发送到数据库。数据库服务器会分析该语句，选择执行计划，然后执行该语句。语法分析与选择执行计划统称为*准备*语句。

如果某条类似的语句必须被执行多次（例如，向一个表中插入多次），使用静态 SQL 会带来很大的开销，因为每次都必须执行准备步骤。

与此相反，动态 SQL 语句包含占位符。只需用这些占位符准备一次语句，就可以多次执行语句，省去了额外的准备开销。

准备 JDBC 示例

以下节中的代码段取自 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC\JDBCExample.java` 中的完整类。在准备这些节时编译示例 Java 应用程序并将其安装到数据库中。

前提条件

您必须具有 MANAGE ANY EXTERNAL OBJECT 系统特权。

必须安装 Java 开发工具包（Java Development Kit，简称 JDK）。

过程

1. 编译 `JDBCExample.java` 源代码。
2. 从 Interactive SQL 连接到数据库。
3. 通过在 Interactive SQL 中执行以下语句，将 `JDBCExample.class` 文件安装到示例数据库中：

   ```
   INSTALL JAVA NEW
   FROM FILE 'JDBCExample.class';
   ```

   如果数据库服务器不是从类文件所在的目录启动，而且数据库服务器的 CLASSPATH 中未列出类文件的路径，则必须将类文件路径包括在 INSTALL 语句中。

JDBCExample 类文件已安装到数据库中，随时都可以进行演示。

使用 JDBC 执行插入、更新和删除

静态 SQL 语句 INSERT、UPDATE 和 DELETE 等不返回结果集，使用 Statement 类的 executeUpdate 方法执行。诸如 CREATE TABLE 的语句及其它数据定义语句也可通过使用 executeUpdate 来执行。

也可以使用 Statement 类的 addBatch、clearBatch 和 executeBatch 方法。由于 JDBC 规范对于 Statement 类的 executeBatch 方法的行为没有明确规定，因此在使用 SQL Anywhere JDBC 驱动程序的情况下使用这种方法时应当考虑如下几点：

- 如果遇到 SQL 异常或结果集，批处理将立即终止。如果批处理停止，executeBatch 方法就会抛出 BatchUpdateException。在 BatchUpdateException 上调用 getUpdateCounts 方法会返回一个行计数整数数组。批处理失败之前的计数集会包含一个有效的非负更新计数；而在批处理失败点及其之后，计数集会包含一个 -1 值。将 BatchUpdateException 转换到 SQLException 时会提供有关批处理停止原因的更多详情。
- 只有在显式调用 clearBatch 方法时批处理才会被清除。因此，重复调用 executeBatch 方法将会导致再三地重复执行批处理。此外，调用 calling execute(sql_query) 或 executeQuery(sql_query) 能够正确地执行指定的 SQL 查询，但不能清除基础批处理。因此，调用 executeBatch 方法，接着调用 execute(sql_query)，然后再次调用 executeBatch 方法，将导致执行成批语句集，再执行指定的 SQL 查询，接下来再次执行成批语句集。

以下代码段说明如何执行 INSERT 语句。它将已传递到 InsertStatic 方法的 Statement 对象用作一个参数。

```
public static void InsertStatic( Statement stmt )
{
  try
  {
    int iRows = stmt.executeUpdate(
      "INSERT INTO Departments (DepartmentID, DepartmentName)"
      + " VALUES (201, 'Eastern Sales')" );
    // Print the number of rows inserted
    System.out.println(iRows + " rows inserted");
  }
  catch (SQLException sqe)
  {
    System.out.println("Unexpected exception : " +
              sqe.toString() + ", sqlstate = " +
              sqe.getSQLState());
  }
  catch (Exception e)
  {
    e.printStackTrace();
  }
}
```

注释

- 此代码段是 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere \JDBC` 目录中所包括的 `JDBCExample.java` 文件的一部分。
- executeUpdate 方法返回一个用于反映操作所影响到的行数的整数。在此情况下，成功的 INSERT 会返回值 1。
- 作为服务器端的类运行时，`System.out.println` 的输出会显示在数据库服务器消息窗口中。

通过 JDBC 使用静态 INSERT 和 DELETE 语句

使用静态 SQL 语句从数据库服务器调用示例 JDBC 应用程序，以在 Departments 表中插入和删除行。

前提条件

要创建外部过程，必须具有 CREATE PROCEDURE 和 CREATE EXTERNAL REFERENCE 系统特权。还必须对要修改的数据库对象具有 SELECT、DELETE 和 INSERT 特权。

必须安装 Java 开发工具包（Java Development Kit，简称 JDK）。

过程

1. 从 Interactive SQL 连接到数据库。
2. 确保已安装 JDBCExample 类。
3. 定义一个名为 JDBCExample 的存储过程，该存储过程充当类中 JDBCExample.main 方法的包装：

   ```
   CREATE PROCEDURE JDBCExample(IN arg CHAR(50))
     EXTERNAL NAME 'JDBCExample.main([Ljava/lang/String;)V'
     LANGUAGE JAVA;
   ```

4. 如下所示调用 JDBCExample.main 方法：

   ```
   CALL JDBCExample( 'insert' );
   ```

   参数字符串 `'insert'` 用以调用 InsertStatic 方法。

5. 确认 Departments 表中已添加一行。

   ```
   SELECT * FROM Departments;
   ```

   示例程序会在数据库服务器消息窗口中显示 Departments 表的更新内容。

6. 在示例类中有一个称为 DeleteStatic 的类似方法，用于显示如何删除刚刚添加的行。如下所示调用 JDBCExample.main 方法：

   ```
   CALL JDBCExample( 'delete' );
   ```

   参数字符串 `'delete'` 用以调用 DeleteStatic 方法。

7. 确认该行已从 Departments 表中删除。

```
SELECT * FROM Departments;
```

示例程序会在数据库服务器消息窗口中显示 Departments 表的更新内容。
已在服务器端 JDBC 应用程序中使用静态 SQL 语句从表中插入和删除行。

如何使用预准备语句进行更有效的访问

如果使用 Statement 接口，则先对您发送到数据库的每条语句进行分析，生成访问计划，然后再执行该语句。这些执行语句之前的步骤称为*准备*语句。

如果使用 PreparedStatement 接口，将会在性能方面获得一些益处。这样您就可以使用占位符来准备语句，然后在执行语句时向占位符赋值。

使用预准备语句在执行多个类似的操作（如插入多行）时特别有用。

示例

以下示例说明如何使用 PreparedStatement 接口，不过，插入单行并未有效地利用了预准备语句。

JDBCExample 类的以下 InsertDynamic 方法用于执行一个预准备语句：

```
public static void InsertDynamic( Connection con,
              String ID, String name )
{
  try
  {
    // Build the INSERT statement
    // ? is a placeholder character
    String sqlStr = "INSERT INTO Departments " +
            "( DepartmentID, DepartmentName ) " +
            "VALUES ( ? , ? )";

    // Prepare the statement
    PreparedStatement stmt =
      con.prepareStatement( sqlStr );

    // Set some values
    int idValue = Integer.valueOf( ID );
    stmt.setInt( 1, idValue );
    stmt.setString( 2, name );

    // Execute the statement
    int iRows = stmt.executeUpdate();

    // Print the number of rows inserted
    System.out.println(iRows + " rows inserted");
  }
  catch (SQLException sqe)
  {
    System.out.println("Unexpected exception : " +
              sqe.toString() + ", sqlstate = " +
              sqe.getSQLState());
  }
```

```
  catch (Exception e)
  {
    e.printStackTrace();
  }
}
```

注释

- 此代码段是 JDBCExample.java\SQLAnywhere\JDBC%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC 目录中所包括的 JDBCExample Java 文件的一部分。
- executeUpdate 方法返回一个用于反映操作所影响到的行数的整数。在此情况下，成功的 INSERT 会返回值 1。
- 作为服务器端的类运行时，System.out.println 的输出会显示在数据库服务器消息窗口中。

通过 JDBC 使用预准备的 INSERT 和 DELETE 语句

使用预准备的语句从数据库服务器调用示例 JDBC 应用程序，以在 Departments 表中插入和删除行。

前提条件

要创建外部过程，必须具有 CREATE PROCEDURE 和 CREATE EXTERNAL REFERENCE 系统特权。还必须对要修改的数据库对象具有 SELECT、DELETE 和 INSERT 特权。

必须安装 Java 开发工具包（Java Development Kit，简称 JDK）。

过程

1. 从 Interactive SQL 连接到数据库。
2. 确保已安装 JDBCExample 类。
3. 定义一个名为 JDBCInsert 的存储过程，该存储过程充当类中 JDBCExample.Insert 方法的包装：

   ```
   CREATE PROCEDURE JDBCInsert(IN arg1 INTEGER, IN arg2 CHAR(50))
     EXTERNAL NAME 'JDBCExample.Insert(ILjava/lang/String;)V'
     LANGUAGE JAVA;
   ```
4. 如下所示调用 JDBCExample.Insert 方法：

   ```
   CALL JDBCInsert( 202, 'Southeastern Sales' );
   ```

   Insert 方法用以调用 InsertDynamic 方法。
5. 确认 Departments 表中已添加一行。

   ```
   SELECT * FROM Departments;
   ```

   示例程序会在数据库服务器消息窗口中显示 Departments 表的更新内容。

6. 在示例类中有一个称为 DeleteDynamic 的类似方法，用于显示如何删除刚刚添加的行。

   定义一个名为 JDBCDelete 的存储过程，该存储过程充当类中 JDBCExample.Delete 方法的包装：

   ```
   CREATE PROCEDURE JDBCDelete(IN arg1 INTEGER)
     EXTERNAL NAME 'JDBCExample.Delete(I)V'
     LANGUAGE JAVA;
   ```

7. 如下所示调用 JDBCExample.Delete 方法：

   ```
   CALL JDBCDelete( 202 );
   ```

   Delete 方法用以调用 DeleteDynamic 方法。

8. 确认该行已从 Departments 表中删除。

   ```
   SELECT * FROM Departments;
   ```

   示例程序会在数据库服务器消息窗口中显示 Departments 表的更新内容。

已在服务器端 JDBC 应用程序中使用预准备的 SQL 语句从表中插入和删除行。

JDBC 批处理方法

PreparedStatement 类的 addBatch 方法用于执行成批插入（或宽插入）。以下是使用此方法的一些指导。

1. INSERT 语句应该使用 Connection 类中的 prepareStatement 方法之一来预准备。

   ```
   // Build the INSERT statement
   String sqlStr = "INSERT INTO Departments " +
                   "( DepartmentID, DepartmentName ) " +
                   "VALUES ( ? , ? )";
   // Prepare the statement
   PreparedStatement stmt =
        con.prepareStatement( sqlStr );
   ```

2. 应设置准备的插入语句的参数并对其进行批处理，如下所示：

   ```
   // loop to batch "n" sets of parameters
   for( i=0; i < n; i++ )
   {
       // "stmt" is the original prepared insert statement from step
   1.
       stmt.setSomeType( 1, param_1 );
       stmt.setSomeType( 2, param_2 );
       .
       .
       .
       // There are "m" parameters in the statement.
       stmt.setSomeType( m , param_m );

       // Add the set of parameters to the batch and
       // move to the next row of parameters.
       stmt.addBatch();
   }
   ```

   示例：

```
for( i=0; i < 5; i++ )
{
    stmt.setInt( 1, idValue );
    stmt.setString( 2, name );
    stmt.addBatch();
}
```

3. 批处理必须用 PreparedStatement 类的 executeBatch 方法执行。

批处理中不支持 BLOB 参数。

使用 SQL Anywhere JDBC 驱动程序执行成批插入时，建议使用较小的列大小。建议不要使用成批插入方式向 long binary 或 long varchar 列中插入大的二进制或字符数据，这样可能会使性能下降。性能会下降的原因是 SQL Anywhere JDBC 驱动程序只有分配较大的内存大小才能保存成批插入的各行。在所有其它情况下，与逐个插入相比，成批插入会提供更佳的性能。

如何从 Java 返回结果集

本节介绍如何从 Java 方法获得一个或多个结果集。

您必须编写一个用于向调用环境返回一个或多个结果集的 Java 方法，并将此方法包装在 SQL 存储过程中。以下代码段说明如何才能向此 Java 过程的调用程序返回多个结果集。它使用三个 executeQuery 语句获得三个不同的结果集。

```
public static void Results( ResultSet[] rset )
       throws SQLException
{
    // Demonstrate returning multiple result sets

    Connection con = DriverManager.getConnection(
                     "jdbc:default:connection" );
    rset[0] = con.createStatement().executeQuery(
        "SELECT * FROM Employees" +
        "   ORDER BY EmployeeID" );
    rset[1] = con.createStatement().executeQuery(
        "SELECT * FROM Departments" +
        "   ORDER BY DepartmentID" );
    rset[2] = con.createStatement().executeQuery(
        "SELECT i.ID,i.LineID,i.ProductID,i.Quantity," +
        "       s.OrderDate,i.ShipDate," +
        "       s.Region,e.GivenName||' '||e.Surname" +
        "   FROM SalesOrderItems AS i" +
        "   JOIN SalesOrders AS s" +
        "   JOIN Employees AS e" +
        "   WHERE s.ID=i.ID" +
        "         AND s.SalesRepresentative=e.EmployeeID" );
    con.close();
}
```

注释

- 服务器端 JDBC 示例是 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\JDBC` 目录中包括的 `JDBCExample.java` 文件的一部分。

- 它包含使用 getConnection 与缺省运行数据库的连接。
- executeQuery 方法返回结果集。

从 JDBC 返回结果集

从数据库服务器调用示例 JDBC 应用程序，以返回多个结果集。

前提条件

必须安装 Java 开发工具包（Java Development Kit，简称 JDK）。

过程

1. 从 Interactive SQL 连接到数据库。
2. 确保已安装 JDBCExample 类。
3. 定义一个名为 JDBCResults 的存储过程，该存储过程充当类中 JDBCExample.Results 方法的包装。

   例如：

   ```
   CREATE PROCEDURE JDBCResults(OUT args LONG VARCHAR)
     DYNAMIC RESULT SETS 3
     EXTERNAL NAME 'JDBCExample.Results([Ljava/sql/ResultSet;)V'
     LANGUAGE JAVA;
   ```

   该示例返回 3 个结果集。
4. 设置以下 Interactive SQL 选项，以便您可以查看查询的所有结果：
   a. 单击**“工具”**»**“选项”**。
   b. 单击**“Sybase IQ”**。
   c. 单击**“结果”**选项卡。
   d. 将**“要显示的最大行数”**的值设置为 5000。
   e. 单击**“显示所有结果集”**。
   f. 单击**“确定”**。
5. 调用 JDBCExample.Results 方法。

   ```
   CALL JDBCResults();
   ```
6. 分别检查以下这三个结果选项卡：**“结果集 1”**、**“结果集 2”**和**“结果集 3”**。

从服务器端 JDBC 应用程序返回三个不同的结果集。

JDBC 说明

了解访问和执行 Java 类的特权。

- **访问特权** – 如同数据库中的所有 Java 类一样，包含有 JDBC 语句的类可由任何用户来访问，只要 GRANT EXECUTE 语句已授予它们用以执行充当 Java 方法包装的存储过程的特权。

- **执行特权** – 执行 Java 类时需要具有执行这些类的连接特权。此行为与存储过程的行为不同，执行存储过程需要具有所有者特权。

JDBC 回调

SQL Anywhere JDBC 驱动程序支持两个异步回调，一个用来处理 SQL MESSAGE 语句，另外一个用于校验文件传输请求。

可使用 SQL MESSAGE 语句将消息从数据库服务器发送到客户端应用程序。长时间运行的数据库服务器语句也可以生成消息。

可创建消息处理程序例程来拦截这些消息。以下是一个消息处理程序回调例程的示例：

```
class T_message_handler implements
sybase.jdbc4.sqlanywhere.ASAMessageHandler
{
    private final int MSG_INFO      = 0x80 | 0;
    private final int MSG_WARNING   = 0x80 | 1;
    private final int MSG_ACTION    = 0x80 | 2;
    private final int MSG_STATUS    = 0x80 | 3;
    T_message_handler()
    {
    }

    public SQLException messageHandler(SQLException sqe)
    {
        String msg_type = "unknown";

        switch( sqe.getErrorCode() ) {
            case MSG_INFO:      msg_type = "INFO   ";   break;
            case MSG_WARNING:   msg_type = "WARNING";   break;
            case MSG_ACTION:    msg_type = "ACTION ";   break;
            case MSG_STATUS:    msg_type = "STATUS ";   break;
        }

        System.out.println( msg_type + ": " + sqe.getMessage() );
        return sqe;
    }
}
```

可以校验客户端文件传输请求。在允许进行任何传输前，JDBC 驱动程序会调用校验回调函数（如果存在）。如果在执行间接语句（从存储过程内部）期间，请求进行客户端数据传输，则除非客户端应用程序注册了校验回调函数，否则 JDBC 驱动程序将不允许进行传输。下面更详尽地介绍了进行校验调用的条件。下面是一个文件传输校验回调例程的示例。

```
class T_filetrans_callback implements
sybase.jdbc4.sqlanywhere.SAValidateFileTransferCallback
{
    T_filetrans_callback()
    {
```

```
    }

    public int callback(String filename, int is_write)
    {
        System.out.println( "File transfer granted for file " +
filename +
                            " with an is_write value of " +
is_write );
        return( 1 ); // 0 to disallow, non-zero to allow
    }
}
```

filename 参数是要读取或写入的文件的名称。如果请求读取（从客户端传输到服务器），则 is_write 参数为 0；如果请求写入，则该参数为非零值。如果不允许进行文件传输，则回调函数应返回 0，否则返回非零值。

为确保数据安全，服务器会跟踪请求文件传输的语句的源。服务器会确定语句是否是从客户端应用程序直接接收的。从客户端启动数据传输时，服务器会将语句源的相关信息发送到客户端软件。对于 JDBC 驱动程序而言，仅当是由于执行客户端应用程序直接发送的语句而请求数据传输时，它才会允许无条件传输数据。否则，应用程序必须注册上文所述的校验回调函数，如果未注册该函数，则传输会被拒绝，而且语句失败并出现一个错误。如果客户端语句调用的某个存储过程在数据库中已经存在，则对该存储过程本身的执行不被视为是客户端启动的语句所完成的。但是，如果客户端应用程序显式地创建一个临时存储过程，则服务器会将对该存储过程的执行视为是由客户端启动的。同样，如果客户端应用程序执行一个批处理语句，则该批处理语句的执行被视为是直接由客户端应用程序完成的。

下面的 Java 应用程序示例演示了如何使用 SQL Anywhere JDBC 4.0 驱动程序支持的回调。需要将 `%ALLUSERSPROFILE%\SybaseIQ\samples\java\sajdbc4.jar` 文件放入类路径中。

```
import java.io.*;
import java.sql.*;
import java.util.*;

public class callback
{
    public static void main (String args[]) throws IOException
    {
        Connection          con = null;
        Statement           stmt;

        System.out.println ( "Starting... " );
        con = connect();
        if( con == null )
        {
            return; // exception should already have been reported
        }
        System.out.println ( "Connected... " );
        try
        {
            // create and register message handler callback
            T_message_handler message_worker = new
```

```
T_message_handler();

((sybase.jdbc4.sqlanywhere.IConnection)con).setASAMessageHandler( m
essage_worker );

            // create and register validate file transfer callback
            T_filetrans_callback filetran_worker = new
T_filetrans_callback();

((sybase.jdbc4.sqlanywhere.IConnection)con).setSAValidateFileTransf
erCallback( filetran_worker );

            stmt = con.createStatement();

          // execute message statements to force message handler to
be called
            stmt.execute( "MESSAGE 'this is an info   message' TYPE
INFO TO CLIENT" );
            stmt.execute( "MESSAGE 'this is an action message' TYPE
ACTION TO CLIENT" );
            stmt.execute( "MESSAGE 'this is a warning message' TYPE
WARNING TO CLIENT" );
            stmt.execute( "MESSAGE 'this is a status  message' TYPE
STATUS TO CLIENT" );

            System.out.println( "\n==================\n" );

            stmt.execute( "set temporary option
allow_read_client_file='on'" );
            try
            {
              stmt.execute( "drop procedure read_client_file_test" );
            }
            catch( SQLException dummy )
            {
                // ignore exception if procedure does not exist
            }
          // create procedure that will force file transfer callback
to be called
          stmt.execute( "create procedure read_client_file_test()" +
                        "begin" +
                        "    declare v long binary;" +
                    "    set v = read_client_file('sample.txt');" +
                        "end" );

          // call procedure to force validate file transfer callback
to be called
            try
            {
                stmt.execute( "call read_client_file_test()" );
            }
            catch( SQLException filetrans_exception )
            {
               // Note: Since the file transfer callback returns 1,
                // do not expect a SQL exception to be thrown
                System.out.println( "SQLException: " +
```

```
                                    filetrans_exception.getMessage() );
            }
            stmt.close();
            con.close();
            System.out.println( "Disconnected" );
        }
        catch( SQLException sqe )
        {
            printExceptions(sqe);
        }
    }

    private static Connection connect()
    {
        Connection  connection;

        System.out.println( "Using jdbc4 driver"  );
        try
        {
            connection = DriverManager.getConnection(
                    "jdbc:sqlanywhere:uid=DBA;pwd=sql" );
        }
        catch( Exception e )
        {
            System.err.println( "Error! Could not connect" );
            System.err.println( e.getMessage() );
            printExceptions( (SQLException)e );
            connection = null;
        }
        return connection;
    }

    static private void printExceptions(SQLException sqe)
    {
        while (sqe != null)
        {

            System.out.println("Unexpected exception : " +
                "SqlState: " + sqe.getSQLState()  +
                " " + sqe.toString() +
                ", ErrorCode: " + sqe.getErrorCode());
            System.out.println( "==================\n" );
            sqe = sqe.getNextException();
        }
    }
}
```

JDBC 转义语法

您可以在任何 JDBC 应用程序（包括 Interactive SQL）中使用 JDBC 转义语法。此转义语法允许您调用存储过程而不管您正在使用哪种数据库管理系统。转义语法的一般格式为

{ `keyword` *parameters* }

这组关键字包括：

- **{d date-string}** – 日期字符串是 SAP Sybase IQ 接受的任意日期值。
- **{t time-string}** – 时间字符串是 SAP Sybase IQ 接受的任意时间值。
- **{ts date-string time-string}** – 日期/时间字符串是 SAP Sybase IQ 接受的任意时间戳值。
- **{guid uuid-string}** – uuid-string 是任意有效的 GUID 字符串，例如 41dfe9ef-db91-11d2-8c43-006008d26a6f。
- **{oj outer-join-expr}** – outer-join-expr 是 SAP Sybase IQ 接受的有效 OUTER JOIN 表达式。
- **{? = call func(p1,...)}** – 此函数是 SAP Sybase IQ 接受的任意有效函数调用。
- **{call proc(p1,...)}** – 此过程是 SAP Sybase IQ 接受的任意有效存储过程调用。
- **{fn func(p1,...)}** – 此函数为下文所列函数库中的任一函数。

您可以使用转义语法访问由 JDBC 驱动程序实现的函数库，这些函数包括数字、字符串、时间、日期和系统函数。

例如，要以与数据库管理系统无关的方式获得当前日期，您需要执行以下语句：

```
SELECT { FN CURDATE() }
```

可用的函数取决于您正在使用的 JDBC 驱动程序。以下两个表分别列出了 SQL Anywhere JDBC 驱动程序和 jConnect 驱动程序支持的函数。

SQL Anywhere JDBC 驱动程序支持的函数

数字函数	字符串函数	系统函数	时间/日期函数
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH		CURRENT_TIMESTAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME

数字函数	字符串函数	系统函数	时间/日期函数
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOUR
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		WEEK
ROUND	RTRIM		YEAR
SIGN	SOUNDEX		
SIN	SPACE		
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

jConnect 支持的函数

数字函数	字符串函数	系统函数	时间/日期函数
ABS	ASCII	DATABASE	CURDATE
ACOS	CHAR	IFNULL	CURTIME
ASIN	CONCAT	USER	DAYNAME
ATAN	DIFFERENCE	CONVERT	DAYOFMONTH
ATAN2	LCASE		DAYOFWEEK
CEILING	LENGTH		HOUR

数字函数	字符串函数	系统函数	时间/日期函数
COS	REPEAT		MINUTE
COT	RIGHT		MONTH
DEGREES	SOUNDEX		MONTHNAME
EXP	SPACE		NOW
FLOOR	SUBSTRING		QUARTER
LOG	UCASE		SECOND
LOG10			TIMESTAMPADD
PI			TIMESTAMPDIFF
POWER			YEAR
RADIANS			
RAND			
ROUND			
SIGN			
SIN			
SQRT			
TAN			

使用转义语法的语句应该可以用于 SAP Sybase IQ、Adaptive Server Enterprise、Oracle、SQL Server 或连接到的其它数据库管理系统。

在 Interactive SQL 中，大括号必须成对使用。连续的括号之间不得有空格：允许使用 "{{"，但不允许使用 "{ {"。同样，您不能在语句中使用新行字符。由于转义语法不是由 Interactive SQL 解析，所以在存储过程中不能使用转义语法。

例如，要使用 SQL 转义语法通过 sa_db_info 过程获得数据库属性，您将在 Interactive SQL 中执行以下语句：

```
{{CALL sa_db_info( 0 ) }}
```

JDBC 4.0 API 支持

SQL Anywhere JDBC 驱动程序支持 JDBC 4.0 规范的所有必需的类和方法。不支持 java.sql.Blob 接口的一些可选方法。这些可选的方法有:

```
long position( Blob pattern, long start );
long position( byte[] pattern, long start );
OutputStream setBinaryStream( long pos )
int setBytes( long pos, byte[] bytes )
int setBytes( long pos, byte[] bytes, int offset, int len );
void truncate( long len );
```

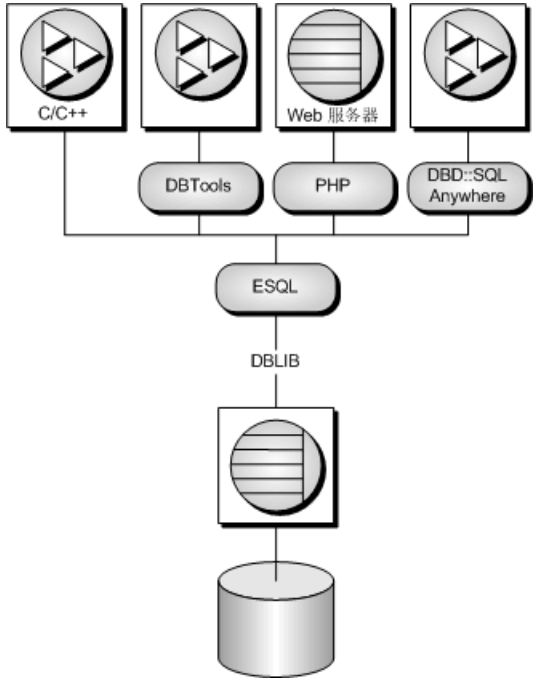
嵌入式 SQL

嵌入在 C 或 C++ 资源文件中的 SQL 语句称为嵌入式 SQL。预处理器将这些语句翻译为对运行时库的调用。嵌入式 SQL 是一种 ISO/ANSI 和 IBM 标准。

嵌入式 SQL 能够向其它数据库和其它环境移植，并且它在所有操作环境中的功能是等同的。它是一个综合的低层接口，可提供某种数据库产品的所有可用功能。使用嵌入式 SQL 需要您了解 C 或 C++ 编程语言。

嵌入式 SQL 应用程序

您可以开发使用 SAP Sybase IQ 嵌入式 SQL 接口访问 SAP Sybase IQ 服务器的 C 或 C++ 应用程序。例如，命令行数据库工具就是以此方式开发的应用程序。



嵌入式 SQL 是用于 C 和 C++ 编程语言的数据库编程接口。它由混杂在（嵌入于） C 或 C++ 源代码中的 SQL 语句组成。这些 SQL 语句先由 SQL 预处理器转换为 C 或 C++ 源代码，然后您可以编译这些源代码。

在运行时，嵌入式 SQL 应用程序使用名为 DBLIB 的 SAP Sybase IQ *接口库* 与数据库服务器通信。在大多数平台上，DBLIB 是一个动态链接库 (DLL) 或共享对象。

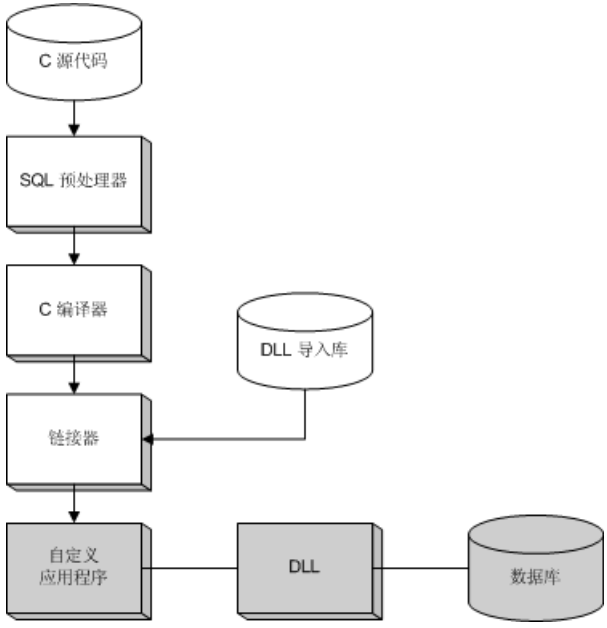
- 在 Windows 操作系统上，接口库是 `dblib16.dll`。

- 在 Unix 操作系统上，接口库为 `libdblib16.so`、`libdblib16.sl`，或 `libdblib16.a`，具体视操作系统而定。
- 在 Mac OS X 上，接口库为 `libdblib16.dylib.1`。

SAP Sybase IQ 提供了两种嵌入式 SQL。静态嵌入式 SQL 使用起来比较简单，但它不如动态嵌入式 SQL 灵活。

开发过程概述

嵌入式 SQL 开发过程概述。



在对程序成功进行预处理和编译后，就可以将其与 DBLIB 的 *导入库* 链接在一起，形成可执行文件。在运行数据库服务器时，这个可执行文件使用 DBLIB 与数据库服务器交互。在对程序进行预处理时，不必运行数据库服务器。

在 Windows 上，有适用于 Microsoft Visual C++ 的 32 位 和 64 位导入库。通过使用导入库可以开发调用 DLL 中函数的应用程序。但是，建议以动态方式加载库，以避免使用导入库。

SQL 预处理器

SQL 预处理器是一个名为 `iqiqsqlpp` 的可执行文件。

预处理器命令行如下：

```
iqsqlpp [ options ] sql-filename [ output-filename ]
```

预处理器将 C 或 C++ 源文件中的嵌入式 SQL 语句转换为 C 代码，并将结果置于一个输出文件中。然后，将使用 C 或 C++ 编译器对输出文件进行处理。含有嵌入式 SQL 的源程序的扩展名通常为 `.sqc`。缺省的输出文件名是 *sql-filename*，扩展名为 `.c`。如果 *sql-filename* 具有 `.c` 扩展名，则缺省输出文件名的扩展名将更改为 `.cc`。

注意： 如果重建应用程序以使用新的主版本数据库接口库，则必须使用相同版本的 SQL 预处理器对嵌入式 SQL 文件进行预处理。

下表描述了各种预处理选项。

选项	说明
`-d`	生成减小数据空间大小的代码。数据结构在使用之前执行时会得到重用和初始化。这会增大代码的大小。
`-e` *level*	将任何未包含在指定标准中的静态嵌入式 SQL 标记为错误。*level* 值表示要使用的标准。例如，`iqsqlpp -e c03 ...` 标记任何未包含在核心 SQL/2008 标准中的语法。支持的 *level* 值为： • **c08** – 标记不是核心 SQL/2008 语法的语法 • **p08** – 标记非完整 SQL/2008 语法的语法 • **c03** – 标记不是核心 SQL/2003 语法的语法 • **p03** – 标记非完整 SQL/2003 语法的语法 • **c99** – 标记不是核心 SQL/1999 语法的语法 • **p99** – 标记非完整 SQL/1999 语法的语法 • **e92** – 标记非入门级 SQL/1992 语法的语法 • **i92** – 标记非中级 SQL/1992 语法的语法 • **f92** – 标记非完整 SQL/1992 语法的语法 • **t** – 标记非标准主机变量类型 • **u** – 标记 UltraLite 不支持的语法 为了与以前的 SAP Sybase IQ 版本兼容，也可指定 e、i 和 f（分别对应 e92、i92 和 f92）。
`-h` *width*	将 iqsqlpp 输出的最大行长度限制为 *width*。续行符是反斜杠 (\)，而 *width* 的最小值是 10。
`-k`	通知预处理器要编译的程序包括 SQLCODE 的用户声明。定义必须是长整型，但不必在声明部分内。

选项	说明
-m *mode*	如果未在嵌入式 SQL 应用程序中显式地指定游标的可更新性模式，则指定游标的可更新性模式。*mode* 可以是以下各项之一： • **HISTORICAL** – 在以前的版本中，嵌入式 SQL 游标在缺省情况下为 FOR UPDATE 或 READ ONLY（取决于查询和 ansi_update_constraints 选项值）。显式游标可更新性在 DECLARE CURSOR 中指定。使用该选项可保留此行为。 • **READONLY** – 嵌入式 SQL 游标缺省设置为 READ ONLY。显式游标可更新性在 PREPARE 上指定。这是缺省行为。READ ONLY 游标可提高性能。
-n	在 C 文件中生成行号信息。此信息包括生成的 C 代码中适当位置处的 *#line* 指令。如果您使用的编译器支持 *#line* 指令，使用此选项可使编译器按照 SQC 文件（其中带有嵌入式 SQL）的行号报错，而不是用 SQL 预处理器生成的 C 文件的行号报错。此外，*#line* 指令由源代码级调试工具间接使用，以便您可以在查看 SQC 源文件时进行调试。
-o *operating-system*	指定目标操作系统。支持的操作系统有： • **WINDOWS** – Microsoft Windows。 • **UNIX** – 当创建 32 位 Unix 应用程序时可使用此选项。 • **UNIX64** – 当创建 64 位 Unix 应用程序时可使用此选项。
-q	安静模式 - 不显示消息。
-r-	生成非重入代码。
-s *len*	设置预处理器放入 C 文件的最大大小的字符串。长度大于此值的字符串将通过使用一组字符（*'a'*、*'b'*、*'c'* 等）进行初始化。大多数 C 编译器都对可以处理的字符串大小有限制。此选项用于设置其上限。缺省值是 500。
-u	为 UltraLite 生成代码。

选项	说明
`-w` *level*	将任何未包含在指定标准中的静态嵌入式 SQL 标记为警告。*level* 值表示要使用的标准。例如，`iqsqlpp -w c08 ...` 标记任何未包含在核心 SQL/2008 语法中的 SQL 语法。支持的 *level* 值为： • **c08** – 标记不是核心 SQL/2008 语法的语法 • **p08** – 标记非完整 SQL/2008 语法的语法 • **c03** – 标记不是核心 SQL/2003 语法的语法 • **p03** – 标记非完整 SQL/2003 语法的语法 • **c99** – 标记不是核心 SQL/1999 语法的语法 • **p99** – 标记非完整 SQL/1999 语法的语法 • **e92** – 标记非入门级 SQL/1992 语法的语法 • **i92** – 标记非中级 SQL/1992 语法的语法 • **f92** – 标记非完整 SQL/1992 语法的语法 • **t** – 标记非标准主机变量类型 • **u** – 标记 UltraLite 不支持的语法 为了与以前的 SAP Sybase IQ 版本兼容，也可指定 e、i 和 f（分别对应 e92、i92 和 f92）。
`-x`	将多字节字符串更改为转义序列，以便它们可以通过编译器。
`-z` *cs*	指定归类序列。要查看建议使用的归类序列的列表，请在命令提示符处运行 `iqinit -l`。 归类序列用于帮助预处理器理解在程序源代码中使用的字符，例如，识别出适合在标识符中使用的字母字符。如果未指定 -z，则预处理器将尝试根据操作系统和 SALANG 与 SACHARSETE 环境变量确定要使用的合理归类。
sql-filename	要处理的包含嵌入式 SQL 的 C 或 C++ 程序。
output-filename	由 SQL 预处理器创建的 C 语言源文件。

支持的编译器

C 语言 SQL 预处理器已经可以与下列编译器联合使用：

操作系统	编译器	版本
Windows	Microsoft Visual C++	6.0 或更高版本
Windows Mobile	Microsoft Visual C++	2005

操作系统	编译器	版本
Unix	GNU 编译器或本地编译器	

嵌入式 SQL 头文件

所有头文件均安装在 SAP Sybase IQ 安装目录的 `SDK\Include` 子目录中。

文件名	说明
`sqlca.h`	包括在所有嵌入式 SQL 程序中的主头文件。此文件包括 SQL 通信区域 (SQLCA) 的结构定义和所有嵌入式 SQL 数据库接口函数的原型。
`sqlda.h`	包括在使用动态 SQL 的嵌入式 SQL 程序中的 SQL 描述符区域结构定义。
`sqldef.h`	嵌入式 SQL 接口数据类型的定义。此文件还包含从 C 程序启动数据库服务器所需的结构定义和返回代码。
`sqlerr.h`	在 SQLCA 的 sqlcode 字段中返回的错误代码的定义。
`sqlstate.h`	在 SQLCA 的 sqlstate 字段中返回的 ANSI/ISO SQL 标准错误状态的定义。
`pshpk1.h`, `pshpk4.h`, `poppk.h`	这些头文件可确保正确地处理结构压缩。

导入库

在 Windows 平台上，所有导入库均安装在 SAP Sybase IQ 安装目录的 `SDK\Lib` 子目录下。Windows 导入库存储在 `SDK\Lib\x86` 和 `SDK\Lib\x64` 子目录中。导出定义列表存储在 `SDK\Lib\Def\dblib.def` 中。

在 Unix 平台上，所有导入库均安装在 SAP Sybase IQ 安装目录的 `lib32` 和 `lib64` 子目录下。

操作系统	编译器	导入库
Windows	Microsoft Visual C++	`dblibtm.lib`
Unix（非线程应用程序）	所有编译器	`libdblib16.so`, `libdbtasks16.so`, `libdblib16.sl`, `libdbtasks16.sl`

操作系统	编译器	导入库
Unix（线程应用程序）	所有编译器	libdblib16_r.so, libdbtasks16_r.so,libdblib16_r.sl, libdbtasks16_r.sl

libdbtasks16 库由 libdblib16 库调用。有些编译器会自动查找 libdbtasks16。对于其它编译器，则需要您显式指定 libdbtasks11。

示例嵌入式 SQL 程序

下面是一个非常简单的嵌入式 SQL 程序示例。

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main()
{
  db_init( &sqlca );
  EXEC SQL WHENEVER SQLERROR GOTO error;
  EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
  EXEC SQL UPDATE Employees
    SET Surname =  'Plankton'
    WHERE EmployeeID = 195;
  EXEC SQL COMMIT WORK;
  EXEC SQL DISCONNECT;
  db_fini( &sqlca );
  return( 0 );
error:
  printf( "update unsuccessful -- sqlcode = %ld\n",
    sqlca.sqlcode );
  db_fini( &sqlca );
  return( -1 );
}
```

本示例连接到数据库，更新 195 号雇员的姓氏，提交更改，然后退出。嵌入式 SQL 代码和 C 代码之间几乎没有交互作用。在本示例中 C 代码仅用于控制流。WHENEVER 语句用于错误检查。错误处理（此示例中的 GOTO）会在任何引起错误的 SQL 语句之后执行。

嵌入式 SQL 程序的结构

SQL 语句置于（嵌入）常规 C 或 C++ 代码内。所有嵌入式 SQL 语句都以 EXEC SQL 开头，并以分号 (;) 结尾。在嵌入式 SQL 语句的中间允许使用常规 C 语言注释。

使用嵌入式 SQL 的每个 C 程序都必须在源文件中任何其它嵌入式 SQL 语句之前包含以下语句。

```
EXEC SQL INCLUDE SQLCA;
```

使用嵌入式 SQL 的每个 C 程序都必须先初始化一个 SQLCA：

```
db_init( &sqlca );
```

C 程序所执行的前几个嵌入式 SQL 语句之一必须是 CONNECT 语句。CONNECT 语句用于建立与数据库服务器的连接，以及指定连接期间用于授权执行的所有语句的用户 ID。

有些嵌入式 SQL 语句不生成任何 C 代码，或不涉及与数据库的通信。因此，允许在 CONNECT 语句之前使用这些语句。最主要的是 INCLUDE 语句和指定错误处理方法的 WHENEVER 语句。

使用嵌入式 SQL 的每个 C 程序都必须完成任何已初始化的 SQLCA。

```
db_fini( &sqlca );
```

在 Windows 中动态装载 DBLIB

使用安装目录的 `SDK\C` 子目录中的 `esqldll.c` 模块，从嵌入式 SQL 应用程序中动态装载 DBLIB，这样就无需链接到导入库。

前提条件

执行此任务没有前提条件。

过程

常用方法是将应用程序链接到包含所需函数定义的动态链接库 (DLL) 的静态导入库，而此任务是这一方法的替代方法。

可使用类似任务在 Unix 平台上动态装载 DBLIB。

1. 应用程序必须调用 db_init_dll 才能装载 DBLIB DLL，而且必须调用 db_fini_dll 才能释放 DBLIB DLL。必须在调用数据库接口中的任何函数之前调用 db_init_dll，而且在调用 db_fini_dll 之后不能调用接口中的任何函数。

   您还必须调用 db_init 和 db_fini 库函数。
2. 您必须在 EXEC SQL INCLUDE SQLCA 语句之前包含 `esqldll.h` 头文件或在嵌入式 SQL 程序中包含 `sqlca.h`。`esqldll.h` 头文件包括 `sqlca.h`。
3. 必须定义一个 SQL OS 宏。`sqlca.h` 所包含的头文件 `sqlos.h` 会尝试确定适合的宏并对其进行定义。但是，平台和编译器的某些组合可能导致此操作失败。在这种情况下，您必须将 #define 添加到此文件的顶部，或者使用编译器选项进行定义。必须为 Windows 定义的宏如下所示。

宏	平台
_SQL_OS_WINDOWS	所有 Windows 操作系统

4. 编译器 esqldll.c。
5. 将对象模块 esqldll.obj 与嵌入式 SQL 应用程序对象链接在一起，而不链接到导入库。

运行嵌入式 SQL 应用程序时，DBLIB 接口 DLL 会动态装载。

示例嵌入式 SQL 程序

示例嵌入式 SQL 程序随 SAP Sybase IQ 安装程序一起提供。它们位于 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C 目录中。

- 静态游标嵌入式 SQL 示例 cur.sqc 演示如何使用静态 SQL 语句。
- 动态游标嵌入式 SQL 示例 dcur.sqc 演示如何使用动态 SQL 语句。

为了减少示例程序重复的代码量，已经将主线和数据打印函数置于单独的文件中。对于字符模式系统，该文件为 mainch.c；对于窗口环境，该文件为 mainwin.c。

每个示例程序都提供了以下三个从主线调用的例程：

- **WSQLEX_Init** – 连接到数据库并打开游标。
- **WSQLEX_Process_Command** – 处理来自用户的命令，根据需要操作游标。
- **WSQLEX_Finish** – 关闭游标并断开与数据库的连接。

主线的功能是：

1. 调用 WSQLEX_Init 例程。
2. 执行循环，从用户获取命令并调用 WSQL_Process_Command，直到用户退出。
3. 调用 WSQLEX_Finish 例程。

连接到数据库这一操作是通过提供相应用户 ID 和口令的嵌入式 SQL CONNECT 语句完成的。

除了这些示例之外，您还可以找到作为 SAP Sybase IQ 的一部分、演示可用于特定平台的功能的其它程序和源文件。

静态游标示例

本示例演示如何使用游标。此处使用的特定游标从示例数据库中的 Employees 表检索特定信息。游标是静态声明的，也就是说，检索信息的实际 SQL 语句会被硬编码到源程序中。这是了解游标工作方式的一个良好起点。动态游标示例采用第一个示例并将其转换为使用动态 SQL 语句。

open_cursor 例程声明特定 SQL 查询的游标并打开此游标。

显示信息页是由 print 例程完成的。它会循环 *pagesize* 次，从游标中读取一行并将其输出。fetch 例程检查警告条件，如未找到行 (SQLCODE 100)，并在这些条件出现时显示相应的消息。另外，此程序还会将游标重新定位到出现在当前数据页顶部的行之前的行。

move、top 和 bottom 例程使用适当形式的 FETCH 语句来定位游标。此形式的 FETCH 语句实际上不获取数据，它只定位游标。另外，move 作为一个通用的相对定位例程，被设计成按照参数的符号上下移动游标。

在用户退出时，会关闭游标，而且还会释放数据库连接。游标由 ROLLBACK WORK 语句关闭，而连接由 DISCONNECT 释放。

运行静态游标示例程序

运行静态游标示例程序。

前提条件

执行此任务没有前提条件。

过程

可执行文件及相应源代码位于 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C` 目录中。

1. 启动 SAP Sybase IQ 示例数据库 `iqdemo.db`.
2. 用于构建示例程序的文件随示例代码提供。

   要在 Windows 上构建 32 位示例，可使用 `build.bat`。

   要在 Windows 上构建 64 位示例，可使用 `build64.bat`。可能需要设置正确的编译和链接环境。以下示例构建了一个适用于 x64 平台的示例程序。

   ```
   set mssdk=c:\MSSDK\v7.0
   build64
   ```

   要在 Unix 上构建示例，可使用 shell 脚本 `build.sh`。
3. 对于 32 位 Windows 示例，运行文件 `curwin.exe`。

   对于 64 位 Windows 示例，运行文件 `curx64.exe`。

   对于 Unix 示例，运行文件 `cur`。
4. 按照屏幕上的说明进行操作。

各种不同的命令可操作数据库游标并在屏幕上显示查询结果。输入要执行的命令的字母。某些系统可能要求您在键入字母之后按 Enter 键。

动态游标示例

本示例演示如何使用动态 SQL SELECT 语句的游标。

动态游标示例程序 (dcur) 允许用户选择要使用 N 命令查看的表。然后，该程序会根据屏幕大小尽可能多地显示表中的信息。

运行此程序时，它会提示输入连接字符串。以下是一个示例。

```
UID=<userid>;PWD=<your_password>;DBF=iqdemo.db
```

含有嵌入式 SQL 的 C 程序位于 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C 目录中。

dcur 程序使用嵌入式 SQL 接口函数 db_string_connect 连接到数据库。此函数还提供额外功能，可支持用于连接数据库的连接字符串。

open_cursor 例程首先构建 SELECT 语句

```
SELECT * FROM table-name
```

其中 *table-name* 是传递给例程的参数。然后，它使用此字符串准备动态 SQL 语句。

嵌入式 SQL DESCRIBE 语句用于以 SELECT 语句的结果填充 SQLDA 结构。

注意： 最初推测的是 SQLDA 的大小 (3)。如果此值不够大，则会使用数据库服务器返回的 SELECT 列表的实际大小来分配一个大小正确的 SQLDA。

然后，用缓冲区填充 SQLDA 结构以存放代表查询结果的字符串。fill_s_sqlda 例程将 SQLDA 中的所有数据类型转换为 DT_STRING 并分配适当大小的缓冲区。

然后，为此语句声明并打开游标。用于移动和关闭游标的其余例程保持不变。

fetch 例程稍有不同：它将结果放在 SQLDA 结构中，而不是放在一组主机变量中。print 例程有很大改动，可按屏幕宽度显示来自 SQLDA 结构的结果。print 例程还使用 SQLDA 的名称字段来显示每列的标题。

运行动态游标示例程序

运行动态游标示例程序。

前提条件

执行此任务没有前提条件。

过程

可执行文件及相应源代码位于 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\C 目录中。对于 Windows Mobile，\SQLAnywhere\CE\esql_sample 目录下还有另外一个示例。

1. 启动 SAP Sybase IQ 示例数据库 iqdemo.db.
2. 用于构建示例程序的文件随示例代码提供。

   要在 Windows 上构建 32 位示例，可使用 build.bat。

   要在 Windows 上构建 64 位示例，可使用 build64.bat。可能需要设置正确的编译和链接环境。以下示例构建了一个适用于 x64 平台的示例程序。

   ```
   set mssdk=c:\MSSDK\v7.0
   build64
   ```

对于 Windows Mobile，使用适用于 Microsoft Visual C++ 的 `esql_sample.sln` 项目文件。此文件位于 `SQLAnywhere\CE\esql_sample` 中。

要在 Unix 上构建示例，可使用 shell 脚本 `build.sh`。

3. 对于 32 位 Windows 示例，运行文件 `dcurwin.exe`。

   对于 64 位 Windows 示例，运行文件 `dcurx64.exe`。

   对于 Windows Mobile 示例，在 Windows Mobile 设备上部署并运行文件 `esql_sample.exe`。

   对于 Unix 示例，运行文件 `dcur`。
4. 每个示例程序都提供一个控制台类型的用户界面并提示您输入命令。输入以下连接字符串以连接到示例数据库：

   ```
   DSN=Sybase IQ Demo
   ```
5. 每个示例程序都提示您选择一个表。选择示例数据库中的一个表。例如，可以输入 `Customers` 或 `Employees`。
6. 按照屏幕上的说明进行操作。

各种不同的命令可操作数据库游标并在屏幕上显示查询结果。输入要执行的命令的字母。某些系统可能要求您在键入字母之后按 Enter 键。

嵌入式 SQL 数据类型

要在程序和数据库服务器间传送信息，每个数据都必须有一个数据类型。嵌入式 SQL 数据类型常量以 DT_ 为前缀，并且可以在 `sqldef.h` 头文件中找到。您可以创建任何一种受支持类型的主机变量。您还可以在 SQLDA 结构中使用这些类型向数据库中传入和从中传出数据。

可以使用 `sqlca.h` 中列出的 DECL_ 宏定义这些数据类型的变量。例如，可使用 DECL_BIGINT 来声明保存 BIGINT 值的变量。

以下数据类型受嵌入式 SQL 编程接口的支持：

- **DT_BIT** – 8 位有符号整数。
- **DT_SMALLINT** – 16 位有符号整数。
- **DT_UNSSMALLINT** – 16 位无符号整数。
- **DT_TINYINT** – 8 位有符号整数。
- **DT_BIGINT** – 64 位有符号整数。
- **DT_UNSBIGINT** – 64 位无符号整数。
- **DT_INT** – 32 位有符号整数。
- **DT_UNSINT** – 32 位无符号整数。
- **DT_FLOAT** – 4 字节浮点数。
- **DT_DOUBLE** – 8 字节浮点数。

- **DT_DECIMAL** – 压缩十进制数（专有格式）。

```
typedef struct TYPE_DECIMAL {
  char array[1];
} TYPE_DECIMAL;
```

- **DT_STRING** – 在 CHAR 字符集中以空值终止的字符串。如果数据库是使用填补空白的字符串进行的初始化，则该字符串便是用空白填补的。
- **DT_NSTRING** – 在 NCHAR 字符集中以 NULL 终止的字符串。如果数据库是使用填补空白的字符串进行的初始化，则该字符串便是用空白填补的。
- **DT_DATE** – 以 NULL 终止的字符串，是一个有效日期。
- **DT_TIME** – 以 NULL 终止的字符串，是一个有效时间。
- **DT_TIMESTAMP** – 以 NULL 终止的字符串，是一个有效时间戳。
- **DT_FIXCHAR** – 在 CHAR 字符集中填补空白的定长字符串。最大长度为 32767（以字节为单位）。该数据不以空值终止。
- **DT_NFIXCHAR** – 在 NCHAR 字符集中填补空白的定长字符串。最大长度为 32767（以字节为单位）。该数据不以空值终止。
- **DT_VARCHAR** – 在 CHAR 字符集中具有双字节长度字段的变长字符串。最大长度为 32765 个字节。发送数据时，必须设置长度字段。读取数据时，数据库服务器设置长度字段。该数据不是以空值终止或以空白填补的。

```
typedef struct VARCHAR {
 a_sql_ulen len;
 char       array[1];
} VARCHAR;
```

- **DT_NVARCHAR** – 在 NCHAR 字符集中具有双字节长度字段的变长字符串。最大长度为 32765 个字节。发送数据时，必须设置长度字段。读取数据时，数据库服务器设置长度字段。该数据不是以空值终止或以空白填补的。

```
typedef struct NVARCHAR {
 a_sql_ulen len;
 char       array[1];
} NVARCHAR;
```

- **DT_LONGVARCHAR** – 在 CHAR 字符集中长变长字符串。

```
typedef struct LONGVARCHAR {
 a_sql_uint32 array_len;  /* number of allocated bytes in array */
 a_sql_uint32 stored_len; /* number of bytes stored in array
                           * (never larger than array_len) */
 a_sql_uint32 untrunc_len;/* number of bytes in untruncated
expression
                           * (may be larger than array_len) */
 char         array[1];   /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

  LONGVARCHAR 结构可用于大于 32767 个字节的数据。对于较大的数据，可以一次全部读取，也可以使用 GET DATA 语句逐段读取。对于较大的数据，可以一次就全部提供给服务器，也可以通过使用 SET 语句附加到数据库变量中来逐段提供。该数据不是以空值终止或以空白填补的。

- **DT_LONGNVARCHAR** – NCHAR 字符集中的长变长字符串。该宏定义一个如下结构：

```
typedef struct LONGVARCHAR {
 a_sql_uint32 array_len;  /* number of allocated bytes in array */
 a_sql_uint32 stored_len; /* number of bytes stored in array
                           * (never larger than array_len) */
 a_sql_uint32 untrunc_len;/* number of bytes in untruncated
expression
                           * (may be larger than array_len) */
 char         array[1];   /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

LONGNVARCHAR 结构可用于大于 32767 个字节的数据。对于较大的数据，可以一次全部读取，也可以使用 GET DATA 语句逐段读取。对于较大的数据，可以一次就全部提供给服务器，也可以通过使用 SET 语句附加到数据库变量中来逐段提供。该数据不是以空值终止或以空白填补的。

- **DT_BINARY** – 具有双字节长度字段的变长二进制数据。最大长度为 32765 个字节。在向数据库服务器提供信息时，您必须设置长度字段。在从数据库服务器读取信息时，服务器设置长度字段。

```
typedef struct BINARY {
 a_sql_ulen len;
 char       array[1];
} BINARY;
```

- **DT_LONGBINARY** – 长二进制数据。该宏定义一个如下结构：

```
typedef struct LONGVARCHAR {
 a_sql_uint32 array_len;  /* number of allocated bytes in array */
 a_sql_uint32 stored_len; /* number of bytes stored in array
                           * (never larger than array_len) */
 a_sql_uint32 untrunc_len;/* number of bytes in untruncated
expression
                           * (may be larger than array_len) */
 char         array[1];   /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

LONGBINARY 结构可用于大于 32767 个字节的数据。对于较大的数据，可以一次全部读取，也可以使用 GET DATA 语句逐段读取。对于较大的数据，可以一次就全部提供给服务器，也可以通过使用 SET 语句附加到数据库变量中来逐段提供。

- **DT_TIMESTAMP_STRUCT** – SQLDATETIME 结构，对于时间戳的每一部分都有一个字段。

```
typedef struct sqldatetime {
 unsigned short year; /* for example 1999 */
 unsigned char  month; /* 0-11 */
 unsigned char  day_of_week; /* 0-6 0=Sunday */
 unsigned short day_of_year; /* 0-365 */
 unsigned char  day; /* 1-31 */
 unsigned char  hour; /* 0-23 */
 unsigned char  minute; /* 0-59 */
 unsigned char  second; /* 0-59 */
 unsigned long  microsecond; /* 0-999999 */
} SQLDATETIME;
```

SQLDATETIME 结构可用于检索 DATE、TIME 和 TIMESTAMP 类型（或者可转换为这些类型之一的任意类型）的字段。应用程序常常具有它们自己的格式和日期操作代码。在该结构中读取数据可为操作该数据提供更多方便。也可以使用任何字符类型来读取和更新 DATE、TIME 和 TIMESTAMP 字段。

如果您使用 SQLDATETIME 结构将日期、时间或时间戳输入到数据库中，则会忽略 day_of_year 和 day_of_week 成员。

- **DT_VARIABLE** – 以 NULL 终止的字符串。字符串必须是 SQL 变量的名称，数据库服务器使用该变量的值。此数据类型仅用于为数据库服务器提供数据。在从数据库服务器读取数据时，不能使用它。

这些结构在 `sqlca.h` 文件中定义。由于 VARCHAR、NVARCHAR、BINARY、DECIMAL 和 LONG 数据类型包含一个单字符数组，因此，它们对于声明主机变量没有帮助。但是，它们对于动态分配变量或强制转换其它变量的类型十分有用。

DATE 和 TIME 数据库类型

对于各种 DATE 和 TIME 数据库类型，没有相应的嵌入式 SQL 接口数据类型。这些数据库类型都是使用 SQLDATETIME 结构或字符串读取和更新的。

嵌入式 SQL 中的主机变量

主机变量是供 SQL 预处理器识别的 C 语言变量。主机变量可用于将值发送到数据库服务器或从数据库服务器接收值。

主机变量非常易于使用，但是它们具有一些限制。动态 SQL 是一种向数据库服务器和从数据库服务器中传递信息的更常用的方法，它使用被称为 SQL 描述符区域 (SQLDA) 的结构。SQL 预处理器为使用主机变量的每个语句自动生成 SQLDA。

主机变量无法在批处理中使用。主机变量不能在 SET 语句的子查询中使用。

主机变量声明

主机变量是通过将它们放入声明部分来定义的。按照 ANSI 嵌入式 SQL 标准，主机变量是通过用以下内容围绕常规 C 变量声明定义的：

```
EXEC SQL BEGIN DECLARE SECTION;
/* C variable declarations */
EXEC SQL END DECLARE SECTION;
```

然后，可以使用这些主机变量代替任意 SQL 语句中的值常量。在数据库服务器执行语句时，会使用主机变量的值。主机变量不能用于代替表或列的名称：此参数需要动态 SQL。在 SQL 语句中，变量名以冒号 (:) 为前缀，以便与语句中允许使用的其它标识符区别开。

在 SQL 预处理器中，只在 DECLARE SECTION 内部扫描 C 语言代码。因此，在 DECLARE SECTION 内，不允许使用 TYPEDEF 类型和结构，但允许使用变量的初始化程序。

示例

下面的示例代码阐释了主机变量在 INSERT 语句中是如何使用的。这些变量由程序填充，然后插入到数据库中：

```
EXEC SQL BEGIN DECLARE SECTION;
long employee_number;
char employee_name[50];
char employee_initials[8];
char employee_phone[15];
EXEC SQL END DECLARE SECTION;
/* program fills in variables with appropriate values
*/
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone );
```

C 主机变量类型

仅有有限数量的 C 数据类型可作为主机变量。而且，某些主机变量类型没有对应的 C 类型。

sqlca.h 头文件中定义的宏可用于声明以下类型的主机变量：NCHAR、VARCHAR、NVARCHAR、LONGVARCHAR、LONGNVARCHAR、BINARY、LONGBINARY、DECIMAL、DT_FIXCHAR、DT_NFIXCHAR、DATETIME (SQLDATETIME)、BIT、BIGINT 或 UNSIGNED BIGINT。它们的使用方法如下所示：

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_NCHAR                  v_nchar[10];
DECL_VARCHAR( 10 )          v_varchar;
DECL_NVARCHAR( 10 )         v_nvarchar;
DECL_LONGVARCHAR( 32768 )   v_longvarchar;
DECL_LONGNVARCHAR( 32768 )  v_longnvarchar;
DECL_BINARY( 4000 )         v_binary;
DECL_LONGBINARY( 128000 )   v_longbinary;
DECL_DECIMAL( 30, 6 )       v_decimal;
DECL_FIXCHAR( 10 )          v_fixchar;
DECL_NFIXCHAR( 10 )         v_nfixchar;
DECL_DATETIME               v_datetime;
DECL_BIT                    v_bit;
DECL_BIGINT                 v_bigint;
DECL_UNSIGNED_BIGINT        v_ubigint;
EXEC SQL END DECLARE SECTION;
```

预处理器能够识别嵌入式 SQL 声明部分中的这些宏，并可将变量视为相应的类型。建议不要使用 DECIMAL（DT_DECIMAL、DECL_DECIMAL）类型，因为十进制数值的格式为专有格式。

下表列出了允许用于主机变量的 C 变量类型及其对应的嵌入式 SQL 接口数据类型。

C 数据类型	嵌入式 SQL 接口类型	说明
`short si; short int si;`	DT_SMALLINT	16 位有符号整数。
`unsigned short int usi;`	DT_UNSSMALLINT	16 位无符号整数。
`long l; long int l;`	DT_INT	32 位有符号整数。
`unsigned long int ul;`	DT_UNSINT	32 位无符号整数。
`DECL_BIGINT ll;`	DT_BIGINT	64 位有符号整数。
`DECL_UNSIGNED_BIGINT ull;`	DT_UNSBIGINT	64 位无符号整数。
`float f;`	DT_FLOAT	4 字节单精度浮点值。
`double d;`	DT_DOUBLE	8 字节双精度浮点值。
`char a[n]; /*n>=1*/`	DT_STRING	在 CHAR 字符集中以空值终止的字符串。 如果数据库是使用填补空白的字符串进行的初始化，则该字符串便是用空白填补的。此变量占有 n-1 个字符加上空终止符。
`char *a;`	DT_STRING	在 CHAR 字符集中以空值终止的字符串。此变量指向一个可保存最多 32766 个字节加上空终止符的区域。
`DECL_NCHAR a[n]; /*n>=1*/`	DT_NSTRING	在 NCHAR 字符集中以空值终止的字符串。如果数据库是使用填补空白的字符串进行的初始化，则该字符串便是用空白填补的。此变量占有 n-1 个字符加上空终止符。
`DECL_NCHAR *a;`	DT_NSTRING	在 NCHAR 字符集中以空值终止的字符串。此变量指向一个可保存最多 32766 个字节加上空终止符的区域。

C 数据类型	嵌入式 SQL 接口类型	说明
`DECL_VARCHAR(n) a;`	DT_VARCHAR	在 CHAR 字符集中具有双字节长度字段的变长字符串。不是以空值终止的或以空白填补的。n 的最大值为 32765（字节）。
`DECL_NVARCHAR(n) a;`	DT_NVARCHAR	在 NCHAR 字符集中具有双字节长度字段的变长字符串。不是以空值终止的或以空白填补的。n 的最大值为 32765（字节）。
`DECL_LONGVARCHAR(n) a;`	DT_LONGVARCHAR	在 CHAR 字符集中具有三个 4 字节长度字段的变长长字符串。不是以空值终止的或以空白填补的。
`DECL_LONGNVARCH-AR(n) a;`	DT_LONGNVARCHAR	在 NCHAR 字符集中具有三个 4 字节长度字段的变长长字符串。不是以空值终止的或以空白填补的。
`DECL_BINARY(n) a;`	DT_BINARY	具有长度为 2 个字节的字段的变长二进制数据。n 的最大值为 32765（字节）。
`DECL_LONGBINARY(n) a;`	DT_LONGBINARY	具有三个长度为 4 个字节的字段的长型变长二进制数据。
`char a; /*n=1*/ DECL_FIXCHAR(n) a;`	DT_FIXCHAR	在 CHAR 字符集中固定长度的字符串。是填补空白的但不是以空值终止的。n 的最大值为 32767（字节）。
`DECL_NCHAR a; /*n=1*/ DECL_NFIXCHAR(n) a;`	DT_NFIXCHAR	在 NCHAR 字符集中固定长度的字符串。是填补空白的但不是以空值终止的。n 的最大值为 32767（字节）。
`DECL_DATETIME a;`	DT_TIMESTAMP_STRUCT	SQLDATETIME 结构

字符集

对于 DT_FIXCHAR、DT_STRING、DT_VARCHAR 和 DT_LONGVARCHAR，字符数据采用应用程序的 CHAR 字符集，此字符集通常是应用程序区域设置的字符集。应用程序可使用 CHARSET 连接参数或通过调用 db_change_char_charset 函数来更改 CHAR 字符集。

对于 DT_NFIXCHAR、DT_NSTRING、DT_NVARCHAR 和 DT_LONGNVARCHAR，数据采用应用程序的 NCHAR 字符集。缺省情况下，应用程序的 NCHAR 字符集与

CHAR 字符集相同。应用程序可以通过调用 db_change_nchar_charset 函数来更改 NCHAR 字符集。

数据长度
无论使用的是 CHAR 还是 NCHAR 字符集，所有数据长度均以字节为单位指定。

如果服务器和应用程序之间发生字符集转换，则应用程序负责确保缓冲区足够大以便处理转换的数据，并在数据被截断时发出附加 GET DATA 语句。

字符指针
数据库接口认为一个声明为*字符指针* (*char * a*) 的主机变量长度为 32767 个字节。用于从数据库检索信息的任何字符指针类型的主机变量都必须指向一个缓冲区，该缓冲区的大小要足以容纳可能从数据库返回的任何值。

这具有相当大的潜在危险，因为可能会有人更改数据库中列的定义，使列比编写程序时更大。这可能会导致随机内存损坏问题。使用声明数组会比较好，甚至是作为函数的参数，其中数组是作为字符指针传递的。此技术可让嵌入式 SQL 语句知道数组的大小。

主机变量的范围
标准主机变量声明部分可以出现在通常声明 C 变量的任意位置，包括 C 函数的参数声明部分。C 语言变量有其常规作用域（在定义它们的块中可用）。但是，因为 SQL 预处理器不扫描 C 代码，所以它不会考虑 C 语言块。

就 SQL 预处理器而言，主机变量对源文件来说是全局的；两个主机变量不能同名。

主机变量的用法

主机变量可用于以下环境中：

- 位于任何允许使用数字或字符串常量的位置的 SELECT、INSERT、UPDATE 和 DELETE 语句。
- SELECT 和 FETCH 语句的 INTO 子句。
- 主机变量也可用来代替语句名、游标名或嵌入式 SQL 特定语句中的选项名。
- 对于 CONNECT、DISCONNECT 和 SET CONNECT 语句，可以用主机变量代替服务器名、数据库名、连接名、用户 ID、口令或连接字符串。
- 对于 SET OPTION 和 GET OPTION，可以用主机变量代替选项值。

主机变量不可用于以下环境中：

- 不能用主机变量代替任何语句中的表名或列名。
- 主机变量无法在批处理中使用。
- 主机变量不能在 SET 语句的子查询中使用。

SQLCODE 和 SQLSTATE 主机变量
ISO/ANSI 标准允许嵌入式 SQL 源文件在嵌入式 SQL 声明部分中声明以下特殊主机变量：

```
long SQLCODE;
char SQLSTATE[6];
```

如果使用这些变量，则在进行数据库请求的任何嵌入式 SQL 语句（DECLARE SECTION、INCLUDE、WHENEVER SQLCODE 等之外的 EXEC SQL 语句）之后设置这些变量。因此，SQLCODE 和 SQLSTATE 主机变量必须在生成数据库请求的每个嵌入式 SQL 语句的范围内可见。

下面是有效的嵌入式 SQL：

```
EXEC SQL INCLUDE SQLCA;
// declare SQLCODE with global scope
EXEC SQL BEGIN DECLARE SECTION;
long SQLCODE;
EXEC SQL END DECLARE SECTION;
sub1() {
 EXEC SQL BEGIN DECLARE SECTION;
 char SQLSTATE[6];
 EXEC SQL END DECLARE SECTION;
 exec SQL CREATE TABLE ...
}
sub2() {
 EXEC SQL BEGIN DECLARE SECTION;
 char SQLSTATE[6];
 EXEC SQL END DECLARE SECTION;
 exec SQL DROP TABLE ...
}
```

因为未在函数 sub2 的范围内定义 SQLSTATE，所以以下嵌入式 SQL 是无效的。

```
EXEC SQL INCLUDE SQLCA;
sub1() {
 EXEC SQL BEGIN DECLARE SECTION;
 char SQLSTATE[6];
 EXEC SQL END DECLARE SECTION;
 exec SQL CREATE TABLE...
}
sub2() {
 exec SQL DROP TABLE...
}
```

指示符变量

指示符变量是在您读取或保存数据时存放补充信息的 C 变量。指示符变量有以下几种不同的用法：

- **NULL** – 使应用程序可以处理 NULL。
- **字符串截断** – 使应用程序可以处理必须截断读取值以适合主机变量的情况。
- **转换错误** – 保存错误消息。

指示符变量是在 SQL 语句中紧跟常规主机变量放置的 a_sql_len 类型的主机变量。例如，在下面的 INSERT 语句中，:ind_phone 是一个指示符变量：

```
EXEC SQL INSERT INTO Employees
 VALUES (:employee_number, :employee_name,
 :employee_initials, :employee_phone:ind_phone );
```

如果在读取或执行操作中，未从数据库服务器接收到任何行（由于出错或者到达结果集的末尾），则指示符值保持不变。

注意： 为了可以在将来使用 32 位与 64 位长度和指示符，不建议使用短整型嵌入式 SQL 指示符变量。请改用 a_sql_len。

指示符变量：SQL NULL

不要将 SQL 的 NULL 概念与同名的 C 语言常量相混淆。在 SQL 语言中，NULL 代表未知属性或不适用的信息。C 语言常量表示未指向内存位置的指针值。

在 SAP Sybase IQ 文档中使用 NULL 时，它指的是以上给出的 SQL 数据库含义。该 C 语言常量称为空指针（小写）。

NULL 不同于列的已定义类型的任何值。因此，要将 NULL 传递到数据库或接收回 NULL 结果，除了常规主机变量外，还需要其它额外的条件。指示符变量正是用于此目的。

插入 NULL 时使用指示符变量

INSERT 语句可以包括一个指示符变量，如下所示：

```
EXEC SQL BEGIN DECLARE SECTION;
short int employee_number;
char employee_name[50];
char employee_initials[6];
char employee_phone[15];
a_sql_len ind_phone;
EXEC SQL END DECLARE SECTION;
/*
This program fills in the employee number,
name, initials, and phone number.
*/
if( /* Phone number is unknown */ ) {
 ind_phone = -1;
} else {
 ind_phone = 0;
}
EXEC SQL INSERT INTO Employees
 VALUES (:employee_number, :employee_name,
 :employee_initials, :employee_phone:ind_phone );
```

如果指示符变量值为 -1，则写入 NULL。如果其值为 0，则写入 employee_phone 的实际值。

在读取 NULL 时使用指示符变量

从数据库中接收数据时也可使用指示符变量。它们用于指示已读取 NULL（指示符为负数）。如果从数据库中读取了 NULL，而又没有提供指示符变量，就会生成错误 (SQLE_NO_INDICATOR)。

指示符变量：截断值

指示符变量指示某些读取值是否为适合主机变量而被截断了。这使应用程序可以正确地处理截断问题。

如果一个值在读取时被截断，则指示符变量会被设为正值，其中包含了在被截断前数据库值的实际长度。如果数据库值的实际长度大于 32767 个字节，则指示符变量值为 32767。

指示符变量：转换错误

缺省情况下，conversion_error 数据库选项设置为 On，任何数据类型转换失败都将导致一个错误，且不返回行。

您可以使用指示符变量来告知哪列导致了数据类型转换失败。如果您将数据库选项 conversion_error 设置为 Off，则任何数据类型转换失败都将给出 CANNOT_CONVERT 警告，而不是错误。如果遇到转换错误的列具有一个指示符变量，则将该变量的值设置为 -2。

如果您在向数据库中插入数据时将 conversion_error 选项设置为 Off，则发生转换失败时就会插入 NULL。

指示符变量值概览

下表提供指示符变量用法的概览。

指示符的值	向数据库提供值	从数据库接收值
> 0	主机变量的值	检索的值被截断 - 指示符变量中的实际长度。
0	主机变量的值	读取成功，或 conversion_error 设置为 On。
-1	NULL	NULL 结果。
-2	NULL	转换错误（仅当 conversion_error 设置为 Off 时）。 SQLCODE 指示一个 CANNOT_CONVERT 警告。
< -2	NULL	NULL 结果。

SQL 通信区域 (SQLCA)

SQL 通信区域 (SQLCA) 是一个内存区域，每个数据库请求都会利用这个区域将统计信息和错误从应用程序传递到数据库服务器再回传到应用程序。SQLCA 用作应用程序到数据库的通信链接的句柄。它会被传递到需要与数据库服务器进行通信的所有数据库库函数中。它在所有嵌入式 SQL 语句中被隐式传递。

全局 SQLCA 变量在接口库中定义。预处理器会为全局 SQLCA 变量生成外部引用，并且会为该变量的指针生成外部引用。该外部引用名为 sqlca，类型为 SQLCA。指针名为 sqlcaptr。实际的全局变量在导入库中声明。

SQLCA 由 sqlca.h 头文件定义，该文件包含在安装目录的 SDK\Include 子目录中。

SQLCA 提供错误代码
可引用 SQLCA 来测试特定错误代码。当数据库请求有错误时，sqlcode 和 sqlstate 字段包含错误代码。某些 C 宏是为引用 sqlcode 字段、sqlstate 字段和某些其它字段而定义的。

SQLCA 字段

SQLCA 中的字段具有以下含义：

- **sqlcaid** – 8 字节字符字段，包含作为 SQLCA 结构标识的字符串 SQLCA。在您查看内存内容时，该字段可帮助进行调试。
- **sqlcabc** – 包含 SQLCA 结构长度（136 字节）的 32 位整数。
- **sqlcode** – 数据库检测到请求有错误时，用来指定错误代码的 32 位整数。错误代码的定义可在头文件 sqlerr.h 中找到。错误代码是 0（零）表示操作成功，正数表示警告，负数表示错误。
- **sqlerrml** – sqlerrmc 字段中信息的长度。
- **sqlerrmc** – 要插入到错误消息中的零个或多个字符串。某些错误消息包含一个或多个占位符字符串（*%1*、*%2*...），这些占位符字符串可由此字段中的字符串替换。

  例如，如果生成 Table Not Found 错误，则 sqlerrmc 包含表名，该表名会插入到错误消息中的相应位置。
- **sqlerrp** – 保留。
- **sqlerrd** – 由 32 位整数组成的实用程序数组。
- **sqlwarn** – 保留。
- **sqlstate** – SQLSTATE 状态值。除了 SQLCODE 值外，ANSI SQL 标准还定义了 SQL 语句的此种类型的返回值。SQLSTATE 值始终是一个由五个字符组成且以空值终止的字符串，它分为双字符类（前两个字符）和三字符子类。每个字符都可以是从 0 到 9 的数字或从 A 到 Z 的大写字母字符。

  以 0 到 4 或 A 到 H 开头的任何类或子类都是由 SQL 标准定义的，其它类和子类则是各实现自行定义的。SQLSTATE 值 '00000' 表示还没有错误或警告。

sqlerror 数组
sqlerror 字段数组具有以下元素。

- **sqlerrd[1] (SQLIOCOUNT)** – 完成某条语句所需的实际输入/输出操作数。

  数据库服务器执行每条语句之前不会将此值设置为零。在执行一个语句序列之前，您的程序可以将此变量设置为零。执行完最后一条语句之后，该数字为整个语句序列的输入/输出操作的总数。
- **sqlerrd[2] (SQLCOUNT)** – 此字段的值取决于要执行的语句。

- **INSERT、UPDATE、PUT 和 DELETE 语句** – 受语句影响的行数。
- **OPEN 和 RESUME 语句** – 在游标 OPEN 或 RESUME 上，此字段由游标中的实际行数（大于或等于 0 的值）或它的估计值（绝对值为估计值的负数）填充。如果数据库服务器不统计该值即可计算出行数，则该值就是实际行数。也可以使用 row_counts 选项，将数据库配置为始终返回实际的行数。
- **FETCH 游标语句** – 如果返回 SQLE_NOTFOUND 警告，则填充 SQLCOUNT 字段。它包含 FETCH RELATIVE 或 FETCH ABSOLUTE 语句超出可能的游标位置（游标可以位于某一行上、第一行之前或最后一行之后）范围之外的行数。对于宽读取，SQLCOUNT 是实际读取的行数，它小于或等于请求的行数。在宽读取过程中，只在未返回行的情况下才设置 SQLE_NOTFOUND。

  如果未找到行但位置有效，则值为 0，例如，当定位到游标的最后一行上时执行 FETCH RELATIVE 1。如果所尝试的读取超出了游标的末尾，则为正值；如果所尝试的读取位于游标开头的前面，则为负值。
- **GET DATA 语句** – SQLCOUNT 字段保存值的实际长度。
- **DESCRIBE 语句** – 如果使用 WITH VARIABLE RESULT 子句描述可能具有多个结果集的过程，则 SQLCOUNT 将会设置为以下值之一：
  - **0** – 结果集可能更改：应该在每个 OPEN 语句后重新描述过程调用。
  - **1** – 结果集是固定的。不需要重新进行描述。

  如果出现语法错误 SQLE_SYNTAX_ERROR，此字段包含语句内检测到错误的大致字符位置。
- **sqlerrd[3] (SQLIOESTIMATE)** – 完成语句所需的输入/输出操作的估计数。OPEN 或 EXPLAIN 语句将为此字段赋一个值。

多线程代码或重入代码的 SQLCA 管理

您可以在多线程代码或重入代码中使用嵌入式 SQL 语句。但是，如果您使用单个连接，则对于每个连接您只能有一个活动请求。在多线程应用程序中，除非使用信号控制访问，否则在每个线程上均不应使用到数据库的同一连接。

对于在希望使用数据库的每个线程上使用单独的连接没有限制。运行时库使用 SQLCA 来区分不同的线程上下文。因此，希望并行使用数据库的每个线程都必须具有自己的 SQLCA。例外的情况是，一个线程可以使用 db_cancel_request 函数取消对另一个使用此线程的 SQLCA 的线程执行的语句。

以下是一个重入多线程嵌入式 SQL 代码的示例。

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>
#include <process.h>
#include <windows.h>
EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

#define TRUE 1
```

```
#define FALSE 0

// multithreading support

typedef struct a_thread_data {
    SQLCA sqlca;
    int num_iters;
    int thread;
    int done;
} a_thread_data;

// each thread's ESQL test

EXEC SQL SET SQLCA "&thread_data->sqlca";

static void PrintSQLError( a_thread_data * thread_data )
/******************************************************/
{
  char                buffer[200];

  printf( "%d: SQL error %d -- %s ... aborting\n",
      thread_data->thread,
      SQLCODE,
      sqlerror_message( &thread_data->sqlca,
                buffer, sizeof( buffer ) ) );
    exit( 1 );
}

EXEC SQL WHENEVER SQLERROR { PrintSQLError( thread_data ); };

static void do_one_iter( void * data )
{
  a_thread_data *  thread_data = (a_thread_data *)data;
  int    i;
  EXEC SQL BEGIN DECLARE SECTION;
  char    user[ 20 ];
  EXEC SQL END DECLARE SECTION;

  if( db_init( &thread_data->sqlca ) != 0 ) {
    for( i = 0; i < thread_data->num_iters; i++ ) {
     EXEC SQL CONNECT "dba" IDENTIFIED BY "sql";
     EXEC SQL SELECT USER INTO :user;
     EXEC SQL DISCONNECT;
    }
    printf( "Thread %d did %d iters successfully\n",
        thread_data->thread, thread_data->num_iters );
    db_fini( &thread_data->sqlca );
  }
  thread_data->done = TRUE;
}

int main()
{
  int num_threads = 4;
  int thread;
  int num_iters = 300;
```

```
  int num_done = 0;
  a_thread_data *thread_data;
  thread_data = (a_thread_data *)malloc( sizeof(a_thread_data) *
num_threads );
  for( thread = 0; thread < num_threads; thread++ ) {
    thread_data[ thread ].num_iters = num_iters;
    thread_data[ thread ].thread = thread;
    thread_data[ thread ].done = FALSE;
    if( _beginthread( do_one_iter,
      8096,
      (void *)&thread_data[thread] ) <= 0 ) {
      printf( "FAILED creating thread.\n" );
      return( 1 );
    }
  }
    while( num_done != num_threads ) {
    Sleep( 1000 );
    num_done = 0;
    for( thread = 0; thread < num_threads; thread++ ) {
      if( thread_data[ thread ].done == TRUE ) {
        num_done++;
      }
    }
  }
  return( 0 );
}
```

多个 SQLCA

不要使用可生成非重入代码的 SQL 预处理器选项 (-r-)。由于无法使用静态初始化的全局变量，因此重入代码稍大且速度稍慢。不过，这些影响是很小的。

在程序中使用的每个 SQLCA 都必须调用 db_init 进行初始化，并在结尾处调用 db_fini 清除它。

嵌入式 SQL 语句 SET SQLCA 用于通知 SQL 预处理器对数据库请求使用不同的 SQLCA。通常，会在程序顶部或头文件中使用类似 `EXEC SQL SET SQLCA 'task_data->sqlca';` 的语句，将 SQLCA 引用设置为指向任务特定的数据。由于此语句不生成任何代码，因而不影响性能。它更改预处理器内的状态，以便对 SQLCA 的任何引用都使用给定的字符串。

每个线程都必须具有自己的 SQLCA。此要求也适用于使用嵌入式 SQL、并被应用程序中的多个线程调用的共享库（例如一个 DLL）中的代码。

您可以在任一受支持的嵌入式 SQL 环境中使用多个 SQLCA 支持，但仅在重入代码中要求这样做。

您无需使用多个 SQLCA 以连接到多个数据库或具有到单个数据库的多个连接。

每个 SQLCA 都可以具有一个未命名的连接。每个 SQLCA 都具有一个活动的或当前的连接。

在所给定数据库连接上进行的所有操作都必须使用建立数据库连接时所使用的同一个 SQLCA。

注意： 不同连接上的操作受常规记录锁定机制的影响，并且可能导致互相阻塞，甚至可能导致死锁。

静态和动态 SQL

有以下两种方式可以将 SQL 语句嵌入到 C 程序中：

静态语句
动态语句

静态 SQL 语句

通过在所有标准 SQL 数据操作语句和数据定义语句之前加上 EXEC SQL，并在语句之后加上分号 (;)，可以将所有这些语句嵌入到 C 程序中。这些语句称为*静态*语句。

静态语句可以包含对主机变量的引用。主机变量只能用来代替字符串或数字常量，不能用来代替列名或表名；执行那些操作需要使用动态语句。

动态 SQL 语句

在 C 语言中，字符串存储在字符数组中。动态语句是用 C 语言字符串构造的。然后，可以使用 PREPARE 和 EXECUTE 语句执行这些语句。这些 SQL 语句无法按与静态语句相同的方式引用主机变量，因为在执行 C 程序时无法通过名称访问 C 语言变量。

要在语句与 C 语言变量之间传递信息，请使用名为 *SQL 描述符区域* (SQLDA) 的数据结构。如果您在 EXECUTE 语句的 USING 子句中指定一组主机变量，则 SQL 预处理器会为您建立此结构。这些变量按位置对应于预准备语句相应位置中的占位符。

将*占位符*放入语句中是为了指示要访问主机变量的位置。占位符可以是问号 (?)，也可以是静态语句中的主机变量引用（主机变量名之前有一个冒号）。在后一种情况下，在语句的实际文本中使用的主机变量名仅充当占位符，指示对 SQL 描述符区域的引用。

用于向数据库传递信息的主机变量称为*绑定变量*。

示例

```
EXEC SQL BEGIN DECLARE SECTION;
char      comm[200];
char      street[30];
char      city[20];
a_sql_len cityind;
long      empnum;
EXEC SQL END DECLARE SECTION;

...
sprintf( comm,
    "UPDATE %s SET Street = :?, City = :?"
```

```
    "WHERE EmployeeID = :?",
    tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
EXEC SQL EXECUTE S1 USING :street, :city:cityind, :empnum;
```

此方法要求您知道语句中共有多少个主机变量。但程序员对这一点往往并不清楚。因此，可以建立您自己的 SQLDA 结构，并在 EXECUTE 语句的 USING 子句中指定此 SQLDA。

DESCRIBE BIND VARIABLES 语句返回在预准备语句中找到的绑定变量的主机变量名。这使 C 程序可以更容易地管理主机变量。一般的方法如下：

```
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
...
sprintf( comm,
    "UPDATE %s SET Street = :street, City = :city"
   " WHERE EmployeeID = :empnum",
   tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
/* Assume that there are no more than 10 host variables.
 * See next example if you cannot put a limit on it. */
sqlda = alloc_sqlda( 10 );
EXEC SQL DESCRIBE BIND VARIABLES FOR S1 INTO sqlda;
/* sqlda->sqld will tell you how many
   host variables there were. */
/* Fill in SQLDA_VARIABLE fields with
   values based on name fields in sqlda. */
...
EXEC SQL EXECUTE S1 USING DESCRIPTOR sqlda;
free_sqlda( sqlda );
```

SQLDA 的内容

SQLDA 包含一组变量描述符。每个描述符说明对应的 C 程序变量的属性或者数据库存储数据的位置或检索数据的位置：

数据类型
如果 *type* 是字符串类型，则为长度
内存地址
指示符变量

指示符变量和 NULL

指示符变量用于将 NULL 传递到数据库或从数据库检索 NULL。数据库服务器还使用指示符变量指示在数据库操作过程中遇到的截断条件。当提供的空间不足，无法接收数据库值时，会将指示符变量设置为正值。

动态 SELECT 语句

可以动态准备仅返回单个行的 SELECT 语句，后跟带 INTO 子句的 EXECUTE 以检索单行结果。但是，返回多个行的 SELECT 语句是使用动态游标管理的。

使用动态游标时，将结果放在 FETCH 语句（FETCH INTO 和 FETCH USING DESCRIPTOR）指定的主机变量列表或 SQLDA 中。由于通常不知道 SELECT 列表项数，因此最常使用 SQLDA。DESCRIBE SELECT LIST 语句建立具有 SELECT 列表项的类型的 SQLDA。然后，使用 fill_sqlda 或 fill_s_sqlda 函数为这些值分配空间，由 FETCH USING DESCRIPTOR 语句检索信息。

典型的情况如下:

```
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
int actual_size;
SQLDA * sqlda;
...
sprintf( comm, "SELECT * FROM %s", table_name );
EXEC SQL PREPARE S1 FROM :comm;
/* Initial guess of 10 columns in result.
   If it is wrong, it is corrected right
   after the first DESCRIBE by reallocating
   sqlda and doing DESCRIBE again. */
sqlda = alloc_sqlda( 10 );
EXEC SQL DESCRIBE SELECT LIST FOR S1
  INTO sqlda;
if( sqlda->sqld > sqlda->sqln )
{
  actual_size = sqlda->sqld;
  free_sqlda( sqlda );
  sqlda = alloc_sqlda( actual_size );
  EXEC SQL DESCRIBE SELECT LIST FOR S1
    INTO sqlda;
}
fill_sqlda( sqlda );
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
EXEC SQL WHENEVER NOTFOUND {break};
for( ;; )
{
  EXEC SQL FETCH C1 USING DESCRIPTOR sqlda;
  /* do something with data */
}
EXEC SQL CLOSE C1;
EXEC SQL DROP STATEMENT S1;
```

注意: 为避免占用不必要的资源，请确保在使用语句后将其删除。

SQL 描述符区域 (SQLDA)

SQLDA（SQL 描述符区域）是一个用于动态 SQL 语句的接口结构。此结构用于将有关主机变量和 SELECT 语句结果的信息传入和传出数据库。SQLDA 在头文件 `sqlda.h` 中定义。

数据库接口共享库或 DLL 中提供了可用于管理 SQLDA 的函数。

当主机变量与静态 SQL 语句一起使用时，预处理器会为这些主机变量构造 SQLDA。实际上，被传入传出数据库服务器的正是此 SQLDA。

SQLDA 头文件

`sqlda.h` 的内容如下：

```
#ifndef _SQLDA_H_INCLUDED
#define _SQLDA_H_INCLUDED
#define II_SQLDA
#include "sqlca.h"
#if defined( _SQL_PACK_STRUCTURES )
  #if defined( _MSC_VER ) && _MSC_VER > 800
    #pragma warning(push)
    #pragma warning(disable:4103)
  #endif
  #include "pshpk1.h"
#endif
#define SQL_MAX_NAME_LEN    30
#define _sqldafar
typedef short int a_sql_type;

struct sqlname {
  short int   length; /* length of char data */
  char        data[ SQL_MAX_NAME_LEN ]; /* data */
};

struct sqlvar {          /* array of variable descriptors */
  short int   sqltype;   /* type of host variable */
  a_sql_len   sqllen;    /* length of host variable */
  void        *sqldata;  /* address of variable */
  a_sql_len   *sqlind;   /* indicator variable pointer */
  struct sqlname sqlname;
};

#if defined( _SQL_PACK_STRUCTURES )
  #include "poppk.h"
  /* The SQLDA should be 4-byte aligned */
  #include "pshpk4.h"
#endif

struct sqlda {
  unsigned char   sqldaid[8]; /* eye catcher "SQLDA" */
  a_sql_int32     sqldabc; /* length of sqlda structure */
```

```
  short int       sqln;    /* descriptor size in number of entries */
  short int      sqld;    /* number of variables found by DESCRIBE */
  struct sqlvar   sqlvar[1]; /* array of variable descriptors */
};

#define SCALE(sqllen)        ((sqllen)/256)
#define PRECISION(sqllen)    ((sqllen)&0xff)
#define SET_PRECISION_SCALE(sqllen,precision,scale)    \
                sqllen = (scale)*256 + (precision)
#define DECIMALSTORAGE(sqllen)  (PRECISION(sqllen)/2 + 1)
typedef struct sqlda    SQLDA;
typedef struct sqlvar   SQLVAR, SQLDA_VARIABLE;
typedef struct sqlname  SQLNAME, SQLDA_NAME;
#ifndef SQLDASIZE
#define SQLDASIZE(n)  ( sizeof( struct sqlda ) +  \
              (n-1) * sizeof( struct sqlvar) )
#endif
#if defined( _SQL_PACK_STRUCTURES )
  #include "poppk.h"
  #if defined( _MSC_VER ) && _MSC_VER > 800
    #pragma warning(pop)
  #endif
#endif
#endif
```

SQLDA 字段

SQLDA 的各字段的含义如下:

字段	说明
sqldaid	8 字节字符字段，包含用于标识 SQLCA 结构的字符串 SQLDA。在您查看内存内容时，该字段可帮助进行调试。
sqldabc	包含 SQLDA 结构长度的 32 位整数。
sqln	sqlvar 数组中包含的变量描述符数。
sqld	有效的变量描述符（包含说明主机变量的信息）的数目。此字段由 DESCRIBE 语句设置。也可在向数据库服务器提供数据时设置它。
sqlvar	一组类型为 struct sqlvar 的描述符，每个描述符描述一个主机变量。

SQLDA 主机变量说明

SQLDA 中的每个 sqlvar 结构都说明一个主机变量。sqlvar 结构的各字段的含义如下:

- **sqltype** – 此描述符描述的变量的类型。

  低序位指示是否允许使用 NULL。有效的类型和常量定义可以在 sqldef.h 头文件中找到。

此字段由 DESCRIBE 语句填充。在向数据库服务器提供数据或从中检索数据时，您可以将此字段设置为任何类型。任何必要的类型转换都会自动完成。

- **sqllen** – 变量的长度。sqllen 值含有 a_sql_len 类型。该长度的实际含义取决于类型信息和 SQLDA 的使用方式。

  对于 LONG VARCHAR、LONG NVARCHAR 和 LONG BINARY 数据类型，使用 DT_LONGVARCHAR、DT_LONGNVARCHAR 或 DT_LONGBINARY 数据类型结构的 array_len 字段，而不是使用 sqllen 字段。
- **sqldata** – 指向此变量所占用的内存的指针。此内存必须对应于 sqltype 和 sqllen 字段。

  对于 UPDATE 和 INSERT 语句，如果 sqldata 指针为空指针，操作中就不会涉及此变量。对于 FETCH，如果 sqldata 指针是空指针，则不返回数据。换句话说，sqldata 指针返回的列是*未绑定列*。

  如果 DESCRIBE 语句使用 LONG NAMES，则此字段保存结果集列的长名称。另外，如果 DESCRIBE 语句是 DESCRIBE USER TYPES 语句，则此字段保存用户定义的数据类型的长名称，而不是列的长名称。如果该类型是基类型，则此字段为空。
- **sqlind** – 指向指示符值的指针。指示符值含有 a_sql_len 类型。负的指示符值表示 NULL。正的指示符值表示此变量已经被 FETCH 语句截断，且指示符值包含截断前的数据长度。如果将 conversion_error 数据库选项设置为 Off，则值 -2 表示出现了转换错误。

  如果 sqlind 指针是空指针，则说明没有适用于此主机变量的指示符变量。

  DESCRIBE 语句也使用 sqlind 字段来指示参数类型。如果类型是用户定义的数据类型，则此字段将设置为 DT_HAS_USERTYPE_INFO。在此情况下，您可能希望执行 DESCRIBE USER TYPES 以获取有关用户定义的数据类型的信息。
- **sqlname** – 类似 VARCHAR 的结构，如下所示：

```
struct sqlname {
   short int  length;
   char  data[ SQL_MAX_NAME_LEN ];
};
```

  它由 DESCRIBE 语句填充，在其它情况下不使用它。对于以下两种格式的 DESCRIBE 语句，此字段具有不同的含义：

  - **SELECT LIST** – 名称数据缓冲区由 SELECT 列表中对应项的列标题填充。
  - **BIND VARIABLES** – 名称数据缓冲区由曾用作绑定变量的主机变量的名称填充，或者，如果使用了未命名的参数标记，则用 "?" 填充。

  在 DESCRIBE SELECT LIST 语句中，出现的任何指示符变量都会用一个指示 SELECT 列表项是否可更新的标志来填充。有关此标志的详细信息可以在 `sqldef.h` 头文件中找到。

  如果 DESCRIBE 语句是 DESCRIBE USER TYPES 语句，则此字段保存用户定义数据类型的长名称，而不是列的长名称。如果该类型是基类型，则此字段为空。

SQLDA sqllen 字段值

DESCRIBE 后、发送值时和检索数据时的 SQLDA sqllen 字段值

DESCRIBE 后的 SQLDA sqllen 字段值

DESCRIBE 语句获取有关主机变量的信息（这些主机变量是存储从数据库检索的数据或将数据传递到数据库时所必需的）。

下表为各种数据库类型指明由 DESCRIBE 语句返回的 sqllen 和 sqltype 结构成员的值（SELECT LIST 和 BIND VARIABLE DESCRIBE 语句）。对于用户定义的数据库数据类型，则对基类型进行说明。

您的程序可以使用从 DESCRIBE 返回的类型和长度，也可以使用另一种类型。数据库服务器在任意两种类型之间执行类型转换。由 sqldata 字段指向的内存必须对应于 sqltype 和 sqllen 字段。嵌入式 SQL 类型通过对 sqltype 和 DT_TYPES 进行按位“与”操作 (sqltype & DT_TYPES) 来获得。

数据库字段类型	返回的嵌入式 SQL 类型	说明时返回的长度（以字节为单位）
BIGINT	DT_BIGINT	8
BINARY(n)	DT_BINARY	n
BIT	DT_BIT	1
CHAR(n)	DT_FIXCHAR[1]	从数据库字符集转换为客户端的 CHAR 字符集时，为最大数据扩展的 n 倍。如果长度超过 32767 字节，则返回的嵌入式 SQL 类型是长度为 32767 字节的 DT_LONGVARCHAR。
CHAR(n CHAR)	DT_FIXCHAR[1]	客户端的 CHAR 字符集中最大字符长度的 n 倍如果长度超过 32767 字节，则返回的嵌入式 SQL 类型是长度为 32767 字节的 DT_LONGVARCHAR。
DATE	DT_DATE	最长的带格式字符串的长度
DECIMAL(p,s)	DT_DECIMAL	SQLDA 中长度字段的低位字节设置为 p，高位字节设置为 s。请参见 sqlda.h 中的 PRECISION 和 SCALE 宏。
DOUBLE	DT_DOUBLE	8
FLOAT	DT_FLOAT	4
INT	DT_INT	4
LONG BINARY	DT_LONGBINARY	32767

数据库字段类型	返回的嵌入式 SQL 类型	说明时返回的长度（以字节为单位）
LONG NVARCHAR	DT_LONGVARCHAR/ DT_LONGNVARCHAR[2]	32767
LONG VARCHAR	DT_LONGVARCHAR	32767
NCHAR(n)	DT_FIXCHAR/DT_ NFIXCHAR[2]	客户端的 NCHAR 字符集中最大字符长度的 n 倍如果长度超过 32767 字节，则返回的嵌入式 SQL 类型是长度为 32767 字节的 DT_LONGNVARCHAR。
NVARCHAR(n)	DT_VARCHAR/DT_ NVARCHAR[2]	客户端的 NCHAR 字符集中最大字符长度的 n 倍如果长度超过 32767 字节，则返回的嵌入式 SQL 类型是长度为 32767 字节的 DT_LONGNVARCHAR。
REAL	DT_FLOAT	4
SMALLINT	DT_SMALLINT	2
TIME	DT_TIME	最长的带格式字符串的长度
TIMESTAMP	DT_TIMESTAMP	最长的带格式字符串的长度
TINYINT	DT_TINYINT	1
UNSIGNED BIGINT	DT_UNSBIGINT	8
UNSIGNED INT	DT_UNSINT	4
UNSIGNED SMALLINT	DT_UNSSMALLINT	2
VARCHAR(n)	DT_VARCHAR[1]	从数据库字符集转换为客户端的 CHAR 字符集时，为最大数据扩展的 n 倍。如果长度超过 32767 字节，则返回的嵌入式 SQL 类型是长度为 32767 字节的 DT_LONGVARCHAR。
VARCHAR(n CHAR)	DT_VARCHAR[1]	客户端的 CHAR 字符集中最大字符长度的 n 倍如果长度超过 32767，则返回的嵌入式 SQL 类型是长度为 32767 的 DT_LONGVARCHAR。

[1] 如果客户端 CHAR 字符集中的最大字节长度超过 32767 字节，为 CHAR 和 VARCHAR 返回的类型可能是 DT_LONGVARCHAR。

[2] 如果客户端 NCHAR 字符集中的最大字节长度超过 32767 字节，为 NCHAR 和 NVARCHAR 返回的类型可能是 DT_LONGNVARCHAR。缺省情况下，NCHAR、NVARCHAR 和 LONG NVARCHAR 分别被描述为 DT_FIXCHAR、DT_VARCHAR 和 DT_LONGVARCHAR。如果调用了 db_change_nchar_charset 函数，则这些类型被分别描述为 DT_NFIXCHAR、DT_NVARCHAR 和 DT_LONGNVARCHAR。

发送值时的 SQLDA sqllen 字段值

下表指明当您向数据库服务器提供数据时指定 SQLDA 中值的长度的方式。

在这种情况下，只允许使用表中显示的数据类型。在向数据库提供信息时，将 DT_DATE、DT_TIME 和 DT_TIMESTAMP 类型视为与 DT_STRING 相同；值必须是具有适当日期或时间格式、且以空值终止的字符串。

嵌入式 SQL 数据类型	设置长度的程序操作
DT_BIGINT	不需要任何操作。
DT_BINARY(n)	采用 BINARY 结构中的字段的长度。
DT_BIT	不需要任何操作。
DT_DATE	由终止的空字节决定的长度。
DT_DOUBLE	不需要任何操作。
DT_FIXCHAR(n)	SQLDA 中的长度字段决定字符串的长度。
DT_FLOAT	不需要任何操作。
DT_INT	不需要任何操作。
DT_LONGBINARY	忽略长度字段。
DT_LONGNVARCHAR	忽略长度字段。
DT_LONGVARCHAR	忽略长度字段。
DT_NFIXCHAR(n)	SQLDA 中的长度字段决定字符串的长度。
DT_NSTRING	长度由终止的 \0 决定。如果 ansi_blanks 选项为 On 且数据库是以空白填充的，则 SQLDA 中的长度字段必须设置为包含该值的缓冲区的长度（至少为该值的长度加上终止空字符的空间）。
DT_NVARCHAR	采用 NVARCHAR 结构中的字段的长度。
DT_SMALLINT	不需要任何操作。

嵌入式 SQL 数据类型	设置长度的程序操作
DT_STRING	长度由终止的 \0 决定。如果 ansi_blanks 选项为 On 且数据库是以空白填充的，则 SQLDA 中的长度字段必须设置为包含该值的缓冲区的长度（至少为该值的长度加上终止空字符的空间）。
DT_TIME	由终止的空字节决定的长度。
DT_TIMESTAMP	由终止的空字节决定的长度。
DT_TIMESTAMP_STRUCT	不需要任何操作。
DT_UNSBIGINT	不需要任何操作。
DT_UNSINT	不需要任何操作。
DT_UNSSMALLINT	不需要任何操作。
DT_VARCHAR(n)	采用 VARCHAR 结构中的字段的长度。
DT_VARIABLE	由终止的 \0 决定的长度。

检索数据时的 SQLDA sqllen 字段值

下表指明在使用 SQLDA 从数据库中检索数据时长度字段的值。在检索数据时，从不修改 sqllen 字段。

在这种情况下，只允许使用表中显示的接口数据类型。在从数据库检索信息时，使用 DT_DATE、DT_TIME 和 DT_TIMESTAMP 数据类型与使用 DT_STRING 的方式是相同的。值会被设置为当前日期格式的字符串。

嵌入式 SQL 数据类型	在接收时程序必须将长度字段设置为	在读取值之后数据库返回长度信息的方式
DT_BIGINT	不需要任何操作。	不需要任何操作。
DT_BINARY(n)	BINARY 结构的最大长度 (n +2)。n 的最大值为 32765。	将 BINARY 结构的 len 字段设置为实际长度（以字节为单位）。
DT_BIT	不需要任何操作。	不需要任何操作。
DT_DATE	缓冲区的长度。	字符串末尾的空字节。
DT_DOUBLE	不需要任何操作。	不需要任何操作。
DT_FIXCHAR(n)	缓冲区的长度（以字节为单位）。n 的最大值为 32767。	通过填补空白至缓冲区的长度。
DT_FLOAT	不需要任何操作。	不需要任何操作。
DT_INT	不需要任何操作。	不需要任何操作。

嵌入式 SQL 数据类型	在接收时程序必须将长度字段设置为	在读取值之后数据库返回长度信息的方式
DT_LONGBINARY	忽略长度字段。	忽略长度字段。
DT_LONGNVARCHAR	忽略长度字段。	忽略长度字段。
DT_LONGVARCHAR	忽略长度字段。	忽略长度字段。
DT_NFIXCHAR(n)	缓冲区的长度（以字节为单位）。n 的最大值为 32767。	通过填补空白至缓冲区的长度。
DT_NSTRING	缓冲区的长度。	字符串末尾的空字节。
DT_NVARCHAR(n)	NVARCHAR 结构的最大长度 (n+2)。n 的最大值为 32765。	将 NVARCHAR 结构的 len 字段设置为字符串的实际长度（以字节为单位）。
DT_SMALLINT	不需要任何操作。	不需要任何操作。
DT_STRING	缓冲区的长度。	字符串末尾的空字节。
DT_TIME	缓冲区的长度。	字符串末尾的空字节。
DT_TIMESTAMP	缓冲区的长度。	字符串末尾的空字节。
DT_TIMESTAMP_STRUCT	不需要任何操作。	不需要任何操作。
DT_UNSBIGINT	不需要任何操作。	不需要任何操作。
DT_UNSINT	不需要任何操作。	不需要任何操作。
DT_UNSSMALLINT	不需要任何操作。	不需要任何操作。
DT_VARCHAR(n)	VARCHAR 结构的最大长度 (n+2)。n 的最大值为 32765。	将 VARCHAR 结构的 len 字段设置为字符串的实际长度（以字节为单位）。

如何使用嵌入式 SQL 读取数据

在嵌入式 SQL 中使用 SELECT 语句实现数据的读取。这包括两种情况：

- **最多返回一行的 SELECT 语句** – 使用 INTO 子句将返回的值直接指派给主机变量。
- **SELECT 语句可能返回多行** – 使用游标管理结果集的行。

最多返回一行的 SELECT 语句

单行查询最多从数据库中检索一行。单行查询 SELECT 语句在 SELECT 列表之后和 FROM 子句之前有一个 INTO 子句。INTO 子句包含一个主机变量的列表，用来接收每个 SELECT 列表项的值。主机变量和 SELECT 列表项的数目必须相同。主机变量可以和指示符变量一起使用，以指示 NULL 结果。

当执行 SELECT 语句时，数据库服务器检索结果并将其放在主机变量中。如果查询结果包含多个行，则数据库服务器会返回一个错误。

如果查询结果是没有选中任何行，则将返回一个错误，指出没有找到任何行 (SQLCODE 100)。在 SQLCA 结构中返回错误和警告。

示例

如果成功地从 Employees 表中读取了一行，则以下代码段返回 1；如果该行不存在，则返回 0；如果出现错误，则返回 -1。

```
EXEC SQL BEGIN DECLARE SECTION;
long      id;
char      name[41];
char      sex;
char      birthdate[15];
a_sql_len ind_birthdate;
EXEC SQL END DECLARE SECTION;
...
int find_employee( long employee_id )
{
  id = employee_id;
  EXEC SQL SELECT GivenName ||
    ' ' || Surname, Sex, BirthDate
    INTO :name, :sex,
      :birthdate:ind_birthdate
    FROM Employees
    WHERE EmployeeID = :id;
  if( SQLCODE == SQLE_NOTFOUND )
  {
    return( 0 ); /* employee not found */
  }
  else if( SQLCODE < 0 )
  {
    return( -1 ); /* error */
  }
  else
  {
    return( 1 ); /* found */
  }
}
```

嵌入式 SQL 中的游标

游标用于从其结果集中具有多个行的查询中检索行。游标是 SQL 查询的句柄或标识符，也是结果集中的一个位置。

嵌入式 SQL 中的游标管理涉及以下步骤：

1. 使用 DECLARE CURSOR 语句声明特定 SELECT 语句的游标。
2. 使用 OPEN 语句打开游标。
3. 使用 FETCH 语句一次一行地从游标中检索结果。
4. 一直读取行，直到返回 `Row Not Found` 警告。
   在 SQLCA 结构中返回错误和警告。
5. 使用 CLOSE 语句关闭游标。

缺省情况下，在事务（COMMIT 或 ROLLBACK 上）的结尾会自动关闭游标。用 WITH HOLD 子句打开的游标对于后续事务保持打开状态，直到它们被显式关闭。

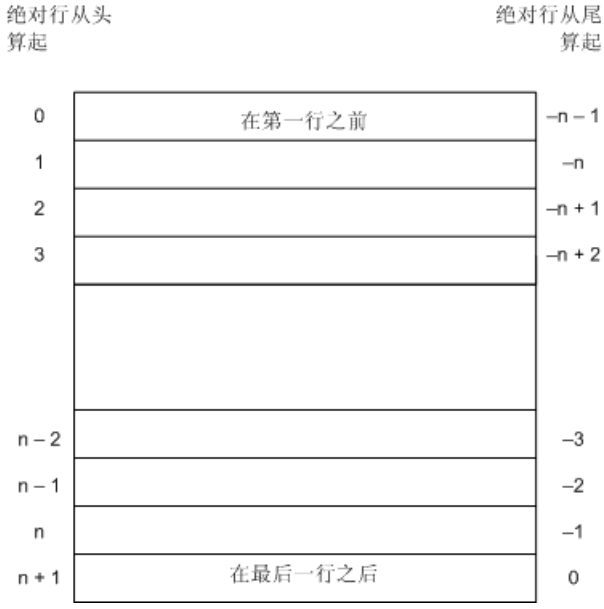
以下是游标用法的简单示例：

```
void print_employees( void )
{
  EXEC SQL BEGIN DECLARE SECTION;
  char      name[50];
  char      sex;
  char      birthdate[15];
  a_sql_len ind_birthdate;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL DECLARE C1 CURSOR FOR
    SELECT GivenName || ' ' || Surname, Sex, BirthDate FROM Employees;
  EXEC SQL OPEN C1;
  for( ;; )
  {
    EXEC SQL FETCH C1 INTO :name, :sex, :birthdate:ind_birthdate;
    if( SQLCODE == SQLE_NOTFOUND )
    {
      break;
    }
    else if( SQLCODE < 0 )
    {
      break;
    }

    if( ind_birthdate < 0 )
    {
      strcpy( birthdate, "UNKNOWN" );
    }
    printf( "Name: %s Sex: %c Birthdate: %s\n", name, sex,
birthdate );
  }
  EXEC SQL CLOSE C1;
}
```

游标定位
游标定位在以下三个位置之一：

- 在一行上
- 在第一行之前
- 在最后一行之后



当游标打开之后，它位于第一行之前。可使用 FETCH 语句来移动游标位置。可以将游标定位到相对查询结果开头或结尾的一个绝对位置。也可以相对当前游标位置移动它。

UPDATE 和 DELETE 语句有特殊的*定位*版本，可用于更新或删除游标当前位置处的行。如果游标位于第一行之前或最后一行之后，则会返回错误，指示游标中没有相应的行。

可以使用 PUT 语句向游标中插入行。

游标定位问题
插入 DYNAMIC SCROLL 游标并对其进行某些更新会导致与游标定位有关的问题。除非 SELECT 语句中有 ORDER BY 子句，否则数据库服务器不将插入的行放在游标内可预知的位置。有时，插入的行要等到关闭并再次打开游标后才会出现。

对于 SAP Sybase IQ，如果必须创建临时表才能打开游标，则会出现这种情况。

UPDATE 语句可导致行在游标中移动。如果游标具有一个使用现有索引（未创建临时表）的 ORDER BY 子句，则会出现这种情况。

宽读取或数组读取

可以将 FETCH 语句修改为一次读取多行，这样可能会改善性能。这种方式称为*宽读取*或*数组读取*。

SAP Sybase IQ 还支持宽放置和宽插入。

要在嵌入式 SQL 中使用宽读取，请将 FETCH 语句包括在代码中，如下所示：

```
EXEC SQL FETCH ... ARRAY nnn
```

其中 ARRAY *nnn* 是 FETCH 语句的最后一项。读取计数 *nnn* 可以是一个主机变量。SQLDA 中的变量数必须是 *nnn* 和每行的列数的乘积。第一行放在 SQLDA 变量 0 和（每行的列数）-1 之间，依此类推。

SQLDA 的每一行中的每一列的类型必须相同，否则会返回 SQLDA_INCONSISTENT 错误。

服务器在 SQLCOUNT 中返回读取的记录数，除非有错误或警告，否则该记录数始终大于零。在宽读取时，在没有错误的情况下，SQLCOUNT 为 1 指示已经读取一个有效行。

示例

下面的示例代码说明如何使用宽读取。也可以在 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\esqlwidefetch\widefetch.sqc` 中找到此代码。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqldef.h"
EXEC SQL INCLUDE SQLCA;

EXEC SQL WHENEVER SQLERROR { PrintSQLError();
        goto err; };

static void PrintSQLError()
{
  char buffer[200];

  printf( "SQL error %d -- %s\n",
     SQLCODE,
     sqlerror_message( &sqlca,
              buffer,
              sizeof( buffer ) ) );
}
static SQLDA * PrepareSQLDA(
  a_sql_statement_number stat0,
  unsigned  width,
  unsigned  *cols_per_row )

/* Allocate a SQLDA to be used for fetching from
   the statement identified by "stat0". "width"
   rows are retrieved on each FETCH request.
```

```
   The number of columns per row is assigned to
   "cols_per_row". */
{
  int           num_cols;
  unsigned      row, col, offset;
  SQLDA *       sqlda;
  EXEC SQL BEGIN DECLARE SECTION;
  a_sql_statement_number  stat;
  EXEC SQL END DECLARE SECTION;
  stat = stat0;
  sqlda = alloc_sqlda( 100 );
  if( sqlda == NULL ) return( NULL );
  EXEC SQL DESCRIBE :stat INTO sqlda;
  *cols_per_row = num_cols = sqlda->sqld;
  if( num_cols * width > sqlda->sqln )
  {
    free_sqlda( sqlda );
    sqlda = alloc_sqlda( num_cols * width );
    if( sqlda == NULL ) return( NULL );
    EXEC SQL DESCRIBE :stat INTO sqlda;
  }
  // copy first row in SQLDA setup by describe
  // to following (wide) rows
  sqlda->sqld = num_cols * width;
  offset = num_cols;
  for( row = 1; row < width; row++ )
  {
    for( col = 0;
      col < num_cols;
      col++, offset++ )
    {
      sqlda->sqlvar[offset].sqltype =
          sqlda->sqlvar[col].sqltype;
      sqlda->sqlvar[offset].sqllen =
          sqlda->sqlvar[col].sqllen;
      // optional: copy described column name
      memcpy( &sqlda->sqlvar[offset].sqlname,
              &sqlda->sqlvar[col].sqlname,
              sizeof( sqlda->sqlvar[0].sqlname ) );
    }
  }
  fill_s_sqlda( sqlda, 40 );
  return( sqlda );
err:
  return( NULL );
}
static void PrintFetchedRows(
  SQLDA * sqlda,
  unsigned cols_per_row )
{
  /* Print rows already wide fetched in the SQLDA */
  long      rows_fetched;
  int       row, col, offset;

  if( SQLCOUNT == 0 )
  {
```

```
    rows_fetched = 1;
  }
  else
  {
    rows_fetched = SQLCOUNT;
  }
  printf( "Fetched %d Rows:\n", rows_fetched );
  for( row = 0; row < rows_fetched; row++ )
  {
    for( col = 0; col < cols_per_row; col++ )
    {
      offset = row * cols_per_row + col;
      printf( " \"%s\"",
        (char *)sqlda->sqlvar[offset].sqldata );
     }
     printf( "\n" );
  }
}
static int DoQuery(
  char * query_str0,
  unsigned fetch_width0 )
{
  /* Wide Fetch "query_str0" select statement
   * using a width of "fetch_width0" rows" */
  SQLDA *                 sqlda;
  unsigned                cols_per_row;
  EXEC SQL BEGIN DECLARE SECTION;
  a_sql_statement_number  stat;
  char *                  query_str;
  unsigned                fetch_width;
  EXEC SQL END DECLARE SECTION;

  query_str = query_str0;
  fetch_width = fetch_width0;

  EXEC SQL PREPARE :stat FROM :query_str;
  EXEC SQL DECLARE QCURSOR CURSOR FOR :stat
       FOR READ ONLY;
  EXEC SQL OPEN QCURSOR;
  sqlda = PrepareSQLDA( stat,
    fetch_width,
    &cols_per_row );
  if( sqlda == NULL )
  {
    printf( "Error allocating SQLDA\n" );
    return( SQLE_NO_MEMORY );
  }
  for( ;; )
  {
    EXEC SQL FETCH QCURSOR INTO DESCRIPTOR sqlda
      ARRAY :fetch_width;
    if( SQLCODE != SQLE_NOERROR ) break;
    PrintFetchedRows( sqlda, cols_per_row );
  }
  EXEC SQL CLOSE QCURSOR;
  EXEC SQL DROP STATEMENT :stat;
```

```
  free_filled_sqlda( sqlda );
err:
  return( SQLCODE );
}
void main( int argc, char *argv[] )
{
  /* Optional first argument is a select statement,
   * optional second argument is the fetch width */
  char *query_str =
    "SELECT GivenName, Surname FROM Employees";
  unsigned fetch_width = 10;

  if( argc > 1 )
  {
    query_str = argv[1];
    if( argc > 2 )
    {
      fetch_width = atoi( argv[2] );
      if( fetch_width < 2 )
      {
        fetch_width = 2;
      }
    }
  }
  db_init( &sqlca );
  EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";

  DoQuery( query_str, fetch_width );

  EXEC SQL DISCONNECT;
err:
  db_fini( &sqlca );
}
```

有关使用宽读取的注意事项

- 在函数 PrepareSQLDA 中，SQLDA 内存是使用 alloc_sqlda 函数分配的。这样就为指示符变量留出了空间，而不用使用 alloc_sqlda_noind 函数。
- 如果读取的行数小于请求的行数，但又不是零（例如在游标的末尾），则通过设置指示符值可将对应于未读取的行的 SQLDA 项作为 NULL 返回。如果没有指示符变量，将生成错误（SQLE_NO_INDICATOR：未给 NULL 结果提供指示符变量）。
- 如果正在读取的行已经更新，并且生成了 SQLE_ROW_UPDATED_WARNING 警告，那么，读取到导致警告的行时就会停止。将返回处理到该点的所有行（包括导致警告的行）的值。SQLCOUNT 包含读取的行数，其中包括导致警告的行。所有剩余的 SQLDA 项都被标记为 NULL。
- 如果正在读取的行已经被删除或锁定，并且生成了 SQLE_NO_CURRENT_ROW 或 SQLE_LOCKED 错误，则 SQLCOUNT 包含出错前读取的行数。这不包括导致错误的行。SQLDA 不包含任何行的值，因为出现错误时不返回 SQLDA 值。如果必要，可使用 SQLCOUNT 值来重新定位游标，以便读取行。

如何使用嵌入式 SQL 发送和检索长整型值

在嵌入式 SQL 应用程序中，发送和检索 LONG VARCHAR、LONG NVARCHAR 和 LONG BINARY 值的方法与发送和检索其它数据类型的值的方法不同。标准 SQLDA 字段包含的数据不得超过 32767 字节，因为保存长度信息的字段（sqllen、*sqlind）是 16 位的值。将这些值更改为 32 位值会破坏现有的应用程序。

说明 LONG VARCHAR、LONG NVARCHAR 和 LONG BINARY 值的方法与说明其它数据类型的值的方法相同。

静态 SQL 结构
使用单独的字段来保存 LONG BINARY、LONG VARCHAR 和 LONG NVARCHAR 数据类型的已分配长度、已存储长度和未截断长度。静态 SQL 数据类型在 `sqlca.h` 中定义如下：

```
#define DECL_LONGVARCHAR( size )          \
  struct { a_sql_uint32    array_len;     \
           a_sql_uint32    stored_len;    \
           a_sql_uint32    untrunc_len;   \
           char            array[size+1];\
         }
#define DECL_LONGNVARCHAR( size )         \
  struct { a_sql_uint32    array_len;     \
           a_sql_uint32    stored_len;    \
           a_sql_uint32    untrunc_len;   \
           char            array[size+1];\
         }
#define DECL_LONGBINARY( size )           \
  struct { a_sql_uint32    array_len;     \
           a_sql_uint32    stored_len;    \
           a_sql_uint32    untrunc_len;   \
           char            array[size];   \
         }
```

动态 SQL 结构
对于动态 SQL，根据需要将 sqltype 字段设置为 DT_LONGVARCHAR、DT_LONGNVARCHAR 或 DT_LONGBINARY。关联的 LONGVARCHAR、LONGNVARCHAR 和 LONGBINARY 结构如下：

```
typedef struct LONGVARCHAR {
    a_sql_uint32    array_len;
    a_sql_uint32    stored_len;
    a_sql_uint32    untrunc_len;
    char            array[1];
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

结构成员定义
对于静态和动态 SQL 结构，结构成员定义如下：

- **array_len** – （发送和检索。）为结构中数组部分分配的字节数。
- **stored_len** – （发送和检索。）数组中存储的字节数。总是小于或等于 array_len 和 untrunc_len。
- **untrunc_len** – （仅检索。）值不被截断的情况下数组中将存储的字节数。总是大于或等于 stored_len。如果发生截断，则此值大于 array_len。

使用静态 SQL 检索 LONG 数据

使用静态 SQL 检索 LONG VARCHAR、LONG NVARCHAR 或 LONG BINARY 值。

前提条件

执行此任务没有前提条件。

过程

1. 根据需要声明类型为 DECL_LONGVARCHAR、DECL_LONGNVARCHAR 或 DECL_LONGBINARY 的主机变量。array_len 成员将自动填充。
2. 使用 FETCH、GET DATA 或 EXECUTE INTO 检索数据。SAP Sybase IQ 会设置以下信息：
   - **指示符变量** – 指示符变量在值为 NULL 时为负，在未发生截断时为 0，在发生截断时为未截断值的字节数（不超过 32767 的正数）。
   - **stored_len** – 数组中存储的字节数。总是小于或等于 array_len 和 untrunc_len。
   - **untrunc_len** – 值不被截断的情况下数组中将存储的字节数。总是大于或等于 stored_len。如果发生截断，则此值大于 array_len。

使用静态 SQL 检索 LONG 数据。

使用动态 SQL 检索 LONG 数据

使用动态 SQL 检索 LONG VARCHAR、LONG NVARCHAR 或 LONG BINARY 值。

前提条件

执行此任务没有前提条件。

过程

1. 根据需要将 sqltype 字段设置为 DT_LONGVARCHAR、DT_LONGNVARCHAR 或 DT_LONGBINARY。
2. 将 sqldata 字段设置为指向 LONGVARCHAR、LONGNVARCHAR 或 LONGBINARY 主机变量结构。

可以使用 LONGVARCHARSIZE (*n*)、LONGNVARCHARSIZE (*n*) 或 LONGBINARYSIZE (*n*) 宏来确定在数组字段中容纳 *n* 字节的数据而要分配的总字节数。

3. 将主机变量结构的 array_len 字段设置为分配给数组字段的字节数。
4. 使用 FETCH、GET DATA 或 EXECUTE INTO 检索数据。SAP Sybase IQ 会设置以下信息：
   - *** sqlind** – sqlda 字段在值为 NULL 时为负，在未发生截断时为 0，在发生截断时为未截断值的字节数（不超过 32767 的正数）。
   - **stored_len** – 数组中存储的字节数。总是小于或等于 array_len 和 untrunc_len。
   - **untrunc_len** – 值不被截断的情况下数组中将存储的字节数。总是大于或等于 stored_len。如果发生截断，则此值大于 array_len。

使用动态 SQL 检索 LONG 数据。

使用静态 SQL 发送 LONG 数据

使用嵌入式 SQL 应用程序中的静态 SQL 将 LONG 值发送到数据库。

前提条件

执行此任务没有前提条件。

过程

1. 根据需要声明类型为 DECL_LONGVARCHAR、DECL_LONGNVARCHAR 或 DECL_LONGBINARY 的主机变量。
2. 如果您要发送 NULL，请将指示符变量设置为负值。
3. 将主机变量结构的 stored_len 字段设置为数组字段中数据的字节数。
4. 通过打开游标或执行语句发送数据。

嵌入式 SQL 应用程序可随时将 LONG 值发送到数据库。

使用动态 SQL 发送 LONG 数据

使用嵌入式 SQL 应用程序中的动态 SQL 将 LONG 值发送到数据库。

前提条件

执行此任务没有前提条件。

过程

1. 根据需要将 sqltype 字段设置为 DT_LONGVARCHAR、DT_LONGNVARCHAR 或 DT_LONGBINARY。

2. 如果您要发送 NULL，请将 `* sqlind` 设置为负值。
3. 如果不是要发送 NULL，请将 sqldata 字段设置为指向 LONGVARCHAR、LONGNVARCHAR 或 LONGBINARY 主机变量结构。

   可以使用 LONGVARCHARSIZE (*n*)、LONGNVARCHARSIZE (*n*) 或 LONGBINARYSIZE (*n*) 宏来确定在数组字段中容纳 *n* 字节的数据而要分配的总字节数。
4. 将主机变量结构的 array_len 字段设置为分配给数组字段的字节数。
5. 将主机变量结构的 stored_len 字段设置为数组字段中数据的字节数。它必须小于或等于 array_len。
6. 通过打开游标或执行语句发送数据。

嵌入式 SQL 应用程序可随时将 LONG 值发送到数据库。

嵌入式 SQL 中的简单存储过程

您可以在嵌入式 SQL 中创建和调用存储过程。

您可以像嵌入任何其它数据定义语句（如 CREATE TABLE）那样嵌入 CREATE PROCEDURE。您还可以嵌入 CALL 语句以执行存储过程。下面的代码段说明如何在嵌入式 SQL 中创建和执行存储过程：

```
EXEC SQL CREATE PROCEDURE pettycash(
 IN Amount DECIMAL(10,2) )
BEGIN
 UPDATE account
 SET balance = balance - Amount
 WHERE name = 'bank';

 UPDATE account
 SET balance = balance + Amount
 WHERE name = 'pettycash expense';
END;
EXEC SQL CALL pettycash( 10.72 );
```

要将主机变量值传递到存储过程，或者要检索输出变量，请准备并执行 CALL 语句。下面的代码段说明主机变量的用法。USING 和 INTO 子句都在 EXECUTE 语句中使用。

```
EXEC SQL BEGIN DECLARE SECTION;
double  hv_expense;
double  hv_balance;
EXEC SQL END DECLARE SECTION;

// Code here
EXEC SQL CREATE PROCEDURE pettycash(
   IN expense  DECIMAL(10,2),
   OUT endbalance DECIMAL(10,2) )
BEGIN
  UPDATE account
  SET balance = balance - expense
```

```
  WHERE name = 'bank';
  UPDATE account
  SET balance = balance + expense
  WHERE name = 'pettycash expense';

  SET endbalance = ( SELECT balance FROM account
         WHERE name = 'bank' );
END;

EXEC SQL PREPARE S1 FROM 'CALL pettycash( ?, ? )';
EXEC SQL EXECUTE S1 USING :hv_expense INTO :hv_balance;
```

具有结果集的存储过程

数据库过程也可以包含 SELECT 语句。声明该过程的方法是这样的：使用 RESULT 子句来指定结果集中列的编号、名称和类型。结果集列与输出参数不同。对于具有结果集的过程，在游标声明中可以使用 CALL 语句代替 SELECT 语句：

```
EXEC SQL BEGIN DECLARE SECTION;
 char hv_name[100];
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE PROCEDURE female_employees()
  RESULT( name char(50) )
BEGIN
  SELECT GivenName || Surname FROM Employees
  WHERE Sex = 'f';
END;

EXEC SQL PREPARE S1 FROM 'CALL female_employees()';

EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
for(;;)
{
  EXEC SQL FETCH C1 INTO :hv_name;
  if( SQLCODE != SQLE_NOERROR ) break;
  printf( "%s\n", hv_name );
}
EXEC SQL CLOSE C1;
```

在本示例中，使用 OPEN 语句而不是 EXECUTE 语句调用该过程。OPEN 语句会使该过程在到达 SELECT 语句之前一直执行。此时，C1 是数据库过程内的 SELECT 语句的游标。您可以使用所有形式的 FETCH 语句（向后和向前滚动），直到其完成为止。CLOSE 语句用于终止该过程的执行。

如果在该过程中在 SELECT 语句之后还有一条语句，则不会执行该语句。要执行 SELECT 之后的语句，请使用 RESUME cursor-name 语句。RESUME 语句可返回警告 SQLE_PROCEDURE_COMPLETE，或返回 SQLE_NOERROR 指示存在另一游标。下面的示例说明了一个双选择过程：

```
EXEC SQL CREATE PROCEDURE people()
  RESULT( name char(50) )
BEGIN
```

```
 SELECT GivenName || Surname
 FROM Employees;

 SELECT GivenName || Surname
 FROM Customers;
END;

EXEC SQL PREPARE S1 FROM 'CALL people()';
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
while( SQLCODE == SQLE_NOERROR )
{
  for(;;)
  {
    EXEC SQL FETCH C1 INTO :hv_name;
    if( SQLCODE != SQLE_NOERROR ) break;
    printf( "%s\n", hv_name );
  }
  EXEC SQL RESUME C1;
}
EXEC SQL CLOSE C1;
```

CALL 语句的动态游标

以上这些示例使用了静态游标。也可以将完全动态的游标用于 CALL 语句。

DESCRIBE 语句完全适用于过程调用。DESCRIBE OUTPUT 生成的 SQLDA 具有每个结果集列的说明。

如果过程没有结果集，则 SQLDA 具有过程的每个 INOUT 或 OUT 参数的说明。DESCRIBE INPUT 语句生成的 SQLDA 具有过程的每个 IN 或 INOUT 参数的说明。

DESCRIBE ALL

DESCRIBE ALL 说明 IN、INOUT、OUT 和 RESULT 等集合参数。DESCRIBE ALL 使用 SQLDA 中的指示符变量提供其它信息。

当描述 CALL 语句时，会在指示符变量中设置 DT_PROCEDURE_IN 和 DT_PROCEDURE_OUT 位。DT_PROCEDURE_IN 指示 IN 或 INOUT 参数，而 DT_PROCEDURE_OUT 则指示 INOUT 或 OUT 参数。过程的 RESULT 列则清除这两个位。

在 DESCRIBE OUTPUT 之后，可以使用这些位来区分具有结果集的语句（需要使用 OPEN、FETCH、RESUME 和 CLOSE）和不具有结果集的语句（需要使用 EXECUTE）。

多个结果集

如果具有一个返回多个结果集的过程，则在结果集改变形状时必须在每条 RESUME 语句之后重新说明。

您需要说明游标（而不是语句），以重新说明游标的当前位置。

使用嵌入式 SQL 管理请求

由于典型的嵌入式 SQL 应用程序必须等待每个数据库请求完成，然后才能执行下一步，因此，使用多个执行线程的应用程序能够继续执行其它任务。

如果必须使用单个执行线程，那么通过使用带 DB_CALLBACK_WAIT 选项的 db_register_a_callback 函数注册一个回调函数，可以实现一定程度的多任务功能。在数据库服务器或客户端库忙于处理您的数据库请求的同时，接口库将反复调用您的回调函数。

在回调函数中，不能启动另外的数据库请求，但是可以使用 db_cancel_request 函数取消当前请求。您可以在消息处理程序中使用 db_is_working 函数来确定是否有正在进行的数据库请求。

使用嵌入式 SQL 备份数据库

建议使用 **BACKUP DATABASE** 语句备份数据库。

db_backup 函数提供了另外一种在嵌入式 SQL 应用程序中执行联机备份的方法。SAP Sybase IQ 实用程序也利用了此函数。

您也可以使用数据库工具 DBBackup 函数直接访问 SAP Sybase IQ Backup 实用程序。

只有在任何其它备份方法都无法满足备份要求时，才应使用 db_backup 函数编写程序。

库函数参考

SQL 预处理器生成对接口库或 DLL 中的函数的调用。除了 SQL 预处理器生成的调用外，还提供了一组库函数方便数据库操作。EXEC SQL INCLUDE SQLCA 语句包括这些函数的原型。

本节包含这些不同函数的参考说明。

DLL 入口点
除了原型具有适合于 DLL 的修改程序外，DLL 入口点都是相同的。

可以使用 `sqlca.h` 中定义的 `_esqlentry_` 以可移植的方式声明入口点。将解析为值 __stdcall。

alloc_sqlda 函数

分配一个具有 *numvar* 个变量描述符的 SQLDA。

语法

```
struct sqlda * alloc_sqlda( unsigned numvar );
```

参数

- **numvar** – 要分配的变量描述符数。

返回值
如果成功则为指向 SQLDA 的指针；如果没有足够的可用内存则返回空指针。

注释
分配一个具有 *numvar* 个变量描述符的 SQLDA。将该 SQLDA 的 sqln 字段初始化为 *numvar*。为指示符变量分配空间，将指示符变量设置为指向此空间，并将指示符值初始化为零。如果无法分配内存，则返回空指针。建议您使用此函数代替 alloc_sqlda_noind 函数。

alloc_sqlda_noind 函数

分配一个具有 *numvar* 个变量描述符的 SQLDA。

语法

```
struct sqlda * alloc_sqlda_noind( unsigned numvar );
```

参数

- **numvar** – 要分配的变量描述符数。

返回值
如果成功则为指向 SQLDA 的指针；如果没有足够的可用内存则返回空指针。

注释
分配一个具有 *numvar* 个变量描述符的 SQLDA。将该 SQLDA 的 sqln 字段初始化为 *numvar*。不为指示符变量分配空间；将指示符指针设置为空指针。如果无法分配内存，则返回空指针。

db_backup 函数

尽管此函数提供了一种向应用程序添加备份功能的方法，但建议您使用 **BACKUP DATABASE** 语句来完成此任务。

语法

```
void db_backup(
SQLCA * sqlca,
int op,
int file_num,
unsigned long page_num,
struct sqlda * sqlda);
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **op** – 要执行的动作或操作。
- **file_num** – 数据库的文件数。
- **page_num** – 数据库的页数。0 至最大页数减 1 范围内的一个值。
- **sqlda** – 指向 SQLDA 结构的指针。

Authorization

必须以具有 BACKUP DATABASE 系统特权的用户身份进行连接，或者具有 SYS_RUN_REPLICATION_ROLE 系统角色。

注释

尽管此函数提供了一种向应用程序添加备份功能的方法，但建议您使用 **BACKUP DATABASE** 语句来完成此任务。

执行的操作取决于 *op* 参数的值：

- **DB_BACKUP_START** – 必须先调用此函数，然后才能开始备份。对于任何给定的数据库服务器，一个数据库一次只能运行一个备份。在备份完成之前禁用数据库检查点（直到使用 *op* 值 DB_BACKUP_END 调用 db_backup）。如果备份无法启动，则 SQLCODE 为 SQLE_BACKUP_NOT_STARTED。否则，将 *sqlca* 的 SQLCOUNT 字段设置为数据库页的大小。一次一页地对备份进行处理。

  忽略 *file_num*、*page_num* 和 *sqlda* 参数。
- **DB_BACKUP_OPEN_FILE** – 打开由 *file_num* 指定的数据库文件，这样便可使用 DB_BACKUP_READ_PAGE 对指定文件的页进行备份。根数据库文件和事务日志文件的有效文件编号分别为 0 到 DB_BACKUP_MAX_FILE 和 0 到 DB_BACKUP_TRANS_LOG_FILE。如果指定的文件不存在，则 SQLCODE 为 SQLE_NOTFOUND。否则，SQLCOUNT 包含文件中的页数，SQLIOESTIMATE 包含一个标识数据库文件创建时间的 32 位值 (POSIX time_t)，操作系统文件名位于 SQLCA 的 *sqlerrmc* 字段中。

忽略参数 *page_num* 和 *sqlda*。

- **DB_BACKUP_READ_PAGE** – 读取由 *file_num* 指定的数据库文件的一页。使用 DB_BACKUP_OPEN_FILE 操作对 db_backup 调用成功后会在 SQLCOUNT 中返回一个页数，*page_num* 值应介于 0 到此页数减去一得到的数之间。否则，SQLCODE 将设置为 SQLE_NOTFOUND。*sqlda* 描述符应使用一个指向缓冲区的 DT_BINARY 或 DT_LONG_BINARY 类型的变量建立。使用 DB_BACKUP_START 操作调用 db_backup 时会在 SQLCOUNT 字段中返回一个大小值，该缓冲区应足以保存这一大小的二进制数据。

  DT_BINARY 数据包含一个后跟实际二进制数据的两字节长度值，因此缓冲区必须比页大小大两个字节。

  注意： 此调用会在缓冲区中制作指定数据库页的一个副本，但应由应用程序将缓冲区保存到某种备份介质。

- **DB_BACKUP_READ_RENAME_LOG** – 此操作与 DB_BACKUP_READ_PAGE 只有一点不同：事务日志的最后一页返回之后，数据库服务器会重命名事务日志并启动一个新的事务日志。

  如果数据库服务器无法在当前时间重命名日志（例如，在 7.0.x 版或更早版本的数据库中可能存在未完成的事务），则会设置 SQLE_BACKUP_CANNOT_RENAME_LOG_YET 错误。在这种情况下，不要使用返回的页，而是要重新发出请求，直到收到 SQLE_NOERROR，然后写入页。继续读取页，直到收到 SQLE_NOTFOUND 条件。

  可能会在多个页上多次返回 SQLE_BACKUP_CANNOT_RENAME_LOG_YET 错误。您应该在重试循环中增加延迟，以便不会因请求过多而降低服务器的速度。

  当您收到 SQLE_NOTFOUND 条件时，事务日志已经成功备份且文件已经重命名。旧日志文件的名称在 SQLCA 的 *sqlerrmc* 字段中返回。

  您应在 db_backup 调用后检查 sqlda->sqlvar[0].sqlind 的值。如果此值大于零，则最后一个日志页已经写入且日志文件已经重命名。新名称仍然在 sqlca.sqlerrmc 中，但 SQLCODE 值是 SQLE_NOERROR。

  此后，您不应再次调用 db_backup（除非要关闭文件并完成备份），否则，您会获得备份日志文件的第二份副本并收到 SQLE_NOTFOUND。

- **DB_BACKUP_CLOSE_FILE** – 处理完一个文件后必须调用此函数以关闭由 *file_num* 指定的数据库文件。

  忽略参数 *page_num* 和 *sqlda*。

- **DB_BACKUP_END** – 备份结束时必须调用此函数。在此备份结束之前，任何其它备份都无法启动。会再次启用检查点。

  忽略 *file_num*、*page_num* 和 *sqlda* 参数。

- **DB_BACKUP_PARALLEL_START** – 开始并行备份。与 DB_BACKUP_START 类似，对于任何给定的数据库服务器，一个数据库一次只能运行一个备份。在备份完成之前禁用数据库检查点（直到使用 *op* 值 DB_BACKUP_END 调用

db_backup）。如果备份无法启动，则将收到 SQLE_BACKUP_NOT_STARTED。否则，将 sqlca 的 SQLCOUNT 字段设置为数据库页的大小。

file_num 参数指示数据库服务器在事务日志的最后一页返回之后重命名事务日志并启动一个新的日志。如果此值非零，则将重命名或重新启动事务日志。否则，不会重命名或重新启动事务日志。使用此此参数后将不必执行 DB_BACKUP_READ_RENAME_LOG 操作，并行备份操作期间不允许进行该操作。

page_num 参数告知数据库服务器客户端缓冲区的最大大小（以数据库页为单位）。在服务器端，并行备份读取程序会尝试读取连续页块 - 此值可告知服务器应为这些块分配多大的内存空间：传递 *nnn* 值可告知服务器：客户端一次将从服务器接收最多 *nnnn* 个数据库页。如果服务器无法为 *nnn* 页的块分配足够内存，服务器可能会返回小于 *nnn* 的页块。如果客户端在调用 DB_BACKUP_PARALLEL_START 之前不清楚数据库页的大小，则可通过 DB_BACKUP_INFO 操作将此值提供给服务器。必须在首次调用前提供此值，以便检索备份页 (DB_BACKUP_PARALLEL_READ)。

注意： 如果使用 db_backup 开始并行备份，db_backup 不会创建写入程序线程。db_backup 的调用程序必须接收数据并充当写入程序。

- **DB_BACKUP_INFO** – 此参数为数据库服务器提供关于并行备份的其它信息。*file_num* 参数指示所提供信息的类型，*page_num* 参数提供值。您可以使用 DB_BACKUP_INFO 指定以下其它信息：
  - **DB_BACKUP_INFO_PAGES_IN_BLOCK** – *page_num* 参数包含应在同一块内发回的最大页数。
  - **DB_BACKUP_INFO_CHKPT_LOG** – 这是相当于 **BACKUP DATABASE** 语句的 WITH CHECKPOINT LOG 选项的客户端选项。*page_num* 的值 DB_BACKUP_CHKPT_COPY 指示 COPY，而值 DB_BACKUP_CHKPT_NOCOPY 指示 NO COPY。如果未提供此值，则缺省为 COPY。
- **DB_BACKUP_PARALLEL_READ** – 此操作从数据库服务器上读取页块。调用此操作前，使用 DB_BACKUP_OPEN_FILE 操作打开所有要备份的文件。DB_BACKUP_PARALLEL_READ 会忽略 *file_num* 和 *page_num* 参数。

  sqlda 描述符应使用一个指向缓冲区的 DT_LONGBINARY 类型的变量建立。缓冲区大小应足以保存 *nnn* 页（在 DB_BACKUP_START_PARALLEL 操作或 DB_BACKUP_INFO 操作中指定）大小的二进制数据。

  服务器为特定数据库文件返回连续数据库页块。块内第一页的页码在 SQLCOUNT 字段中返回。页所属的文件号在 SQLIOESTIMATE 字段中返回，此值与在 DB_BACKUP_OPEN_FILE 调用中使用的文件号之一相匹配。返回的数据大小存储在 DT_LONGBINARY 变量的 *stored_len* 字段中，并且总是数据库页面大小的倍数。当此调用所返回的数据内包含某个给定文件的连续页块时，单独的数据块不一定会按顺序返回，某一数据库文件的所有页也不一定会在另一数据库文件的页之前全部返回。调用程序应该准备接收不按顺序的另一单个文件的一部分或在任何给定调用中打开的任何数据库文件的一部分。

应用程序应该重复调用此操作，直到读取的数据大小为 0 或者 sqlda->sqlvar[0].sqlind 的值大于 0。如果备份启动时重命名/重新启动了事务日志，则 SQLERROR 可能设置为 SQLE_BACKUP_CANNOT_RENAME_LOG_YET。在这种情况下，不要使用返回的页，而是要重新发出请求，直到收到 SQLE_NOERROR，然后写入数据。可能会在多个页上多次返回 SQLE_BACKUP_CANNOT_RENAME_LOG_YET 错误。您应该在重试循环中增加延迟，以便不会因请求过多而降低数据库服务器的速度。继续读取页，直到满足上述两个条件之一。

dbbackup 实用程序使用下面的算法。这不是 C 代码，不包括错误检查。

```
sqlda->sqld = 1;
sqlda->sqlvar[0].sqltype = DT_LONGBINARY

/* Allocate LONGBINARY value for page buffer. It MUST have */
/* enough room to hold the requested number (128) of database pages
*/
sqlda->sqlvar[0].sqldata = allocated buffer

/* Open the server files needing backup */
for file_num = 0 to DB_BACKUP_MAX_FILE
  db_backup( ... DB_BACKUP_OPEN_FILE, file_num ... )
  if SQLCODE == SQLE_NO_ERROR
    /* The file exists */
    num_pages = SQLCOUNT
    file_time = SQLE_IO_ESTIMATE
    open backup file with name from sqlca.sqlerrmc
end for

/* read pages from the server, write them locally */
while TRUE
  /* file_no and page_no are ignored */
  db_backup( &sqlca, DB_BACKUP_PARALLEL_READ, 0, 0, &sqlda );

  if SQLCODE != SQLE_NO_ERROR
     break;

  if buffer->stored_len == 0 || sqlda->sqlvar[0].sqlind > 0
     break;

  /* SQLCOUNT contains the starting page number of the block    */
  /* SQLIOESTIMATE contains the file number the pages belong to */
  write block of pages to appropriate backup file
end while

/* close the server backup files */
for file_num = 0 to DB_BACKUP_MAX_FILE
    /* close backup file */
    db_backup( ... DB_BACKUP_CLOSE_FILE, file_num ... )
end for

/* shut down the backup */
db_backup( ... DB_BACKUP_END ... )

/* cleanup */
free page buffer
```

db_cancel_request 函数

取消当前活动的数据库服务器请求。此函数会进行检查，以确保在发送取消请求之前数据库服务器请求是活动的。

语法

```
int db_cancel_request( SQLCA * sqlca );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。

返回值
发送取消请求时返回 1；无请求发送时返回 0。

注释
一个非零返回值不表示请求被取消。在几个临界时刻，取消请求和来自数据库或服务器的响应会发生交错。在这些情况下，即使函数仍然返回 TRUE，取消请求也没有效果。

可以异步调用 db_cancel_request 函数。数据库接口库中只有此函数和 db_is_working 可以使用可能正被另一请求使用的 SQLCA 进行异步调用。

如果您取消正在执行游标操作的请求，则游标的位置是不确定的。在取消之后，您必须按游标的绝对位置定位该游标或关闭它。

db_change_char_charset 函数

更改用于此连接的应用程序的 CHAR 字符集。

语法

```
unsigned int db_change_char_charset(
SQLCA * sqlca,
char * charset );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **charset** – 表示字符集的字符串。

返回值
如果更改成功则返回 1；否则返回 0。

注释
使用 DT_FIXCHAR、DT_VARCHAR、DT_LONGVARCHAR 和 DT_STRING 类型发送和读取的数据采用 CHAR 字符集。

db_change_nchar_charset 函数

更改用于此连接的应用程序的 NCHAR 字符集。

语法

```
unsigned int db_change_nchar_charset(
SQLCA * sqlca,
char * charset );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **charset** – 表示字符集的字符串。

返回值
如果更改成功则返回 1；否则返回 0。

注释
使用 DT_NFIXCHAR、DT_NVARCHAR、DT_LONGNVARCHAR 和 DT_NSTRING 主机变量类型发送和读取的数据采用 NCHAR 字符集。

如果没有调用 db_change_nchar_charset 函数，将使用 CHAR 字符集发送和读取所有数据。通常，要发送和读取 Unicode 数据的应用程序应该将 NCHAR 字符集设置为 UTF-8。

如果调用此函数，charset 参数通常为 "UTF-8"。NCHAR 字符集不能设置为 UTF-16。

在嵌入式 SQL 中，缺省情况下将 NCHAR、NVARCHAR 和 LONG NVARCHAR 分别描述为 DT_FIXCHAR、DT_VARCHAR 和 DT_LONGVARCHAR。如果调用了 db_change_nchar_charset 函数，则这些类型被分别描述为 DT_NFIXCHAR、DT_NVARCHAR 和 DT_LONGNVARCHAR。

db_find_engine 函数

返回有关本地数据库服务器的状态信息。

语法

```
unsigned short db_find_engine(
SQLCA * sqlca,
char * name );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **name** – NULL 或包含服务器名称的字符串。

返回值
表示服务器状态的无符号的短整型值；如果共享内存中找不到服务器，则为 0。

注释
返回无符号的短整型值，指示有关名为 *name* 的本地数据库服务器的状态信息。如果共享内存中找不到具有指定名称的服务器，则返回值为 0。非零值表示本地服务器当前正在运行。

如果为 *name* 指定了空指针，则返回有关缺省数据库服务器的信息。

返回值中的每个位都指示某一信息。代表不同信息段的位的常量在 `sqldef.h` 头文件中定义。其含义如下。

- **DB_ENGINE** – 始终设置该标志。
- **DB_CLIENT** – 始终设置该标志。
- **DB_CAN_MULTI_DB_NAME** – 该标志已过时。
- **DB_DATABASE_SPECIFIED** – 始终设置该标志。
- **DB_ACTIVE_CONNECTION** – 始终设置该标志。
- **DB_CONNECTION_DIRTY** – 该标志已过时。
- **DB_CAN_MULTI_CONNECT** – 该标志已过时。
- **DB_NO_DATABASES** – 如果服务器未启动数据库，则设置该标志。

db_fini 函数

此函数释放由数据库接口或 DLL 使用的资源。

语法

```
int db_fini( SQLCA * sqlca );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。

返回值
成功时返回非零值；否则返回 0。

注释
在调用 db_fini 之后不能进行任何其它库调用或执行任何嵌入式 SQL 语句。如果在处理过程中出现错误，则在 SQLCA 中设置错误代码且函数返回 0。如果没有错误，则返回非零值。

需要为每个使用的 SQLCA 调用一次 db_fini。

不应从 Windows 动态链接库中的 DllMain 函数直接或间接地调用 db_fini 函数。DllMain 入口点函数只用于执行简单初始化和终止任务。调用 db_fini 可能创建死锁和循环相关性。

db_get_property 函数

用于获得有关数据库接口或连接到的服务器的信息。

语法

```
unsigned int db_get_property(
SQLCA * sqlca,
a_db_property property,
char * value_buffer,
int value_buffer_size );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **a_db_property** – 请求的属性，DB_PROP_CLIENT_CHARSET、DB_PROP_SERVER_ADDRESS 或 DB_PROP_DBLIB_VERSION。
- **value_buffer** – 此参数填充的是以空值终止的字符串形式的属性值。
- **value_buffer_size** – 字符串 value_buffer 的最大长度，包括终止空字符的位置。

返回值
如果成功则返回 1；否则返回 0。

注释
支持以下属性：

- **DB_PROP_CLIENT_CHARSET** – 此属性值可获取客户端字符集（如 "windows-1252"）。
- **DB_PROP_SERVER_ADDRESS** – 此属性值获取当前连接的服务器网络地址，作为可打印字符串。共享内存协议始终会为地址返回空字符串。TCP/IP 协议会返回非空字符串地址。
- **DB_PROP_DBLIB_VERSION** – 此属性值可获取数据库接口库的版本（例如，"16.0.0.1297"）。

db_init 函数

此函数初始化数据库接口库。

语法

```
int db_init( SQLCA * sqlca );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。

返回值
成功返回非零值；否则返回 0。

注释
此函数必须在调用任何其它库之前或执行任何嵌入式 SQL 语句之前进行调用。在进行此调用时将分配和初始化接口库对您的程序所要求的资源。

在程序结尾处使用 db_fini 来释放资源。如果在处理过程中出现错误，则在 SQLCA 中返回这些错误且返回 0。如果没有错误，则返回非零值，您便可开始使用嵌入式 SQL 语句和函数。

通常，只应调用一次此函数（传递 `sqlca.h` 头文件中定义的全局 sqlca 变量的地址）。如果您要使用嵌入式 SQL 来编写具有多个线程的 DLL 或应用程序，则每使用一个 SQLCA，就要调用一次 db_init。

db_is_working 函数

如果您的应用程序有一个使用给定 *sqlca* 的数据库请求正在进行中，则返回 1；若没有使用给定 *sqlca* 的请求正在进行中，则返回 0。

语法

```
unsigned short db_is_working( SQLCA * sqlca );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。

返回值
如果您的应用程序有一个使用给定 *sqlca* 的数据库请求正在进行中，则返回 1；若没有使用给定 *sqlca* 的请求正在进行中，则返回 0。

注释
可以异步调用此函数。数据库接口库中只有此函数和 db_cancel_request 可以使用 SQLCA（可能正被另一请求使用）进行异步调用。

db_locate_servers 函数

提供对 dblocate 实用程序所显示的信息的编程式访问，列出本地网络上正在监听 TCP/IP 的所有 SAP Sybase IQ 数据库服务器。

语法

```
unsigned int db_locate_servers(
SQLCA * sqlca,
SQL_CALLBACK_PARM callback_address,
void * callback_user_data );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。

- **callback_address** – 回调函数的地址。
- **callback_user_data** – 用于存储数据的用户定义区域的地址。

返回值
如果成功则返回 1；否则返回 0。

注释
回调函数必须具有以下原型：

```
int (*)( SQLCA * sqlca,
a_server_address * server_addr,
void * callback_user_data );
```

对于找到的每台服务器都要调用回调函数。如果回调函数返回 0，则 db_locate_servers 停止遍历服务器。

传递到回调函数的 sqlca 和 callback_user_data 是那些传递到 db_locate_servers 中的函数。第二个参数是指向 a_server_address 结构的指针。a_server_address 在 `sqlca.h` 中定义，其定义如下：

```
typedef struct a_server_address {
    a_sql_uint32 port_type;
    a_sql_uint32 port_num;
    char         *name;
    char         *address;
} a_server_address;
```

- **port_type** – 它在此时始终为 PORT_TYPE_TCP（在 `sqlca.h` 中定义为 6）。
- **port_num** – 它是此服务器正在监听的 TCP 端口号。
- **name** – 指向包含服务器名称的缓冲区。
- **address** – 指向包含服务器 IP 地址的缓冲区。

db_locate_servers_ex 函数

提供对 dblocate 实用程序所显示的信息的编程式访问，列出本地网络上正在监听 TCP/IP 的所有 SAP Sybase IQ 数据库服务器，但提供用于选择传递给回调函数的地址的掩码参数。

语法

```
unsigned int db_locate_servers_ex(
SQLCA * sqlca,
SQL_CALLBACK_PARM callback_address,
void * callback_user_data,
unsigned int bitmask);
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **callback_address** – 回调函数的地址。

- **callback_user_data** – 用于存储数据的用户定义区域的地址。
- **bitmask** – 由 DB_LOOKUP_FLAG_NUMERIC、DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT 或 DB_LOOKUP_FLAG_DATABASES 中的任意一个组成的掩码。

返回值
如果成功则返回 1；否则返回 0。

注释
回调函数必须具有以下原型：

```
int (*)( SQLCA * sqlca,
a_server_address * server_addr,
void * callback_user_data );
```

对于找到的每台服务器都要调用回调函数。如果回调函数返回 0，则 db_locate_servers_ex 停止遍历服务器。

传递到回调函数的 sqlca 和 callback_user_data 是那些传递到 db_locate_servers 中的函数。第二个参数是指向 a_server_address 结构的指针。a_server_address 在 `sqlca.h` 中定义，其定义如下：

```
typedef struct a_server_address {
    a_sql_uint32   port_type;
    a_sql_uint32   port_num;
    char           *name;
    char           *address;
    char           *dbname;
} a_server_address;
```

- **port_type** – 它在此时始终为 PORT_TYPE_TCP（在 `sqlca.h` 中定义为 6）。
- **port_num** – 它是此服务器正在监听的 TCP 端口号。
- **name** – 指向包含服务器名称的缓冲区。
- **address** – 指向包含服务器 IP 地址的缓冲区。
- **dbname** – 指向包含数据库名称的缓冲区。

支持三种位掩码标志：

DB_LOOKUP_FLAG_NUMERIC
DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT
DB_LOOKUP_FLAG_DATABASES

这些标志在 `sqlca.h` 中定义，可用 OR 连起来。

DB_LOOKUP_FLAG_NUMERIC 确保传递给回调函数的地址是 IP 地址，而不是主机名。

DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT 指定传递给回调函数的地址包括 a_server_address 结构中的 TCP/IP 端口号。

DB_LOOKUP_FLAG_DATABASES 指定为找到的每个数据库调用一次回调函数，或者，如果数据库服务器不支持发送数据库信息（9.0.2 及更早版本的数据库服务器），指定为找到的每个数据库服务器调用一次回调函数。

db_register_a_callback 函数

此函数注册回调函数。

语法

```
void db_register_a_callback(
SQLCA * sqlca,
a_db_callback_index index,
( SQL_CALLBACK_PARM ) callback );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **index** – 标识下述回调类型的索引值。
- **回调** – 用户定义的回调函数的地址。

注释

如果您不注册 DB_CALLBACK_WAIT 回调，则缺省操作是不执行任何操作。您的应用程序块，等待数据库响应。必须为 MESSAGE TO CLIENT 语句注册一个回调。

要删除回调，请传递一个空指针作为 *callback* 函数。

对于 *index* 参数，允许使用以下值：

- **DB_CALLBACK_DEBUG_MESSAGE** – 对每条调试信息都要调用一次提供的函数，而且会有一个包含该调试信息文本的以空值终止的字符串传递给该函数。调试消息是记录到 LogFile 文件的消息。为了将调试消息传递给此回调函数，必须使用 LogFile 连接参数。该字符串通常在紧邻终止的空字符之前有一个换行符 (\n)。回调函数的原型如下：

  ```
  void SQL_CALLBACK debug_message_callback(
  SQLCA * sqlca,
  char * message_string );
  ```

- **DB_CALLBACK_START** – 原型如下：

  ```
  void SQL_CALLBACK start_callback( SQLCA * sqlca );
  ```

  此函数在数据库请求发送到服务器之前进行调用。DB_CALLBACK_START 只在 Windows 上使用。

- **DB_CALLBACK_FINISH** – 原型如下：

  ```
  void SQL_CALLBACK finish_callback( SQLCA * sqlca );
  ```

  此函数在 DBLIB 接口 DLL 收到对数据库请求的响应之后进行调用。DB_CALLBACK_FINISH 仅在 Windows 操作系统上使用。

- **DB_CALLBACK_CONN_DROPPED** – 原型如下：

```
void SQL_CALLBACK conn_dropped_callback (
SQLCA * sqlca,
char * conn_name );
```

此函数在数据库服务器要通过 DROP CONNECTION 语句删除连接（因活动超时或因该数据库服务器正在关闭）时进行调用。会为此函数传递连接名称 *conn_name*，以便使您能够区分连接。如果没有命名连接，则它的值为 NULL。

- **DB_CALLBACK_WAIT** – 原型如下：

```
void SQL_CALLBACK wait_callback( SQLCA * sqlca );
```

在数据库服务器或客户端库忙于处理您的数据库请求的同时，接口库反复调用此函数。

您可以按如下所示注册此回调函数：

```
db_register_a_callback( &sqlca,
   DB_CALLBACK_WAIT,
   (SQL_CALLBACK_PARM)&db_wait_request );
```

- **DB_CALLBACK_MESSAGE** – 使用此函数后，应用程序可以对处理请求期间从服务器接收的消息进行处理。可使用 SQL MESSAGE 语句将消息从数据库服务器发送到客户端应用程序。长时间运行的数据库服务器语句也可以生成消息。

回调原型如下：

```
void SQL_CALLBACK message_callback(
SQLCA * sqlca,
unsigned char msg_type,
an_sql_code code,
unsigned short length,
char *  msg
);
```

msg_type 参数说明消息的重要程度。您可能希望不同的消息类型以不同的方式来处理。*msg_type* 的以下可能值在 `sqldef.h` 中定义。

- **MESSAGE_TYPE_INFO** – 消息类型为 INFO。
- **MESSAGE_TYPE_WARNING** – 消息类型为 WARNING。
- **MESSAGE_TYPE_ACTION** – 消息类型为 ACTION。
- **MESSAGE_TYPE_STATUS** – 消息类型为 STATUS。
- **MESSAGE_TYPE_PROGRESS** – 消息类型为 PROGRESS。此类消息是由长时间运行的数据库服务器语句生成的，如 BACKUP DATABASE 和 LOAD TABLE。

code 字段可提供与消息关联的 SQLCODE，否则该值为 0。*length* 字段指定消息的长度。消息不以空值终止。SAP Sybase IQ DBLIB 和 ODBC 客户端可以使用 DB_CALLBACK_MESSAGE 参数来接收进度消息。

例如，Interactive SQL 回调在“消息”选项卡中显示 STATUS 和 INFO 消息，而在窗口中显示类型为 ACTION 和 WARNING 的消息。如果应用程序不注册此回调，

会有一个缺省回调，它导致将所有消息写入服务器日志文件（如果正在调试并指定了日志文件）。另外，MESSAGE_TYPE_WARNING 和 MESSAGE_TYPE_ACTION 类型的消息会以与操作系统相关的方式更为突出地显示。

如果某消息回调未被应用程序注册，则在指定了 LogFile 连接参数后，发送至客户端的消息将被保存到日志文件中。此外，发送至客户端的 ACTION 或 STATUS 消息在 Windows 操作系统下会出现窗口中，而在 Unix 操作系统下则被记录到 stderr。

- **DB_CALLBACK_VALIDATE_FILE_TRANSFER** – 用于注册文件传输校验回调函数。在允许进行任何传输前，客户端库会调用校验回调函数（如果存在）。如果在执行间接语句（从存储过程内部）期间，请求进行客户端数据传输，则除非客户端应用程序注册了校验回调函数，否则客户端库将不允许进行传输。下面更详尽地介绍了进行校验调用的条件。

  回调原型如下：

  ```
  int SQL_CALLBACK file_transfer_callback(
  SQLCA * sqlca,
  char * file_name,
  int is_write
  );
  ```

  file_name 参数是要读取或写入的文件的名称。如果请求读取（从客户端传输到服务器），则 *is_write* 参数为 0；如果请求写入，则该参数为非零值。如果不允许进行文件传输，则回调函数应返回 0，否则返回非零值。

  为确保数据安全，服务器会跟踪请求文件传输的语句的源。服务器会确定语句是否是从客户端应用程序直接接收的。从客户端启动数据传输时，服务器会将语句源的相关信息发送到客户端软件。对于嵌入式 SQL 客户端库而言，仅当是由于执行客户端应用程序直接发送的语句而请求数据传输时，它才会允许无条件传输数据。否则，应用程序必须注册上文所述的校验回调函数，如果未注册该函数，则传输会被拒绝，而且语句失败并出现一个错误。如果客户端语句调用的某个存储过程在数据库中已经存在，则对该存储过程本身的执行不被视为是客户端启动的语句所完成的。但是，如果客户端应用程序显式地创建一个临时存储过程，则服务器会将对该存储过程的执行视为是由客户端启动的。同样，如果客户端应用程序执行一个批处理语句，则该批处理语句的执行被视为是直接由客户端应用程序完成的。

db_start_database 函数

如果可能，请在现有服务器上启动数据库。否则，启动新的服务器。

语法

```
unsigned int db_start_database( SQLCA * sqlca, char * parms );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **parms** – 以空值终止的字符串，其中包含以分号分隔的参数设置列表，每个参数设置的形式均为“关键字=*值*”。例如：

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

返回值
成功返回非零值；否则返回 0。

注释
在现有服务器上启动数据库（如果可能）。否则，启动新的服务器。

如果数据库已运行或已成功启动，则返回值为 true（非零）并且将 SQLCODE 设置为 0。错误消息在 SQLCA 中返回。

如果在参数中提供了用户 ID 和口令，则它们将被忽略。

启动和停止数据库所需的特权在服务器命令行上使用 -gd 选项进行设置。

db_start_engine 函数

如果数据库服务器尚未运行，则将其启动。

语法

```
unsigned int db_start_engine( SQLCA * sqlca, char * parms );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **parms** – 以空值终止的字符串，其中包含以分号分隔的参数设置列表，每个参数设置的形式均为“关键字=*值*”。例如：

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

返回值
成功返回非零值；否则返回 0。

注释
如果数据库已运行或已成功启动，则返回值为 TRUE（非零）并且将 SQLCODE 设置为 0。错误消息在 SQLCA 中返回。

下面对 db_start_engine 的调用将启动数据库服务器、加载指定数据库，以及将服务器命名为 demo。

```
db_start_engine( &sqlca, "DBF=demo.db;START=iqsrv16" );
```

除非使用 ForceStart (FORCE) 连接参数并将其设置为 YES，否则 db_start_engine 函数将尝试在启动服务器前连接到服务器，来避免尝试启动已在运行的服务器。

当 ForceStart 连接被设置为 YES 时，在尝试启动服务器之前不尝试连接到服务器。这样，下面的一对命令就能够按预期方式工作：

1. 启动名为 server_1 的数据库服务器：
```
iqsrv16 -n server_1 demo.db
```
2. 强制启动一台新服务器并连接到它：
```
db_start_engine( &sqlda,
  "START=iqsrv16 -n server_2 mydb.db;ForceStart=YES" )
```

如果未使用 ForceStart (FORCE)，而且未使用 ServerName (Server) 参数，则第二个命令会尝试连接到 server_1。db_start_engine 函数不从 StartLine (START) 参数的 -n 选项获取服务器名。

db_stop_database 函数

在由 ServerName (Server) 标识的服务器上停止由 DatabaseName (DBN) 标识的数据库。如果未指定 ServerName，则使用缺省服务器。

语法

```
unsigned int db_stop_database( SQLCA * sqlca, char * parms );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **parms** – 以 NULL 终止的字符串，其中包含以分号分隔的参数设置列表，每个参数设置的形式均为“关键字=*值*”。例如：
```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

返回值
成功返回非零值；否则返回 0。

注释
缺省情况下，此函数不停止存在现有连接的数据库。如果 Unconditional (UNC) 设置为 yes，则无论当前是否存在连接都会停止数据库。

返回值 TRUE 指示没有错误。

启动和停止数据库所需的特权在服务器命令行上使用 -gd 选项进行设置。

db_stop_engine 函数

停止数据库服务器的执行。

语法

```
unsigned int db_stop_engine( SQLCA * sqlca, char * parms );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **parms** – 以 NULL 终止的字符串，其中包含以分号分隔的参数设置列表，每个参数设置的形式均为“关键字=*值*”。例如：

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

返回值
成功返回非零值；否则返回 0。

注释
此函数执行的步骤有：

- 查找名称与 ServerName (Server) 参数匹配的本地数据库服务器。如果未指定 ServerName，则查找缺省的本地数据库服务器。
- 如果找不到匹配的服务器，则此函数成功返回。
- 向服务器发送一个请求，让服务器执行检查点操作并关闭所有数据库。
- 卸载数据库服务器。

缺省情况下，此函数不停止有现有连接的数据库。如果指定了 Unconditional=yes 连接参数，则不管是否存在现有连接都会停止数据库服务器。

C 程序可以使用此函数，而不用正在生成的 dbstop。返回值 TRUE 指示没有错误。

能否使用 db_stop_engine 取决于使用 -gk 服务器选项设置的特权。

db_string_connect 函数

提供了嵌入式 SQL CONNECT 语句所具有的功能以外的其它功能。

语法

```
unsigned int db_string_connect( SQLCA * sqlca, char * parms );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **parms** – 以 NULL 终止的字符串，其中包含以分号分隔的参数设置列表，每个参数设置的形式均为“关键字=*值*”。例如：

```
"UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
```

返回值
成功返回非零值；否则返回 0。

注释
如果成功建立连接，则返回值为 TRUE（非零），否则返回值为 FALSE（零）。有关启动服务器、启动数据库或进行连接的错误消息均在 SQLCA 中返回。

db_string_disconnect 函数

此函数断开由 ConnectionName 参数标识的连接。忽略所有其它参数。

语法

```
unsigned int db_string_disconnect(
    SQLCA * sqlca,
    char * parms );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **parms** – 以 NULL 终止的字符串，其中包含以分号分隔的参数设置列表，每个参数设置的形式均为“关键字=*值*”。例如：

  ```
  "UID=DBA;PWD=sql;DBF=c:\\db\\mydatabase.db"
  ```

返回值
成功返回非零值；否则返回 0。

注释
如果在字符串中未指定 ConnectionName 参数，则断开未命名的连接。它等效于嵌入式 SQL DISCONNECT 语句。如果连接成功结束，则返回值为 TRUE。错误消息在 SQLCA 中返回。

如果数据库是使用 AutoStop=yes 连接参数启动的且没有其它到该数据库的连接，则此函数关闭该数据库。如果服务器是使用 AutoStop=yes 参数启动的且没有其它正在运行的数据库，它也会停止该服务器。

db_string_ping_server 函数

此函数可用于确定是否能找到服务器，也可确定是否可成功连接到数据库。

语法

```
unsigned int db_string_ping_server(
SQLCA * sqlca,
char * connect_string,
unsigned int connect_to_db );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **connect_string** – *connect_string* 是一个常规连接字符串，它可能包含服务器和数据库信息，也可能不包含。
- **connect_to_db** – 如果 *connect_to_db* 为非零值 (TRUE)，此函数会尝试连接到服务器上的数据库。只有在连接字符串足以连接到指定服务器上的指定数据库时，才返回 TRUE。

如果 *connect_to_db* 为零，此函数只尝试定位服务器。只有在连接字符串足以定位服务器时，才返回 TRUE。它不尝试连接到数据库。

返回值
如果已成功定位服务器或数据库则返回 TRUE （非零）；否则返回 FALSE （零）。在 SQLCA 中返回关于定位服务器或数据库的错误消息。

db_time_change 函数

此函数允许客户端通知服务器客户端上的时间已发生更改。

语法

```
unsigned int db_time_change(
SQLCA * sqlca);
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。

返回值
如果成功则返回 TRUE，否则返回 FALSE。

注释
此函数将重新计算时区调整并将其发送给服务器。在 Windows 平台上，建议应用程序在接收到 WM_TIMECHANGE 消息时调用此函数。这样可确保 UTC 时间戳与时间更改、时区更改或夏令时更改保持一致。

fill_s_sqlda 函数

与 fill_sqlda 相同，只不过它将 *sqlda* 中的所有数据类型更改为类型 DT_STRING。

语法

```
struct sqlda * fill_s_sqlda(
struct sqlda * sqlda,
unsigned int maxlen );
```

参数

- **sqlda** – 指向 SQLDA 结构的指针。
- **maxlen** – 要为字符串分配的最大字节数。

返回值
如果成功则返回 *sqlda*；如果没有足够的可用内存则返回 NULL。

注释
将分配足够的空间，以保存最初由 SQLDA 指定的类型的字符串表示，最大为 *maxlen* 字节。会相应修改 SQLDA 中的长度字段 (sqllen)。

SQLDA 应使用 free_filled_sqlda 函数释放。

fill_sqlda 函数

为 *sqlda* 的每个描述符中说明的每个变量分配空间，并将此内存的地址指派给对应描述符的 sqldata 字段。

语法

```
struct sqlda * fill_sqlda( struct sqlda * sqlda );
```

参数

- **sqlda** – 指向 SQLDA 结构的指针。

返回值
如果成功则返回 *sqlda*；如果没有足够的可用内存则返回 NULL。

注释
为描述符中指明的数据库类型和长度分配足够的空间。

SQLDA 应使用 free_filled_sqlda 函数释放。

fill_sqlda_ex 函数

为 *sqlda* 的每个描述符中说明的每个变量分配空间，并将此内存的地址指派给对应描述符的 sqldata 字段。

语法

```
struct sqlda * fill_sqlda_ex( struct sqlda * sqlda , unsigned int
flags);
```

参数

- **sqlda** – 指向 SQLDA 结构的指针。
- **flags** – 0 或 FILL_SQLDA_FLAG_RETURN_DT_LONG

返回值
如果成功则返回 *sqlda*；如果没有足够的可用内存则返回 NULL。

注释
为描述符中指明的数据库类型和长度分配足够的空间。

SQLDA 应使用 free_filled_sqlda 函数释放。

支持一个标志位：FILL_SQLDA_FLAG_RETURN_DT_LONG。此标志在 sqlca.h 中定义。

FILL_SQLDA_FLAG_RETURN_DT_LONG 在填充的描述符中保留 DT_LONGVARCHAR、DT_LONGNVARCHAR 和 DT_LONGBINARY 类型。如果未

指定此标志位，fill_sqlda_ex 将把 DT_LONGVARCHAR、DT_LONGNVARCHAR 和 DT_LONGBINARY 类型分别转换为 DT_VARCHAR、DT_NVARCHAR 和 DT_BINARY。使用 DT_LONG*xyz* 类型可读取 32767 字节，而使用 DT_VARCHAR、DT_NVARCHAR 和 DT_BINARY 只能读取 32765 字节。

fill_sqlda(sqlda) 等价于 fill_sqlda_ex(sqlda, 0)。

free_filled_sqlda 函数

释放分配给每个 sqldata 指针的内存和分配给 SQLDA 自身的空间。不释放任何空指针。

语法

```
void free_filled_sqlda( struct sqlda * sqlda );
```

参数

- **sqlda** – 指向 SQLDA 结构的指针。

注释

只有在 fill_sqlda、fill_sqlda_ex 或 fill_s_sqlda 用来分配 SQLDA 的 sqldata 字段时才应该调用此函数。

调用此函数会导致自动调用 free_sqlda，因此由 alloc_sqlda 分配的任何描述符都被释放。

free_sqlda 函数

释放分配给此 *sqlda* 的空间并释放指示符变量空间，这些空间是在 fill_sqlda 中分配的。

语法

```
void free_sqlda( struct sqlda * sqlda );
```

参数

- **sqlda** – 指向 SQLDA 结构的指针。

注释

不会释放每个 sqldata 指针引用的内存。

free_sqlda_noind 函数

释放分配给此 *sqlda* 的空间。不会释放每个 sqldata 指针引用的内存。将忽略指示符变量指针。

语法

```
void free_sqlda_noind( struct sqlda * sqlda );
```

参数

- **sqlda** – 指向 SQLDA 结构的指针。

sql_needs_quotes 函数

此函数向数据库服务器发出请求以确定是否需要引号。相关信息存储在 sqlcode 字段中。

语法

```
unsigned int sql_needs_quotes( SQLCA *sqlca, char * str );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **str** – 作为 SQL 标识符候选对象的字符串。

返回值

返回 TRUE 或 FALSE，用于指示当字符串用作 SQL 标识符时是否需要用双引号引起来。

注释

返回值/代码的组合有三种情况：

- **return = FALSE, sqlcode = 0** – 该字符串不需要引号。
- **return = TRUE** – sqlcode 始终是 SQLE_WARNING，该字符串需要引号。
- **return = FALSE** – 如果 sqlcode 不是 0 或 SQLE_WARNING，则此测试不能确定是否需要引号。

sqlda_storage 函数

返回无符号的 32 位整数值表示存储 varno 变量的任意值时所需的存储量。

语法

```
a_sql_uint32 sqlda_storage( struct sqlda * sqlda, int varno );
```

参数

- **sqlda** – 指向 SQLDA 结构的指针。
- **varno** – sqlvar 主机变量的索引。

返回值

返回无符号的 32 位整数值表示存储变量的任意值时所需的存储量。

sqlda_string_length 函数

返回无符号的 32 位整数值，该值表示保存变量 sqlda->sqlvar[varno]（不管其类型是什么）所需的 C 字符串（DT_STRING 类型）的长度。

语法

```
a_sql_uint32 sqlda_string_length( struct sqlda * sqlda, int
varno );
```

参数

- **sqlda** – 指向 SQLDA 结构的指针。
- **varno** – sqlvar 主机变量的索引。

返回值
返回无符号的 32 位整数值，该值表示保存变量 sqlda->sqlvar[varno]（不管其类型是什么）所需的 C 字符串（DT_STRING 类型）的长度。

sqlerror_message 函数

返回到包含错误消息的字符串的指针。错误消息包含 SQLCA. 中的错误代码的文本。如果没有指出错误，则返回空指针。将错误消息置于提供的缓冲区中，如有必要则将其长度截至 *max*。

语法

```
char * sqlerror_message( SQLCA * sqlca, char * buffer, int max );
```

参数

- **sqlca** – 指向 SQLCA 结构的指针。
- **buffer** – 用于放置消息的缓冲区（最多 *max* 个字符）。
- **max** – 缓冲区的最大长度。

返回值
返回一个指向包含错误消息的字符串的指针；如果无错误指示，则返回 NULL。

嵌入式 SQL 语句汇总

所有嵌入式 SQL 语句都必须以 EXEC SQL 开头且以分号 (;) 结尾。

有两组嵌入式 SQL 语句。标准 SQL 语句的用法是：只需将其置于 C 程序中，并在其前后分别加上 EXEC SQL 和分号 (;) 即可。CONNECT、DELETE、SELECT、SET 和 UPDATE 有一些附加格式只能在嵌入式 SQL 中使用。具有附加格式的语句则属于特定于嵌入式 SQL 语句的第二个类别。

有几个 SQL 语句是特定于嵌入式 SQL 的并只能在 C 程序中使用。

可以从嵌入式 SQL 应用程序使用标准的数据操作语句和数据定义语句。另外，以下语句也是专用于嵌入式 SQL 编程的语句：

- **ALLOCATE DESCRIPTOR 语句 [ESQL]** – 为描述符分配内存。
- **CLOSE 语句 [ESQL] [SP]** – 关闭游标。
- **CONNECT 语句 [ESQL] [Interactive SQL]** – 连接到数据库。
- **DEALLOCATE DESCRIPTOR 语句 [ESQL]** – 回收描述符占用的内存。
- **声明部分 [ESQL]** – 为数据库通信声明主机变量。
- **DECLARE CURSOR 语句 [ESQL] [SP]** – 声明游标。
- **DELETE 语句（定位） [ESQL] [SP]** – 删除游标中当前位置的行。
- **DESCRIBE 语句 [ESQL]** – 描述特定 SQL 语句的主机变量。
- **DISCONNECT 语句 [ESQL] [Interactive SQL]** – 断开与数据库服务器的连接。
- **DROP STATEMENT 语句 [ESQL]** – 释放预准备语句所使用的资源。
- **EXECUTE 语句 [ESQL]** – 执行特定的 SQL 语句。
- **EXPLAIN 语句 [ESQL]** – 解释特定游标的优化策略。
- **FETCH 语句 [ESQL] [SP]** – 从游标读取行。
- **GET DATA 语句 [ESQL]** – 从游标读取 Long 型值。
- **GET DESCRIPTOR 语句 [ESQL]** – 检索有关 SQLDA 中变量的信息。
- **GET OPTION 语句 [ESQL]** – 获得特定数据库选项的设置。
- **INCLUDE 语句 [ESQL]** – 包括要进行 SQL 预处理的文件。
- **OPEN 语句 [ESQL] [SP]** – 打开游标。
- **PREPARE 语句 [ESQL]** – 准备特定的 SQL 语句。
- **PUT 语句 [ESQL]** – 向游标中插入行。
- **SET CONNECTION 语句 [Interactive SQL] [ESQL]** – 更改活动连接。
- **SET DESCRIPTOR 语句 [ESQL]** – 描述 SQLDA 中的变量并将数据置于 SQLDA 中。
- **SET SQLCA 语句 [ESQL]** – 使用缺省全局 SQLCA 以外的 SQLCA。
- **UPDATE（定位）语句 [ESQL] [SP]** – 更新游标当前所在的行。
- **WHENEVER 语句 [ESQL]** – 指定 SQL 语句中出现错误时要执行的操作。

C/C++ 的 SAP Sybase IQ 数据库 API

SAP Sybase IQ C 应用程序编程接口 (API) 是 C/C++ 语言的数据访问 API。C API 规范定义了一组函数、变量和约定，这些函数、变量和约定提供了一致的数据库接口,该接口独立于实际所用数据库。使用 SAP Sybase IQ C API，您的 C/C++ 应用程序可以直接访问 SAP Sybase IQ 数据库服务器。

Perl DBI 支持

DBD::SQLAnywhere 是用于 DBI 的 SAP Sybase IQ 数据库驱动程序，DBI 是用于 Perl 语言的数据访问 API。DBI API 规范定义了一组函数、变量和约定，这些函数、变量和约定提供了一致的数据库接口,该接口独立于实际所用数据库。使用 DBI 和 DBD::SQLAnywhere，您的 Perl 脚本可以直接访问 SAP Sybase IQ 数据库服务器。

DBD::SQLAnywhere

DBD::SQLAnywhere 是由 Tim Bunce 编写的用于 Perl 的数据库独立接口 (DBI) 模块的驱动程序。安装 DBI 模块和 DBD::SQLAnywhere 后，便可使用 Perl 来访问和更改 SAP Sybase IQ 数据库中的信息。

使用 Perl ithread 模式时，DBD::SQLAnywhere 驱动程序是线程安全的。

要求
DBD::SQLAnywhere 接口需要以下组件。

- Perl 5.6.0 或更高版本。在 Windows 上，要求使用 ActivePerl 5.6.0 内部版本 616 或更高版本。
- DBI 1.34 或更高版本。
- C 编译器。Windows 仅支持 Microsoft Visual C++ 编译器。

在 Windows 上安装 DBD::SQLAnywhere

在支持的 Windows 平台上安装 DBD::SQLAnywhere 接口后，才能使用 Perl 访问 SAP Sybase IQ 数据库。

前提条件

- 建立 iqdemo 数据库。
- 安装 ActivePerl 5.6.0 或更高版本。您可以使用 ActivePerl 安装程序安装 Perl 并配置计算机。无需重新编译 Perl。
- 安装 Microsoft Visual Studio 并配置环境。
  如果没有选择在安装时配置环境，则必须正确设置 PATH、LIB 和 INCLUDE 环境变量才能继续。Microsoft 为此提供了一个批处理文件。对于 32 位版本，Visual Studio 2005 或 2008 安装目录的 `vc\bin` 子目录中提供了名为 `vcvars32.bat` 的批处理文件。对于 64 位版本，请查找此批处理文件的 64 位版本，例如 `vcvarsamd64.bat`。打开一个新的系统命令提示符并运行此批处理文件，然后再继续。

有关配置 64 位 Visual C++ 版本环境的详细信息，请参见 http://msdn.microsoft.com/en-us/library/x4d2c09s.aspx。

过程

1. 在命令提示符处，转到 ActivePerl 安装目录的 `bin` 子目录。

   强烈建议使用该系统命令提示符，因为下面的步骤可能无法从其它 shell 运行。
2. 通过 Perl Module Manager，输入以下命令。

   ```
   ppm query dbi
   ```

   如果 ppm 无法运行，请检查 Perl 是否安装正确。

   该命令应生成两行如下所示的文本。在此情况下，该信息指示 ActivePerl 版本 5.8.1 内部版本 807 正在运行且 DBI 版本 1.38 已安装。

   ```
   Querying target 1 (ActivePerl 5.8.1.807)
     1. DBI [1.38] Database independent interface for Perl
   ```

   对于较新版本的 Perl，则可能会显示如下所示的表。这种情况下，该信息指示已安装了 DBI 1.58 版。

name	version	abstract	area
DBI	1.58	Database independent interface for Perl	perl

   如果没有安装 DBI，则必须安装。为此，请在 ppm 提示符处输入以下命令。

   ```
   ppm install dbi
   ```
3. 在命令提示符处，转到 SAP Sybase IQ 安装目录的 `SDK\Perl` 子目录。
4. 输入以下命令生成并测试 DBD::SQLAnywhere。

   ```
   perl Makefile.PL
   ```

   ```
   nmake
   ```

   如果出于任何原因需要从头开始，则可以运行 `nmake clean` 命令删除所有未完全生成的目标。
5. 要测试 DBD::SQLAnywhere，请将示例数据库文件复制到 `SDK\Perl` 目录，然后进行测试。

   ```
   copy "%ALLUSERSPROFILE%\SybaseIQ\demo\iqdemo.db" .
   ```

   ```
   iqsrv16 demo
   ```

   ```
   nmake test
   ```

   如果测试未运行，请确保路径中包含 SAP Sybase IQ 安装目录的 `bin32` 或 `bin64` 子目录。
6. 在同一提示符下运行以下命令以完成安装。

   ```
   nmake install
   ```

现在即可使用 DBI Perl 模块和 DBD::SQLAnywhere 接口。

在 Unix 上安装 DBD::SQLAnywhere

在支持的 Unix 平台上安装 DBD::SQLAnywhere 接口后，才能使用 Perl 访问 SAP Sybase IQ 数据库。

前提条件

必须安装 ActivePerl 5.6.0 内部版本 616 或更高版本以及 C 编译器。

过程

1. 从 http://www.cpan.org 下载 DBI 模块源。
2. 将该文件的内容抽取到一个新目录中。
3. 在命令提示符处，转到该新目录并运行以下命令以生成 DBI 模块。

   ```
   perl Makefile.PL
   ```

   ```
   make
   ```

   如果出于任何原因需要从头开始，可使用 `make clean` 命令删除所有未完全生成的目标。
4. 使用下面的命令测试 DBI 模块。

   ```
   make test
   ```
5. 在同一提示符下运行以下命令以完成安装。

   ```
   make install
   ```
6. 确保为 SAP Sybase IQ 设置相应的环境。

   根据您使用的 shell，输入相应的命令以从 SAP Sybase IQ 安装目录执行 SAP Sybase IQ 配置脚本：

shell	使用的命令
sh、ksh 或 bash	`.bin/sa_config.sh`
csh 或 tcsh	source `bin/sa_config.csh`
7. 在 shell 提示符处，转到 SAP Sybase IQ 安装目录的 `sdk/perl` 子目录。
8. 在命令提示符处，运行以下命令来生成 DBD::SQLAnywhere。

   ```
   perl Makefile.PL
   ```

   ```
   make
   ```

   如果出于任何原因需要从头开始，可使用 `make clean` 命令删除所有未完全生成的目标。

9. 要测试 DBD::SQLAnywhere，请将示例数据库文件复制到 `sdk/perl` 目录，然后进行测试。

```
cp samples-dir/demo.db .
```

```
iqsrv16 demo
```

```
make test
```

   如果测试未运行，请确保路径中包含 SAP Sybase IQ 安装目录的 `bin32` 或 `bin64` 子目录。

10. 在同一提示符下运行以下命令以完成安装。

```
make install
```

DBI Perl 模块和 DBD::SQLAnywhere 接口已准备就绪，可供使用。

下一步

您可以选择删除 DBI 源树。不再需要该源树。

使用 DBD::SQLAnywhere 的 Perl 脚本

本节将概述如何编写使用 DBD::SQLAnywhere 接口的 Perl 脚本。

DBD::SQLAnywhere 是 DBI 模块的驱动程序。DBI 模块的完整文档可从 http://dbi.perl.org 获得。

DBI 模块

要在 Perl 脚本中使用 DBD::SQLAnywhere 接口，必须先告知 Perl 您打算使用 DBI 模块。为此，请在文件的顶部加入下面的命令行。

```
use DBI;
```

此外，强烈建议您在严格模式下运行 Perl。例如强制要求显式变量定义的语句，可大大减少由于常见失误（如键入错误）而产生的莫名其妙的错误。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
```

必要时 DBI 模块会自动装载 DBD 驱动程序（包括 DBD::SQLAnywhere）。

如何使用 Perl DBI 打开和关闭数据库连接

通常，打开一个到数据库的连接，然后通过该连接运行一系列 SQL 语句来执行所有需要的操作。要打开连接，请使用 connect 方法。返回值是一个到数据库连接的句柄，使用该句柄可以在连接上执行后继操作。

connect 方法的参数如下所示：

1. "DBI:SQLAnywhere:"和其它连接参数以分号分隔。
2. 用户名。除非该字符串为空，否则 ";UID=*value*" 将附加到连接字符串。
3. 口令值。除非该字符串为空，否则 ";PWD=*value*" 将附加到连接字符串。
4. 指向散列缺省值的指针。AutoCommit、RaiseError 和 PrintError 等可按该方法进行设置。

以下代码示例可打开和关闭到 SAP Sybase IQ 示例数据库的连接。必须先启动数据库服务器和示例数据库然后才能运行此脚本。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my %defaults = (
     AutoCommit => 1, # Autocommit enabled.
     PrintError => 0  # Errors not automatically printed.
   );
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
  or die "Cannot connect to $data_src: $DBI::errstr\n";
$dbh->disconnect;
exit(0);
__END__
```

或者，您可以选择将用户名或口令值附加到数据源字符串，而不是将它们作为独立参数提供。如果要这样做，请为相应参数提供空白字符串。例如，使用下面的语句替换打开连接的语句可修改上面的脚本：

```
$data_src .= ";UID=$uid";
$data_src .= ";PWD=$pwd";
my $dbh = DBI->connect($data_src, '', '', \%defaults)
  or die "Cannot connect to $data_src: $DBI::errstr\n";
```

如何使用 Perl DBI 获取结果集

获得了打开的连接的句柄后，您可以访问和修改存储在数据库中的数据。可能最简单的操作是检索某些行并输出它们。

必须先准备好返回行集的 SQL 语句，然后才能执行它们。prepare 方法为该语句返回一个句柄。使用该句柄执行语句，然后检索关于结果集和结果集的行的元信息。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $sel_stmt = "SELECT ID, GivenName, Surname
                FROM Customers
                ORDER BY GivenName, Surname";
```

```
my %defaults = (
     AutoCommit => 0, # Require explicit commit or rollback.
     PrintError => 0
   );
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
  or die "Cannot connect to $data_src: $DBI::errstr\n";
&db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
      my($sel, $dbh) = @_;
      my($row, $sth) = undef;
      $sth = $dbh->prepare($sel);
      $sth->execute;
      print "Fields:     $sth->{NUM_OF_FIELDS}\n";
      print "Params:     $sth->{NUM_OF_PARAMS}\n\n";
      print join("\t\t", @{$sth->{NAME}}), "\n\n";
      while($row = $sth->fetchrow_arrayref) {
         print join("\t\t", @$row), "\n";
      }
      $sth = undef;
}
__END__
```

直到 Perl 语句句柄被破坏，预准备语句才会从数据库服务器上删除。要破坏语句句柄，请重新使用变量或将其设置为 undef。调用 finish 方法不会删除句柄。实际不应调用 finish 方法，除非决定不完成读取结果集的操作。

为检测句柄泄漏，缺省情况下 SAP Sybase IQ 数据库服务器将允许的游标和准备好的语句数量限制为每个连接最多 50 个。如果超过这些限制，资源调控器自动生成错误。如果收到此错误，请检查未被释放的语句句柄。请谨慎使用 prepare_cached，因为语句句柄并没有被释放。

如有必要，可通过设置 max_cursor_count 和 max_statement_count 选项来修改这些限制。

如何使用 Perl DBI 处理多个结果集

对查询的多个结果集的处理方法涉及使用在结果集间移动的另一个循环包装读取循环。

必须先准备好返回多个结果集的 SQL 语句，然后再执行。prepare 方法为该语句返回一个句柄。使用该句柄执行语句，然后检索关于结果集和每个结果集的行的元信息。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
```

```
my $sel_stmt = "SELECT ID, GivenName, Surname
                  FROM Customers
                  ORDER BY GivenName, Surname;
                SELECT *
                  FROM Departments
                  ORDER BY DepartmentID";
my %defaults = (
     AutoCommit => 0, # Require explicit commit or rollback.
     PrintError => 0
   );
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
  or die "Cannot connect to $data_src: $DBI::errstr\n";
&db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
      my($sel, $dbh) = @_;
      my($row, $sth) = undef;
      $sth = $dbh->prepare($sel);
      $sth->execute;
      do {
          print "Fields:     $sth->{NUM_OF_FIELDS}\n";
          print "Params:     $sth->{NUM_OF_PARAMS}\n\n";
          print join("\t\t", @{$sth->{NAME}}), "\n\n";
          while($row = $sth->fetchrow_arrayref) {
             print join("\t\t", @$row), "\n";
          }
          print "---end of results---\n\n";
      } while (defined $sth->more_results);
      $sth = undef;
}
__END__
```

如何使用 Perl DBI 插入行

插入行需要打开的连接的句柄。最简单的方法是使用参数化的 INSERT 语句，这意味着问号用作值的占位符。首先准备好语句，然后对每一新行执行一次。新行的值作为参数提供给 execute 方法。

以下示例程序插入两个新的客户。尽管行值显示为文字字符串，您可能希望从文件中读取这些值。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $ins_stmt = "INSERT INTO Customers (ID, GivenName, Surname,
                        Street, City, State, Country, PostalCode,
                        Phone, CompanyName)
```

```
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
my %defaults = (
     AutoCommit => 0, # Require explicit commit or rollback.
     PrintError => 0
   );
my $dbh = DBI->connect($data_src, $uid, $pwd, \%defaults)
     or die "Can't connect to $data_src: $DBI::errstr\n";
&db_insert($ins_stmt, $dbh);
$dbh->commit;
$dbh->disconnect;
exit(0);

sub db_insert {
      my($ins, $dbh) = @_;
      my($sth) = undef;
      my @rows = (
        "801,Alex,Alt,5 Blue Ave,New York,NY,USA,
10012,5185553434,BXM",
        "802,Zach,Zed,82 Fair St,New York,NY,USA,
10033,5185552234,Zap"
       );
      $sth = $dbh->prepare($ins);
      my $row = undef;
      foreach $row ( @rows ) {
        my @values = split(/,/, $row);
        $sth->execute(@values);
      }
}
__END__
```

Python 支持

SAP Sybase IQ Python 数据库接口 (sqlanydb) 是用于 Python 语言的数据访问 API。本节介绍如何结合 Python 使用 SAP Sybase IQ。

sqlanydb

SQL Anywhere Python 数据库接口 (sqlanydb) 是用于 Python 语言的数据访问 API。Python 数据库 API 规范定义了一组方法，这些方法提供了一致的数据库接口,该接口独立于实际所用数据库。使用 sqlanydb 模块，您的 Python 脚本可以直接访问 SAP Sybase IQ 数据库服务器。

sqlanydb 模块及其扩展模块执行由 Marc-Andr¨¦ Lemburg 编写的 Python 数据库 API 规范 v2.0。安装完 sqlanydb 模块后，即可通过 Python 访问和更改 SAP Sybase IQ 数据库中的信息。

有关 Python 数据库 API 规范 v2.0 的信息，请参见 http://www.python.org/dev/peps/pep-0249/。

将 Python 用于多线程时，sqlanydb 模块是线程安全的。

要求
sqlanydb 模块要求具有以下组件：

- 需要 Python。有关支持版本的列表，请参见 http://www.sybase.com/detail?id=1068981。
- 需要 ctypes 模块。要测试 ctypes 模块是否存在，请打开命令提示窗口并运行 Python。
  在 Python 提示符处，输入以下语句：

  ```
  import ctypes
  ```

  如果看到错误消息，则表明 ctypes 不存在。以下是一个示例。

  ```
  >>> import ctypes
  Traceback (most recent call last):
    File "<stdin>", line 1, in ?
  ImportError: No module named ctypes
  ```

  如果 Python 安装目录中没有 ctypes，则安装 ctypes。可在以下网址的 SourceForge.net 文件部分找到安装程序：http://sourceforge.net/project/showfiles.php?group_id=71702。
  Peak EasyInstall 还会安装 ctypes。要下载 Peak EasyInstall，请转到 http://peak.telecommunity.com/DevCenter/EasyInstall。

在 Windows 上安装 Python 支持

要在 Windows 上安装 Python 支持，可从 SAP Sybase IQ 安装目录的 `SDK\Python` 子目录运行相应的 Python 安装脚本。

前提条件

确保 Python 和 ctypes 模块均已安装。有关受支持 Python 版本的列表，请参见 http://www.sybase.com/detail?id=1068981。

过程

1. 在系统命令提示符处，转到 SAP Sybase IQ 安装目录的 `SDK\Python` 子目录。
2. 运行以下命令安装 sqlanydb。

   ```
   python setup.py install
   ```
3. 要测试 sqlanydb，请在当前目录中创建示例数据库文件的副本，然后进行测试。

   ```
   newdemo
   cd "%ALLSERPROFILE%"\SybaseIQ\demo
   start_iq @iqdemo.cfg iqdemo.db
   python Scripts\test.py
   ```

   测试脚本同数据库服务器建立连接并执行 SQL 查询。如果成功，测试结果将会显示消息 `sqlanydb successfully installed.`

   如果测试未运行，请确保路径中包含 SAP Sybase IQ 安装目录的 `bin32` 或 `bin64` 子目录。

现在即可使用 sqlanydb 模块。

在 Unix 上安装 Python 支持

要在 Unix 上安装 Python 支持，可从 SAP Sybase IQ 安装目录的 `sdk/python` 子目录运行相应的 Python 安装脚本。

前提条件

确保 Python 和 ctypes 模块均已安装。有关受支持 Python 版本的列表，请参见 http://www.sybase.com/detail?id=1068981。

过程

1. 确保为 SAP Sybase IQ 设置相应的环境。

根据使用的 shell，输入合适的命令从 SAP Sybase IQ 安装目录中获取 SAP Sybase IQ 配置脚本（如果安装的是 64 位软件，则可能要用 `bin64` 来替代 `bin32`）：

shell	使用的命令
sh、ksh 或 bash	`.bin32/sa_config.sh`
csh 或 tcsh	source `bin32/sa_config.csh`

2. 在 shell 提示符处，转到 SAP Sybase IQ 安装目录的 `sdk/python` 子目录。
3. 输入以下命令安装 sqlanydb。

```
python setup.py install
```

4. 要测试 sqlanydb，请在当前目录中创建示例数据库文件的副本，然后进行测试。

```
newdemo
cd "%ALLSERPROFILE%"\SybaseIQ\demo
start_iq @iqdemo.cfg iqdemo.db
python scripts/test.py
```

测试脚本同数据库服务器建立连接并执行 SQL 查询。如果成功，测试结果将会显示消息 `sqlanydb successfully installed.`

如果测试未运行，请确保路径中包含 SAP Sybase IQ 安装目录的 `bin32` 或 `bin64` 子目录。

现在即可使用 sqlanydb 模块。

使用 sqlanydb 的 Python 脚本

本节概述如何编写使用 sqlanydb 接口的 Python 脚本。

API 的完整文档可从 http://www.python.org/dev/peps/pep-0249/ 获得。

sqlanydb 模块

要在 Python 脚本中使用 sqlanydb 模块，必须首先通过在文件顶部添加以下行来装载该模块。

```
import sqlanydb
```

如何使用 Python 打开和关闭数据库连接

通常，打开一个到数据库的连接，然后通过该连接运行一系列 SQL 语句来执行所有需要的操作。要打开连接，请使用 connect 方法。返回值是一个到数据库连接的句柄，使用该句柄可以在连接上执行后继操作。

connect 方法的参数指定为一系列以逗号分隔的“关键字=值”对。

```
sqlanydb.connect( keyword=value, ...)
```

下面是一些常用连接参数：

- **DataSourceName="*dsn*"** – 此连接参数的简写形式是 DSN="*dsn*"。例如，DataSourceName="Sybase IQ demo"。
- **UserID="*user-id*"** – 此连接参数的简写形式是 UID="*user-id*"。
- **Password="*passwd*"** – 此连接参数的简写形式是 PWD="*passwd*"。
- **DatabaseFile="*db-file*"** – 此连接参数的简写形式是 DBF="*db-file*"。例如，DatabaseFile="iqdemo.db"。

以下代码示例可打开和关闭到 SAP Sybase IQ 示例数据库的连接。必须先启动数据库服务器和示例数据库然后才能运行此脚本。

```
import sqlanydb

# Create a connection object
con = sqlanydb.connect( userid="<user_id>",
                        password="<password>" )
# Close the connection
con.close()
```

为避免手动启动数据库服务器，可以用经过配置的数据源启动服务器，如下例所示。

```
import sqlanydb

# Create a connection object
con = sqlanydb.connect( DSN="Sybase IQ Demo" )

# Close the connection
con.close()
```

如何使用 Python 获取结果集

获得了打开的连接的句柄后，您可以访问和修改存储在数据库中的数据。可能最简单的操作是检索某些行并输出它们。

cursor 方法用于在打开的连接上创建游标。execute 方法用于创建结果集。fetchall 方法用于获取此结果集中的行。

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>",
                        password="<password>" )
cursor = con.cursor()

# Execute a SQL string
sql = "SELECT * FROM Employees"
cursor.execute(sql)

# Get a cursor description which contains column names
desc = cursor.description
print len(desc)

# Fetch all results from the cursor into a sequence,
# display the values as column name=value pairs,
# and then close the connection
rowset = cursor.fetchall()
```

```
for row in rowset:
    for col in range(len(desc)):
        print "%s=%s" % (desc[col][0], row[col] )
    print
cursor.close()
con.close()
```

如何使用 Python 插入行

在表中插入行最简单的方法是使用非参数化 INSERT 语句，这意味着值作为 SQL 语句的一部分来指定。会为每个新行构建和执行一条新语句。在前面的示例中，需要游标才能执行 SQL 语句。

以下示例程序在示例数据库中插入两名新客户。断开连接前，它将事务提交到数据库。

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>", pwd="<password>" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801,'Alex','Alt','5 Blue Ave','New York','NY',
         'USA','10012','5185553434','BXM'),
         (802,'Zach','Zed','82 Fair St','New York','NY',
         'USA','10033','5185552234','Zap'))

# Set up a SQL INSERT
parms = ("'%s'," * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
print sql % rows[0]
cursor.execute(sql % rows[0])
print sql % rows[1]
cursor.execute(sql % rows[1])
cursor.close()
con.commit()
con.close()
```

另一种方法是使用参数化的 INSERT 语句，这意味着问号用作值的占位符。executemany 方法用于为每个行集成员执行 INSERT 语句。新行的值作为单个参数提供给 executemany 方法。

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>", pwd="<password>" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801,'Alex','Alt','5 Blue Ave','New York','NY',
         'USA','10012','5185553434','BXM'),
         (802,'Zach','Zed','82 Fair St','New York','NY',
         'USA','10033','5185552234','Zap'))
```

```
# Set up a parameterized SQL INSERT
parms = ("?," * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
print sql
cursor.executemany(sql, rows)
cursor.close()
con.commit()
con.close()
```

虽然两个示例似乎都是将行数据插入表中的较为合适的方法，但后一种方法更佳，原因有两点。如果数据值是通过提示来输入而获取的，在注入包含 SQL 语句的游荡数据时，第一个示例容易受到影响。第一个示例需要为每个要插入到表中的行调用 execute 方法。而第二个示例只调用一次 executemany 方法，就可将所有行插入到表中。

数据库类型转换

要控制在从数据库服务器读取结果时如何将数据库类型映射到 Python 对象中，可以注册转换回调。

使用模块级的 register_converter 方法来注册回调。调用此方法时将以数据库类型为第一参数，以转换函数为第二参数。例如，要请求 sqlanydb 为任意列中描述为 DT_DECIMAL 类型的数据创建 Decimal 对象，可以使用下例：

```
import sqlanydb
import decimal

def convert_to_decimal(num):
    return decimal.Decimal(num)

sqlanydb.register_converter(sqlanydb.DT_DECIMAL,
convert_to_decimal)
```

转换器可以注册为下列数据库类型：

```
DT_DATE
DT_TIME
DT_TIMESTAMP
DT_VARCHAR
DT_FIXCHAR
DT_LONGVARCHAR
DT_DOUBLE
DT_FLOAT
DT_DECIMAL
DT_INT
DT_SMALLINT
DT_BINARY
DT_LONGBINARY
DT_TINYINT
DT_BIGINT
DT_UNSINT
DT_UNSSMALLINT
DT_UNSBIGINT
DT_BIT
```

下面的示例演示了如何截去小数位后的数字，将带小数的结果转换为整数。应用程序运行时显示的薪水金额是整数值。

```
import sqlanydb

def convert_to_int(num):
    return int(float(num))

sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_int)

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>",
                        password="<password>" )
cursor = con.cursor()

# Execute a SQL string
sql = "SELECT * FROM Employees WHERE EmployeeID=105"
cursor.execute(sql)

# Get a cursor description which contains column names
desc = cursor.description
print len(desc)

# Fetch all results from the cursor into a sequence,
# display the values as column name=value pairs,
# and then close the connection
rowset = cursor.fetchall()
for row in rowset:
    for col in range(len(desc)):
        print "%s=%s" % (desc[col][0], row[col] )
    print
cursor.close()
con.close()
```

PHP 支持

PHP 提供在多个常用数据库中检索信息的功能。SAP Sybase IQ 包括可通过 PHP 访问 SAP Sybase IQ 数据库的模块。您可使用 PHP 语言在 SAP Sybase IQ 数据库中检索信息，并在您自己的 Web 站点上提供动态 Web 内容。

SAP Sybase IQ PHP 扩展

PHP 是 `PHP: Hypertext Preprocessor`（超文本预处理器）的缩写，是一种开放源代码的脚本编写语言。虽然可以将其用作通用目的脚本编写语言，但是其设计目的是成为一种用来编写可嵌入到 HTML 文档中的脚本的方便语言。与客户端经常执行的用 JavaScript 编写的脚本不同，PHP 脚本由 Web 服务器处理，然后将生成的 HTML 输出发送到客户端。PHP 的语法是由其它流行语言（如 Java 和 Perl）的语法派生出来的。

为了使其成为一种可以开发动态 Web 页的便利语言，PHP 提供了从许多常见数据库（如 SAP Sybase IQ）检索信息的功能。SAP Sybase IQ 中带有可以从 PHP 访问 SAP Sybase IQ 数据库的扩展。您可以使用 SAP Sybase IQ PHP 扩展和 PHP 语言来编写独立脚本，并创建依赖于存储在 SAP Sybase IQ 数据库中的信息的动态 Web 页。

SAP Sybase IQ PHP 扩展提供了一种从 PHP 访问数据库的内在方法。与其它 PHP 数据访问技术相比，您可能更喜欢此技术，因为它很简单，而且可帮助您避免使用其它技术可能发生的系统资源泄露。

为 Windows、Linux 和 Solaris 提供 PHP 扩展的预建版本，并将其安装在 SAP Sybase IQ 安装目录的二进制子目录中。SAP Sybase IQ PHP 扩展的源代码安装在 SAP Sybase IQ 安装目录的 `sdk\php` 子目录中。

有关更多信息和最新的 SAP Sybase IQ PHP 驱动程序，请参见 http://www.sybase.com/detail?id=1019698。

测试 PHP 扩展

执行快速检查，验证 SAP Sybase IQ PHP 扩展是否正常工作。

前提条件

必须在系统中安装所有必需的 PHP 组件

过程

1. 确保路径中包含 SAP Sybase IQ 安装目录的 `bin32` 子目录。SAP Sybase IQ PHP 扩展要求路径中包含 `bin32` 目录。

2. 在命令提示符处运行以下命令以启动 SAP Sybase IQ 示例数据库。

```
cd "%ALLUSERSPROFILE%"\SybaseIQ\demo
start_iq @iqdemo.cfg iqdemo.db
```

   该命令使用示例数据库启动数据库服务器。

3. 在命令提示符处，转到 SAP Sybase IQ 安装目录的 `SDK\PHP\Examples` 子目录。请确保将 `php` 可执行目录包含在您的路径中。输入以下命令：

```
php test.php
```

   应该会出现类似以下内容的消息。如果无法识别此 PHP 命令，请验证它是否在路径中。

```
Installation successful
Using php-5.2.11_sqlanywhere.dll
Connected successfully
```

   如果未装载 SAP Sybase IQ PHP 扩展，则可通过 "php -i" 命令获得有关 PHP 安装的帮助信息。在此命令的输出中搜索 `extension_dir` 和 `sqlanywhere`。

4. 完成后请单击数据库服务器消息窗口中的**“关闭”**来停止 SAP Sybase IQ 数据库服务器。

测试应成功，表示 SAP Sybase IQ PHP 扩展工作正常。

创建和运行 PHP 测试页

创建并运行多个 Web 页，以测试 PHP 是否设置正确。

前提条件

必须安装 PHP。有关安装 PHP 的信息，请参见 http://us2.php.net/install。

过程

此过程适用于所有配置。

1. 在 Web 内容的根目录中创建一个名为 `info.php` 的文件。

   如果您不能确定要使用哪个目录，请检查 Web 服务器的配置文件。在 Apache 安装目录中，内容目录名通常为 `htdocs`。

2. 将以下代码插入到此文件中：

```
<?php phpinfo(); ?>
```

   PHP 函数 phpinfo 可生成一个系统设置信息页。这可确认安装的 PHP 和 Web 服务器是否能够一起正常工作。

3. 将 `connect.php` 文件从 `sdk\php\examples` 目录复制到 Web 内容的根目录中。这可确认安装的 PHP 和 SQL Anywhere 是否能够一起正常工作。

4. 在 Web 内容的根目录下创建一个名为 `sa_test.php` 的文件，然后将以下代码插入此文件：

```
<?php
  $conn = sasql_connect( "UID=DBA;PWD=sql" );
  $result = sasql_query( $conn, "SELECT * FROM Employees" );
  sasql_result_all( $result );
  sasql_free_result( $result );
  sasql_disconnect( $conn );
?>
```

sa_test 页将显示 Employees 表的内容。

5. 如果需要，请启动 Web 服务器。

   例如，要启动 Apache Web 服务器，可从 Apache 安装目录的 `bin` 子目录运行以下命令：

   ```
   apachectl start
   ```

6. 在 Linux 上，使用提供的脚本之一设置 SAP Sybase IQ 环境变量。

   根据您使用的 shell，输入相应的命令从 SAP Sybase IQ 安装目录执行 SAP Sybase IQ 配置脚本：

shell	...使用此命令
sh、ksh 或 bash	`. /bin32/sa_config.sh`
csh 或 tcsh	`source /bin32/sa_config.csh`

7. 在命令提示符处，启动 iqdemo.db 示例数据库。
8. 要测试 PHP 和 Web 服务器是否与 SAP Sybase IQ 一起正常工作，请通过服务器所在计算机上运行的浏览器来访问测试页：

测试页	使用的 URL
`info.php`	`http://localhost/info.php`
`connect.php`	`http://localhost/connect.php`
`sa_test.php`	`http://localhost/sa_test.php`

信息页将显示 `phpinfo()` 调用的输出。

连接页将显示消息 `Connected successfully`。

sa_test 页将显示 Employees 表的内容。

PHP 脚本开发

本节介绍如何编写使用 SAP Sybase IQ PHP 扩展访问 SAP Sybase IQ 数据库的 PHP 脚本。

上述示例及其它示例的源代码位于 SAP Sybase IQ 安装目录的 `SDK\PHP\Examples` 子目录中。

如何使用 PHP 连接到数据库

要与数据库建立连接，可将标准的 SAP Sybase IQ 连接字符串作为 sasql_connect 函数的参数传递给数据库服务器。<?php 和 ?> 标志指示 Web 服务器应允许 PHP 执行这两个标志之间的代码，并用 PHP 输出替换这些代码。

本示例的源代码包含在 SAP Sybase IQ 安装目录中名为 `connect.php` 的文件中。

```
<?php
  # Connect using the default user ID and password
  $conn = sasql_connect( "UID=DBA;PWD=sql" );
  if( ! $conn ) {
      echo "Connection failed\n";
  } else {
      echo "Connected successfully\n";
      sasql_close( $conn );
  }?>
```

此脚本会尝试与本地服务器上的数据库建立连接。要使此代码成功，必须在本地服务器上启动 SAP Sybase IQ 示例数据库或有相同证书的数据库。

如何使用 PHP 检索数据库中的数据

PHP 脚本在 Web 页中的用途之一是检索并显示数据库中包含的信息。以下示例介绍了一些非常有用的技术。

简单选择查询

以下 PHP 代码介绍了一种在 Web 页中包含 SELECT 语句的结果集的便捷方法。此示例用于连接到 SAP Sybase IQ 示例数据库并返回客户列表。

此代码可以嵌入到 Web 页中，条件是 Web 服务器已配置为可以执行 PHP 脚本。

本示例的源代码包含在 SAP Sybase IQ 安装目录中名为 `query.php` 的文件中。

```
<?php
  # Connect using the default user ID and password
  $conn = sasql_connect( "UID=DBA;PWD=sql" );
  if( ! $conn ) {
      echo "sasql_connect failed\n";
  } else {
      echo "Connected successfully\n";
      # Execute a SELECT statement
      $result = sasql_query( $conn, "SELECT * FROM Customers" );
      if( ! $result ) {
          echo "sasql_query failed!";
      } else {
          echo "query completed successfully\n";
          # Generate HTML from the result set
          sasql_result_all( $result );
          sasql_free_result( $result );
      }
      sasql_close( $conn );
  }
?>
```

sasql_result_all 函数可读取结果集中的所有行，并生成一个 HTML 输出表来显示这些行。sasql_free_result 函数释放用来存储结果集的资源。

按列名读取

某些情况下，您可能不想显示结果集中的所有数据，或是希望以其它方式显示这些数据。下面的示例说明如何对结果集的输出格式进行更好地控制。PHP 允许您以选择的任何方式显示任意数量的信息。

本示例的源代码包含在 SAP Sybase IQ 安装目录中名为 `fetch.php` 的文件中。

```
<?php
  # Connect using the default user ID and password
  $conn = sasql_connect( "UID=DBA;PWD=sql" );
  if( ! $conn ) {
      die ("Connection failed");
  } else {
      # Connected successfully.
  }
  # Execute a SELECT statement
  $result = sasql_query( $conn, "SELECT * FROM Customers" );
  if( ! $result ) {
      echo "sasql_query failed!";
      return 0;
  } else {
      echo "query completed successfully\n";
  }
  # Retrieve meta information about the results
  $num_cols = sasql_num_fields( $result );
  $num_rows = sasql_num_rows( $result );
  echo "Num of rows = $num_rows\n";
  echo "Num of cols = $num_cols\n";
  while( ($field = sasql_fetch_field( $result )) ) {
      echo "Field # : $field->id \n";
      echo "\tname   : $field->name \n";
      echo "\tlength : $field->length \n";
      echo "\ttype   : $field->type \n";
  }
  # Fetch all the rows
  $curr_row = 0;
  while( ($row = sasql_fetch_row( $result )) ) {
      $curr_row++;
      $curr_col = 0;
      while( $curr_col < $num_cols ) {
          echo "$row[$curr_col]\t|";
          $curr_col++;
      }
      echo "\n";
  }
  # Clean up.
  sasql_free_result( $result );
  sasql_disconnect( $conn );
?>
```

sasql_fetch_array 函数从表中返回单行。可以按列名和列索引检索这些数据。

asql_fetch_assoc 函数从表中返回单行作为一个联合数组。可以使用列名作为索引来检索这些数据。以下是一个示例。

```
<?php
  # Connect using the default user ID and password
  $conn = sasql_connect("UID=DBA;PWD=sql");

  /* check connection */
  if( sasql_errorcode() ) {
      printf("Connect failed: %s\n", sasql_error());
      exit();
  }

  $query = "SELECT Surname, Phone FROM Employees ORDER by
EmployeeID";

  if( $result = sasql_query($conn, $query) ) {

      /* fetch associative array */
      while( $row = sasql_fetch_assoc($result) ) {
          printf ("%s (%s)\n", $row["Surname"], $row["Phone"]);
      }

      /* free result set */
      sasql_free_result($result);
  }

  /* close connection */
  sasql_close($conn);
?>
```

在 PHP 接口中还提供了其它两个类似的方法：sasql_fetch_row 返回一个仅能按列索引加以搜索的行，而 sasql_fetch_object 则返回一个仅能按列名搜索的行。

有关 sasql_fetch_object 函数的示例，请参见 `fetch_object.php` 示例脚本。

嵌套结果集

向数据库发送一条 SELECT 语句时，会返回一个结果集。sasql_fetch_row 和 sasql_fetch_array 函数从结果集的各行检索数据，然后将每行作为可以进一步查询的列数组返回。

本示例的源代码包含在 SAP Sybase IQ 安装目录中名为 `nested.php` 的文件中。

```
<?php
  $conn = sasql_connect( "UID=DBA;PWD=sql" );
  if( $conn ) {
      // get the GROUPO user id
      $result = sasql_query( $conn,
            "SELECT user_id FROM SYS.SYSUSER " .
            "WHERE user_name='GROUPO'" );
      if( $result ) {
          $row = sasql_fetch_array( $result );
          $user = $row[0];
      } else {
          $user = 0;
```

```
      }
      // get the tables created by user GROUPO
      $result = sasql_query( $conn,
            "SELECT table_id, table_name FROM SYS.SYSTABLE " .
            "WHERE creator = $user" );
      if( $result ) {
          $num_rows = sasql_num_rows( $result );
          echo "Returned rows : $num_rows\n";
          while( $row = sasql_fetch_array( $result ) ) {
              echo "Table: $row[1]\n";
            $query = "SELECT table_id, column_name FROM SYS.SYSCOLUMN
" .
                        "WHERE table_id = '$row[table_id]'";
              $result2 = sasql_query( $conn, $query );
              if( $result2 ) {
                  echo "Columns:";
                while( $detailed = sasql_fetch_array( $result2 ) ) {
                      echo " $detailed[column_name]";
                  }
                  sasql_free_result( $result2 );
              }
              echo "\n\n";
          }
          sasql_free_result( $result );
      }
      sasql_disconnect( $conn );
  }
?>
```

在上例中，SQL 语句从 SYSTAB 中为每个表选择表 ID 和名称。sasql_query function 函数返回一个行数组。该脚本使用 sasql_fetch_array 函数迭代通过各行，以从数组检索行。一个内部迭代遍历每行中的列并显示其值。

Web 表单

PHP 可以从 Web 表单获取用户输入，将用户输入作为 SQL 查询传递给数据库服务器，并显示返回的结果。以下示例介绍了一个简单的 Web 表单，该表单使用户能够使用 SQL 语句查询示例数据库，并将结果显示在一个 HTML 表中。

本示例的源代码包含在 SAP Sybase IQ 安装目录中名为 `webisql.php` 的文件中。

```
<?php
  echo "<HTML>\n";
  $qname = $_POST["qname"];
  $qname = str_replace( "\\", "", $qname );
  echo "<form method=post action=webisql.php>\n";
 echo "<br>Query: <input type=text Size=80 name=qname value=\"$qname
\">\n";
  echo "<input type=submit>\n";
  echo "</form>\n";
  echo "<HR><br>\n";
  if( ! $qname ) {
      echo "No Current Query\n";
      return;
  }
```

```
  # Connect to the database
  $con_str =
"UID=<user_id>;PWD=<password>;SERVER=iqdemo;LINKS=tcpip";
  $conn = sasql_connect( $con_str );
  if( ! $conn ) {
      echo "sasql_connect failed\n";
      echo "</html>\n";
      return 0;
  }
  $qname = str_replace( "\\", "", $qname );
  $result = sasql_query( $conn, $qname );
  if( ! $result ) {
        echo "sasql_query failed!";
  } else {
        // echo "query completed successfully\n";
        sasql_result_all( $result, "border=1" );
      sasql_free_result( $result );
  }
  sasql_disconnect( $conn );
  echo "</html>\n";
?>
```

通过基于用户输入的值的公式化自定义 SQL 查询，可以扩展这一设计，从而处理复杂的 Web 表单。

PHP 应用程序中的 BLOB

SAP Sybase IQ 数据库可以将任何类型的数据作为一个二进制大对象 (BLOB) 存储。如果 Web 浏览器可以读取该类型的数据，则 PHP 脚本可以从数据库方便地检索数据，并将其显示在动态生成的页上。

BLOB 字段通常用于存储非文本数据（如 GIF 或 JPG 格式的图像）。许多类型的数据无需使用第三方软件或通过数据类型转换，即可传递到 Web 浏览器。以下示例说明了将一个图像添加到数据库中，然后对其进行再次检索以在 Web 浏览器中显示的过程。

此示例类似于 SAP Sybase IQ 安装目录中的文件 `image_insert.php` 和 `image_retrieve.php` 中的示例代码。这些示例也说明了如何使用 BLOB 列存储图像。

```
<?php
  $conn = sasql_connect( "UID=DBA;PWD=sql" )
        or die("Cannot connect to database");
  $create_table = "CREATE TABLE images (ID INTEGER PRIMARY KEY, img
IMAGE)";
  sasql_query( $conn, $create_table);
  $insert = "INSERT INTO images VALUES (99,
xp_read_file('ianywhere_logo.gif'))";
  sasql_query( $conn, $insert );
  $query = "SELECT img FROM images WHERE ID = 99";
  $result = sasql_query($conn, $query);
  $data = sasql_fetch_row($result);
  $img = $data[0];
  header("Content-type: image/gif");
```

```
  echo $img;
  sasql_disconnect($conn);
?>
```

为了能够将二进制数据从数据库直接发送到 Web 浏览器，脚本必须使用标头函数设置数据的 MIME 类型。在此示例中，系统会告诉浏览器该数据是一个 GIF 图像，以便浏览器正确显示该数据。

如何在 Unix 上构建 SAP Sybase IQ PHP 扩展

要在 Unix 上使用 SAP Sybase IQ PHP 扩展将 PHP 连接到 SAP Sybase IQ，必须将 SAP Sybase IQ PHP 扩展文件添加到 PHP 的源树，然后重新编译 PHP。

在 Unix 上将 SAP Sybase IQ PHP 扩展文件添加到 PHP 源树

本主题介绍了将 SAP Sybase IQ PHP 扩展文件添加到 PHP 源树所需的步骤。

前提条件

下面列出了要实现在 Unix 上使用 SAP Sybase IQ PHP 扩展而需要在系统上安装的软件：

- SAP Sybase IQ 的安装（可以与 Apache Web 服务器在同一台计算机上进行，也可以不在同一台计算机上进行）。
- SQL Anywhere PHP 扩展的源代码（可从 http://download.sybase.com/ianywhere/php/2.0.3/src/sasql_php.zip 下载）。
  您还需要安装 `sqlpp` 和 `libdblib16.so` (Unix)（请检查 SAP Sybase IQ `lib32` 目录）。
- PHP 源代码（可从 http://www.php.net 下载）。
  有关支持版本的列表，请参见 http://www.sybase.com/detail?id=1068981。
- Apache Web 服务器源代码（可从 http://httpd.apache.org 下载）。
  如果您打算使用 Apache 的预建版本，请确保您已安装 `apache` 和 `apache-devel`。
- 如果打算使用统一 ODBC PHP 扩展，则需要安装 `libdbodbc16.so` (Unix)（请检查 SAP Sybase IQ 的 `lib32` 目录）。

应该从您的 Unix 安装磁盘中安装以下二进制文件（如果还未安装它们），并且可以 RPM 的形式找到这些二进制文件：

make
automake
autoconf
makeinfo
bison
gcc
cpp

glibc-devel
kernel-headers
flex

要执行特定的安装步骤，必须具有与安装 PHP 的用户相同的访问特权。大多数基于 Unix 的系统都会提供一个 sudo 命令，该命令允许那些权限不够的用户如同具有足够权限的用户那样执行特定的命令。

过程

要执行特定的安装步骤，必须具有与安装 PHP 的用户相同的访问特权。大多数基于 Unix 的系统都会提供一个 sudo 命令，该命令允许那些权限不够的用户如同具有足够权限的用户那样执行特定的命令。

1. 从 http://www.sybase.com/detail?id=1019698 下载 SAP Sybase IQ PHP 扩展源代码。查找标题为 **“Building the Driver from Source”** 的部分。
2. 将文件从保存 SAP Sybase IQ PHP 扩展的目录抽取到 PHP 源树的 ext 子目录中：

   ```
   $ tar -xzf sasql_php.zip -C PHP-source-directory/ext/
   ```

   以下示例适用于 PHP 5.2.11 版。必须将下面的 php-5.2.11 改为正在使用的 PHP 版本。

   ```
   $ tar -xzf sqlanywhere_php-1.0.8.tar.gz -C ~/php-5.2.11/ext
   ```
3. 让 PHP 接受该扩展：

   ```
   $ cd PHP-source-directory/ext/sqlanywhere
   $ touch *
   $ cd ~/PHP-source-directory
   $ ./buildconf
   ```

   以下示例适用于 PHP 5.2.11 版。必须将下面的 php-5.2.11 改为正在使用的 PHP 版本。

   ```
   $ cd ~/php-5.2.11/ext/sqlanywhere
   $ touch *
   $ cd ~/php-5.2.11
   $ ./buildconf
   ```
4. 验证 PHP 是否接受该扩展：

   ```
   $ ./configure -help | egrep sqlanywhere
   ```

如果已成功让 PHP 接受了 SAP Sybase IQ 扩展，则应看到以下文本：

```
--with-sqlanywhere=[DIR]
```

如果未成功，则跟踪此命令的输出并将其发布到 SQL Anywhere 论坛 (http://sqlanywhere-forum.sybase.com/) 以寻求帮助。

如何编译 Apache 和 PHP

PHP 可作为 Web 服务器（如 Apache）或 CGI 可执行程序的共享模块进行编译。如果正在使用 PHP 不支持的 Web 服务器，或者要在命令 shell 中而不在 Web 页上执行 PHP

脚本，则应将 PHP 作为 CGI 可执行文件进行编译。否则，如果要安装 PHP 与 Apache 一起使用，则需将其作为 Apache 模块进行编译。

将 PHP 作为 Apache 模块进行编译
要安装 PHP 与 Apache 一起使用，需将其作为 Apache 模块进行编译。

前提条件

使用以下步骤配置 Apache，使其能够识别共享模块。

- 配置 Apache 以识别共享模块。
  在抽取 Apache 文件的目录中执行与以下命令类似的命令：

  ```
  $ cd Apache-source-directory
  $ ./configure --enabled-shared=max --enable-module=most --
  prefix=/Apache-installation-directory
  ```

  以下示例适用于 Apache 版本 2.2.9。必须将 `apache_2.2.9` 更改为正在使用的 Apache 版本。

  ```
  $ cd ~/apache_2.2.9
  $ ./configure --enabled-shared=max --enable-module=most --
  prefix=/usr/local/web/apache
  ```
- 重新编译和安装相关组件：

  ```
  $ make
  $ make install
  ```

  现在，可以编译 PHP 来将其作为 Apache 模块运行。

过程

1. 确保为 SAP Sybase IQ 设置相应的环境。

   根据您使用的 shell，从 SAP Sybase IQ 的安装目录输入相应的命令。

如果正在使用此 shell...	...使用此命令
sh、ksh、bash	`../IQ_16.sh`
csh、tcsh	`./IQ_16.csh`

2. 将 PHP 作为 Apache 模块进行配置以包括 SAP Sybase IQ PHP 扩展。

   运行以下命令：

   ```
   $ cd PHP-source-directory
   $ ./configure --with-sqlanywhere --with- apxs=/Apache-
   installation-directory/bin/apxs
   ```

   以下示例适用于 PHP 5.2.11 版。必须将 php-5.2.11 改为为正在使用的 PHP 版本。

   ```
   $ cd ~/php-5.2.11
   ```

```
$ ./configure --with-sqlanywhere --with- apxs=/usr/local/web/
apache/bin/apxs
```

`configure` 脚本将尝试确定 SAP Sybase IQ 安装的版本及位置。

3. 重新编译相关组件：

```
$ make
```

4. 检查是否已正确链接到库。
   - Linux 用户（以下示例假定您正在使用 PHP 版本 5）：

     ```
     ldd ./.libs/libphp5.so
     ```

5. 在 Apache 的 `lib` 目录中安装 PHP 二进制文件：

```
$ make install
```

6. 执行验证。PHP 会自动执行此操作。只需确保 `httpd.conf` 配置文件经过验证以使 Apache 承认 `.php` 文件作为 PHP 脚本。

   `httpd.conf` 存储在 `Apache` 目录的 `conf` 子目录中：

   ```
   $ cd Apache-installation-directory/conf
   ```

   例如：

   ```
   $ cd /usr/local/web/apache/conf
   ```

   在编辑文件前制作 `httpd.conf` 的一个备份副本（可使用所选的文本编辑器替换 `pico`）：

   ```
   $ cp httpd.conf httpd.conf.backup
   $ pico httpd.conf
   ```

   在 `httpd.conf` 中添加以下行或取消以下行的注释（这些行在文件中不在一起）：

   ```
   LoadModule php5_module   libexec/libphp5.so
   AddModule mod_php5.c
   AddType application/x-httpd-php .php
   AddType application/x-httpd-php-source .phps
   ```

   开头两行会将 Apache 指向用于解释 PHP 代码的文件，而另外两行声明扩展名为 `.php` 或 `.phps` 的文件的文件类型，以便 Apache 能够识别并正确地处理它们。

PHP 已成功编译为共享模块。

将 PHP 作为 CGI 可执行文件进行编译

如果正在使用 PHP 不支持的 Web 服务器，或者要在命令 shell 中而不在 Web 页上执行 PHP 脚本，则应将 PHP 作为 CGI 可执行文件进行编译。

前提条件

执行此任务没有前提条件。

过程

1. 确保为 SAP Sybase IQ 设置相应的环境。

   根据您使用的 shell，从 SAP Sybase IQ 的安装目录输入相应的命令。

如果正在使用此 shell...	...使用此命令
sh、ksh、bash	`../IQ_16.sh`
csh、tcsh	`./IQ_16.csh`

2. 将 PHP 作为 CGI 可执行文件进行配置，与 SAP Sybase IQ PHP 扩展一起使用。

   从提取 PHP 文件的目录中运行以下命令：

```
$ cd PHP-source-directory
$ ./configure --with-sqlanywhere
```

   以下示例适用于 PHP 5.2.11 版。必须将 php-5.2.11 改为为正在使用的 PHP 版本。

```
$ cd ~/php-5.2.11
$ ./configure --with-sqlanywhere
```

   配置脚本将尝试确定 SAP Sybase IQ 安装的版本及位置。

3. 编译可执行文件：

```
$ make
```

4. 安装组件。

```
$ make install
```

PHP 已成功编译为 CGI 可执行文件。

SAP Sybase IQ PHP API 参考

PHP API 支持以下函数：

sasql_affected_rows

返回受上一 SQL 语句影响的行数。此函数通常用于 INSERT、UPDATE 或 DELETE 语句。对于 SELECT 语句，使用 sasql_num_rows 函数。

原型

```
int sasql_affected_rows( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
受影响的行数。

sasql_commit

结束 SQL Anywhere 数据库上的一个事务，并使该事务期间所做的所有更改永久生效。仅当 auto_commit 选项为 Off 时才有效

原型

```
bool sasql_commit( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_close

关闭先前打开的数据库连接。

原型

```
bool sasql_close( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_connect

建立一个与 SAP Sybase IQ 数据库的连接。

原型

```
sasql_conn sasql_connect( string $con_str )
```

参数

$con_str – 一个由 SAP Sybase IQ 识别的连接字符串。

返回值
一个正的 SAP Sybase IQ 连接资源（成功时），或一个错误和 0（失败时）。

sasql_data_seek

将光标定位到使用 sasql_query 打开的 *$result* 的第 *row_num* 行上。

原型

```
bool sasql_data_seek( sasql_result $result, int row_num )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

row_num – 结果资源中表示游标的新位置的整数。例如，指定 0 将游标移动到结果集的第一行，指定 5 将游标移动到第六行。负数表示相对于结果集结尾的行。例如，-1 将游标移动到结果集的最后一行，-2 将游标移动到倒数第二行。

返回值
TRUE（成功时）或 FALSE（错误时）。

sasql_disconnect

关闭一个已经用 sasql_connect 或 sasql_pconnect 打开的连接。

原型

```
bool sasql_disconnect( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
TRUE（成功时）或 FALSE（错误时）。

sasql_error

返回最近执行的 SAP Sybase IQ PHP 函数的错误文本。错误消息按连接进行存储。如果未指定 *$conn*，则 sasql_error 返回没有可用连接的最后一条错误消息。例如，如果您调用 sasql_connect，但连接失败，则可调用不含 *$conn* 参数的 sasql_error 以获取错误消息。要获取相应的错误代码值，请使用 sasql_errorcode 函数。

原型

```
string sasql_error( [ sasql_conn $conn ] )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
描述错误的字符串。

sasql_errorcode

返回最近执行的 SAP Sybase IQ PHP 函数的错误代码。错误代码按连接进行存储。如果未指定 *$conn*，则 sasql_errorcode 返回没有可用连接的最后一个错误代码。例如，如果您调用 sasql_connect，但连接失败，则可调用不含 *$conn* 参数的 sasql_errorcode 以获取错误代码。要获取相应的错误消息，请使用 sasql_error 函数

原型

```
int sasql_errorcode( [ sasql_conn $conn ] )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
一个代表错误代码的整数。错误代码 0 表示成功。正的错误代码表示成功但有警告。负的错误代码表示失败。

sasql_escape_string

转义所提供的字符串中的所有特殊字符。要转义的特殊字符是 \r、\n、'、"、;、\ 和 NULL 字符。此函数是 sasql_real_escape_string 的别名。

原型

```
string sasql_escape_string( sasql_conn $conn, string $str )
```

参数

$conn – 由连接函数返回的连接资源。

$string – 要转义的字符串。

返回值
已转义字符串。

sasql_fetch_array

从结果集读取一行。该行作为一个可以按列名或列索引进行索引的数组返回。

原型

```
array sasql_fetch_array( sasql_result $result [, int
$result_type ])
```

参数

$result – 由 sasql_query 函数返回的结果资源。

$result_type – 此可选参数是一个常量，它指示应该从当前行数据中生成何种类型的数组。参数的可能值是常量 SASQL_ASSOC、SASQL_NUM 或 SASQL_BOTH。它缺省为 SASQL_BOTH。

通过使用 SASQL_ASSOC 常量，此函数的行为将与 sasql_fetch_assoc 函数相同，而 SASQL_NUM 的行为则与 sasql_fetch_row 函数相同。最后一个选项 SASQL_BOTH 将创建一个具备这两种属性的单一数组。

返回值
一个表示结果集中的一行的数组，或 FALSE（无法获取行时）。

sasql_fetch_assoc

从结果集中读取一行作为联合数组。

原型

```
array sasql_fetch_assoc( sasql_result $result )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

返回值
代表从结果集中读取的行的字符串的联合数组，数组中的各个关键字代表结果集中的一个列的名称或 FALSE（如果结果集中不再有行）。

sasql_fetch_field

返回一个包含关于某个特定列的信息的对象。

原型

```
object sasql_fetch_field( sasql_result $result [, int
$field_offset ] )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

$field_offset – 一个整数，表示您希望在其中检索信息的列/字段。列的编号以零为基准；要获取第一列，应指定值 0。如果忽略此参数，则返回下一个字段对象。

返回值
一个具有以下属性的对象：

- **id** – 包含字段号。
- **name** – 包含字段名。
- **numeric** – 指出该字段是否是数值。
- **length** – 返回字段的本地存储大小。
- **type** – 返回字段类型。
- **native_type** – 返回字段的本地类型。这些值是 DT_FIXCHAR、DT_DECIMAL 或 DT_DATE。
- **precision** – 返回字段的数字精度。仅为 native_type 等于 DT_DECIMAL 的字段设置此属性。
- **scale** – 返回字段的数字小数位数。仅为 native_type 等于 DT_DECIMAL 的字段设置此属性。

sasql_fetch_object

从结果集中读取一行作为对象。

原型

```
object sasql_fetch_object( sasql_result $result  )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

返回值

代表从结果集中读取的行的对象，其中的每个属性名称都匹配着一个结果集的列名称或 FALSE（如果结果集中不再有行）。

sasql_fetch_row

从结果集读取一行。该行作为一个仅可按列索引进行索引的数组返回。

原型

```
array sasql_fetch_row( sasql_result $result )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

返回值

一个表示结果集中的一行的数组，或 FALSE（无法获取行时）。

sasql_field_count

返回最后一个结果所包含的列（字段）数。

原型

```
int sasql_field_count( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
一个正的列数，或 FALSE（如果 *$conn* 无效）。

sasql_field_seek

将字段游标设置为给定的偏移。下次调用 sasql_fetch_field 将检索与该偏移相关联的列的字段定义。

原型

```
bool sasql_field_seek( sasql_result $result, int $field_offset )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

$field_offset – 一个整数，表示您希望在其中检索信息的列/字段。列的编号以零为基准；要获取第一列，应指定值 0。如果忽略此参数，则返回下一个字段对象。

返回值
TRUE（成功时）或 FALSE（错误时）。

sasql_free_result

释放与从 sasql_query 返回的结果资源相关联的数据库资源。

原型

```
bool sasql_free_result( sasql_result $result )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

返回值
TRUE（成功时）或 FALSE（错误时）。

sasql_get_client_info

返回客户端的版本信息。

原型

```
string sasql_get_client_info( )
```

参数
无

返回值
代表 SQL Anywhere 客户端软件版本的字符串。返回的字符串采用的是 X.Y.Z.W 形式，其中 X 为主版本号，Y 为子版本号，Z 为补丁号，W 为内部版本号（如 10.0.1.3616）。

sasql_insert_id

返回最后插入到 IDENTITY 列或 DEFAULT AUTOINCREMENT 列中的值，如果是最后插入到了不含 IDENTITY 或 DEFAULT AUTOINCREMENT 列的表中，则返回零。提供 sasql_insert_id 函数是为了与 MySQL 数据库兼容。

原型

```
int sasql_insert_id( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
由前一个 INSERT 语句为 AUTOINCREMENT 列生成的 ID，如果最后的插入对 AUTOINCREMENT 列没有影响，则返回零。如果 *$conn* 无效，则该函数会返回 FALSE。

sasql_message

将一条消息写入服务器消息窗口。

原型

```
bool sasql_message( sasql_conn $conn, string $message )
```

参数

$conn – 由连接函数返回的连接资源。

$message – 要写入服务器消息窗口的消息。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_multi_query

使用提供的连接资源准备并执行由 *$sql_str* 指定的一个或多个 SQL 查询。查询之间使用分号进行分隔。第一个查询结果可使用 sasql_use_result 或 sasql_store_result 来检索或存储。sasql_field_count 可用于查看查询是否返回结果集。可使用 sasql_next_result 和 sasql_use_result/sasql_store_result 处理所有的后继查询结果。

原型

```
bool sasql_multi_query( sasql_conn $conn, string $sql_str )
```

参数

$conn – 由连接函数返回的连接资源。

$sql_str – 由分号分隔的一个或多个 SQL 语句。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_next_result

通过在 *$conn* 上执行的上一个查询准备下一个结果集。

原型

```
bool sasql_next_result( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
如果没有其它结果集要检索，则返回 FALSE。如果有其它结果要检索，则返回 TRUE。调用 sasql_use_result 或 sasql_store_result 来检索下一结果集。

sasql_num_fields

返回 *$result* 中的行所包含的字段数。

原型

```
int sasql_num_fields( sasql_result $result )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

返回值
返回指定结果集中的字段数。

sasql_num_rows

返回 *$result* 包含的行数。

原型

```
int sasql_num_rows( sasql_result $result )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

返回值
一个正数（如果行数是确切值），或一个负数（如果行数是估计值）。要获取确切的行数，必须针对数据库永久设置或针对连接临时设置数据库选项 row_counts。

sasql_pconnect

建立一个与 SAP Sybase IQ 数据库的持久连接。由于 Apache 创建子进程的方式，您可能会在使用 sasql_pconnect 代替 sasql_connect 时发现性能得到了提高。持久性连接可以通过与连接池相似的方式提高性能。如果数据库服务器具有的连接的数量有限（例如，个人数据库服务器限制为 10 个并发连接），则使用持久性连接时应当谨慎。持久性连接可以附加到每个子进程上，并且如果 Apache 中拥有的子进程数目超过可用连接的数目，您就会接收到连接错误。

原型

```
sasql_conn sasql_pconnect( string $con_str )
```

参数

$con_str – 一个由 SAP Sybase IQ 识别的连接字符串。

返回值
一个正的 SAP Sybase IQ 持久连接资源（成功时），或一个错误和 0（失败时）。

sasql_prepare

准备所提供的 SQL 字符串。

原型

```
sasql_stmt sasql_prepare( sasql_conn $conn, string $sql_str )
```

参数

$conn – 由连接函数返回的连接资源。

$sql_str – 要准备的 SQL 语句。此字符串可通过在适当位置嵌入问号来包含参数标记。

返回值
语句对象或 FALSE（失败时）。

sasql_query

准备并执行已经使用 sasql_connect 或 sasql_pconnect 打开并以 *$conn* 标识的连接上的 SQL 查询 *$sql_str*。sasql_query 函数等效于调用两个函数：sasql_real_query 和 sasql_store_result 或 sasql_use_result 中的一个。

原型

```
mixed sasql_query( sasql_conn $conn, string $sql_str [, int
$result_mode ] )
```

参数

$conn – 由连接函数返回的连接资源。

$sql_str – 一个 SQL Anywhere 支持的 SQL 语句。

$result_mode – SASQL_USE_RESULT 或 SASQL_STORE_RESULT（缺省值）。

返回值
失败时返回 FALSE；对于 INSERT、UPDATE、DELETE、CREATE，成功时返回 TRUE；对于 SELECT，返回 sasql_result。

sasql_real_escape_string

转义所提供的字符串中的所有特殊字符。要转义的特殊字符是 \r、\n、'、"、;、\ 和 NULL 字符。

原型

```
string sasql_real_escape_string( sasql_conn $conn, string $str )
```

参数

$conn – 由连接函数返回的连接资源。

$string – 要转义的字符串。

返回值
已转义字符串或 FALSE（错误时）。

sasql_real_query

使用提供的连接资源针对数据库执行查询。可使用 sasql_store_result 或 sasql_use_result 检索或存储查询结果。sasql_field_count 函数可用于查看查询是否返回结果集。sasql_query 函数等效于调用此函数以及 sasql_store_result 或 sasql_use_result 之一。

原型

```
bool sasql_real_query( sasql_conn $conn, string $sql_str )
```

参数

$conn – 由连接函数返回的连接资源。

$sql_str – 一个 SQL Anywhere 支持的 SQL 语句。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_result_all

读取 *$result* 的所有结果，并生成一个带可选格式字符串的 HTML 输出表。

原型

```
bool sasql_result_all( resource $result
[, $html_table_format_string
[, $html_table_header_format_string
[, $html_table_row_format_string
[, $html_table_cell_format_string
] ] ] ] )
```

参数

$result – 由 sasql_query 函数返回的结果资源。

$html_table_format_string – 一个适用于 HTML 表的格式字符串。例如，`"Border=1; Cellpadding=5"`。特殊值 none 不创建 HTML 表。这可用于自定义列名称或脚本。如不想为此参数指定显式值，请对此参数值使用 NULL。

$html_table_header_format_string – 一个适用于 HTML 表的列标题的格式字符串。例如，"bgcolor=#FF9533"。特殊值 none 不创建 HTML 表。这可用于自定义列名称或脚本。如不想为此参数指定显式值，请对此参数值使用 NULL。

$html_table_row_format_string – 一个适用于 HTML 表中的行的格式字符串。例如，"onclick='alert('this')'"。如果您希望交替使用不同的格式，请使用特殊标识 ><。标识的左侧指示对奇数行使用的格式，标识的右侧用于格式化偶数行。如果不在格式字符串中放入此标识，则所有行都使用相同的格式。如果您不想为此参数指定显式值，请对此参数值使用 NULL。

$html_table_cell_format_string – 一个适用于 HTML 表行中的单元格的格式字符串。例如，"onclick='alert('this')'"。如果您不想为此参数指定显式值，请对此参数值使用 NULL。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_rollback

结束数据库上的一个事务，并放弃该事务期间所做的所有更改。仅当 auto_commit 选项为 Off 时此函数才有效。

原型

```
bool sasql_rollback( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_set_option

在指定连接上设置指定选项的值。

原型

```
bool sasql_set_option( sasql_conn $conn, string $option, mixed
$value )
```

说明
可以为以下选项设置值：

名称	说明	缺省值
auto_commit	此选项设置为 on 时，数据库服务器在每执行一条语句后提交。	on

名称	说明	缺省值
row_counts	此选项设置为 FALSE 时，sasql_num_rows 函数返回受影响的行数估计值。要获得准确计数，请将此选项设置为 TRUE。	FALSE
verbose_errors	此选项设置为 TRUE 时，PHP 驱动程序返回详细的错误信息。此选项设置为 FALSE 时，必须调用 sasql_error 或 sasql_errorcode 函数以获取更详细的错误信息。	TRUE

可以通过在 `php.ini` 文件中加入以下行，更改选项的缺省值。在本示例中，为 auto_commit 选项设置了缺省值。

```
sqlanywhere.auto_commit=0
```

参数

$conn – 由连接函数返回的连接资源。

$option – 要设置的选项的名称。

$value – 新的选项值。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_affected_rows

返回受执行语句影响的行数。

原型

```
int sasql_stmt_affected_rows( sasql_stmt $stmt )
```

参数

$stmt – 由 sasql_stmt_execute 执行的语句资源。

返回值
受影响的行数或 FALSE（失败时）。

sasql_stmt_bind_param

将 PHP 变量绑定到语句参数。

原型

```
bool sasql_stmt_bind_param( sasql_stmt $stmt, string $types,
mixed &$var_1 [, mixed &$var_2 .. ] )
```

参数

$stmt – 由 sasql_prepare 函数返回的预准备语句资源。

$types – 指定相应绑定类型的包含一个或多个字符的字符串。此类型可以是以下各项之一：`s`（对于字符串）、`i`（对于整数）、`d`（对于双精度型）、`b`（对于 blob）。$types 字符串的长度必须与跟在 $types 参数 ($var_1, $var_2, ...) 后的参数的数量一致。字符数也应该与预准备语句中的参数标记（问号）的数量一致。

$var_n – 变量引用。

返回值
如果绑定成功则为 TRUE，否则为 FALSE。

sasql_stmt_bind_param_ex

将一个 PHP 变量绑定到语句参数。

原型

```
bool sasql_stmt_bind_param_ex( sasql_stmt $stmt, int
$param_number, mixed &$var, string $type [, bool $is_null [, int
$direction ] ] )
```

参数

$stmt – 由 sasql_prepare 函数返回的预准备语句资源。

$param_number – 参数编号。它应该是 0 和 (sasql_stmt_param_count($stmt) - 1) 之间的一个数字。

$var – PHP 变量。仅允许对 PHP 变量的引用。

$type – 变量的类型。此类型可以是以下各项之一：`s`（对于字符串）、`i`（对于整数）、`d`（对于双精度型）、`b`（对于 blob）。

$is_null – 不论变量值是否为 NULL。

$direction – 可以是 SASQL_D_INPUT、SASQL_D_OUTPUT 或 SASQL_INPUT_OUTPUT。

返回值
如果绑定成功则为 TRUE，否则为 FALSE。

sasql_stmt_bind_result

将一或多个 PHP 变量绑定到已执行的语句的结果列，并返回结果集。

原型

```
bool sasql_stmt_bind_result( sasql_stmt $stmt, mixed &$var1 [,
mixed &$var2 .. ] )
```

参数

$stmt – 由 sasql_stmt_execute 执行的语句资源。

$var1 – 对于由 sasql_stmt_fetch 返回的、将被绑定到结果集列的 PHP 变量的引用。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_close

关闭所提供的语句资源并释放任何与其相关联的资源。此函数还将释放由 sasql_stmt_result_metadata 返回的任何结果对象。

原型

```
bool sasql_stmt_close( sasql_stmt $stmt )
```

参数

$stmt – 由 sasql_prepare 函数返回的预准备语句资源。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_data_seek

此函数查找结果集中的指定偏移量。

原型

```
bool sasql_stmt_data_seek( sasql_stmt $stmt, int $offset )
```

参数

$stmt – 语句资源。

$offset – 结果集中的偏移量。它是 0 和 (sasql_stmt_num_rows($stmt) - 1) 之间的一个数字。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_errno

返回最近使用指定的语句资源执行的语句函数的错误代码。

原型

```
int sasql_stmt_errno( sasql_stmt $stmt )
```

参数

$stmt – 由 sasql_prepare 函数返回的预准备语句资源。

返回值
一个整数错误代码。

sasql_stmt_error

返回最近使用指定的语句资源执行的语句函数的错误文本。

原型

```
string sasql_stmt_error( sasql_stmt $stmt )
```

参数

$stmt – 由 sasql_prepare 函数返回的预准备语句资源。

返回值
描述错误的字符串。

sasql_stmt_execute

执行预准备语句。sasql_stmt_result_metadata 可用于检查语句是否返回结果集。

原型

```
bool sasql_stmt_execute( sasql_stmt $stmt )
```

参数

$stmt – 由 sasql_prepare 函数返回的预准备语句资源。应该在执行调用前绑定变量。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_fetch

此函数为语句读取结果之外的一行，并将列置于使用 sasql_stmt_bind_result 绑定的变量中。

原型

```
bool sasql_stmt_fetch( sasql_stmt $stmt )
```

参数

$stmt – 语句资源。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_field_count

此函数返回语句结果集中的列数。

原型

```
int sasql_stmt_field_count( sasql_stmt $stmt )
```

参数

$stmt – 语句资源。

返回值
语句结果中的列数。如果语句不返回结果，则它返回 0。

sasql_stmt_free_result

此函数释放语句的高速缓存结果集。

原型

```
bool sasql_stmt_free_result( sasql_stmt $stmt )
```

参数

$stmt – 使用 sasql_stmt_execute 执行的语句资源。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_insert_id

返回最后插入到 IDENTITY 列或 DEFAULT AUTOINCREMENT 列中的值，如果是最后插入到了不含 IDENTITY 或 DEFAULT AUTOINCREMENT 列的表中，则返回零。

原型

```
int sasql_stmt_insert_id( sasql_stmt $stmt )
```

参数

$stmt – 由 sasql_stmt_execute 执行的语句资源。

返回值
由前一个 INSERT 语句为 IDENTITY 列或 DEFAULT AUTOINCREMENT 列生成的 ID 或者零（如果最后的插入不影响 IDENTITY 或 DEFAULT AUTOINCREMENT 列）。如果 $stmt 无效，此函数可返回 FALSE (0)。

sasql_stmt_next_result

此函数可以从语句前进到下一结果。如果没有另一个结果集，则放弃当前已完成的结果，并删除相关联的结果集对象（由 sasql_stmt_result_metadata 返回）。

原型

```
bool sasql_stmt_next_result( sasql_stmt $stmt )
```

参数

$stmt – 语句资源。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_num_rows

返回结果集中的行数。只有在调用 sasql_stmt_store_result 函数以缓冲整个结果集之后，才能确定结果集的实际行数。如果尚未调用 sasql_stmt_store_result 函数，则返回 0。

原型

```
int sasql_stmt_num_rows( sasql_stmt $stmt )
```

参数

$stmt – 由 sasql_stmt_execute 执行的语句资源，且针对该资源调用 sasql_stmt_store_result。

返回值
结果中可用的行数或 0（失败时）。

sasql_stmt_param_count

返回提供的预准备语句资源中的参数数目。

原型

```
int sasql_stmt_param_count( sasql_stmt $stmt )
```

参数

$stmt – 由 sasql_prepare 函数返回的语句资源。

返回值
参数数目或 FALSE（错误时）。

sasql_stmt_reset

此函数将 *$stmt* 对象重置为描述之后的状态。对所有被绑定的变量解除绑定，并删除所有使用 sasql_stmt_send_long_data 发送的数据。

原型

```
bool sasql_stmt_reset( sasql_stmt $stmt )
```

参数

$stmt – 语句资源。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_result_metadata

为所提供的语句返回结果集对象。

原型

```
sasql_result sasql_stmt_result_metadata( sasql_stmt $stmt )
```

参数

$stmt – 准备和执行的语句资源。

返回值
sasql_result 对象或 FALSE（如果语句不返回任何结果）。

sasql_stmt_send_long_data

允许用户发送块中的参数数据。用户在尝试发送任何数据之前，必须先调用 sasql_stmt_bind_param 或 sasql_stmt_bind_param_ex。绑定参数必须属于字符串或 blob 类型。重复调用此函数将会附加至先前发送的内容。

原型

```
bool sasql_stmt_send_long_data( sasql_stmt $stmt, int
$param_number, string $data )
```

参数

$stmt – 使用 sasql_prepare 准备的语句资源。

$param_number – 参数编号。它必须是 0 和 (sasql_stmt_param_count($stmt) - 1) 之间的一个数字。

$data – 要发送的数据。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_stmt_store_result

此函数允许客户端高速缓存语句的整个结果集。可以使用函数 sasql_stmt_free_result 来释放高速缓存的结果。

原型

```
bool sasql_stmt_store_result( sasql_stmt $stmt )
```

参数

$stmt – 使用 sasql_stmt_execute 执行的语句资源。

返回值
TRUE（成功时）或 FALSE（失败时）。

sasql_store_result

传输在要与 sasql_data_seek 函数配合使用的数据库连接 *$conn* 上所进行的最后一次查询的结果集。

原型

```
sasql_result sasql_store_result( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
FALSE（如果查询未返回结果对象）或结果集对象（包含所有行的结果）。该结果在客户端被高速缓存。

sasql_sqlstate

返回最近的 SQLSTATE 字符串。SQLSTATE 指示最近执行的 SQL 语句是否产生了成功、错误或警告状态。由 5 个字符 "00000" 所构成的 SQLSTATE 代码表示没有错误。这些值由 ISO/ANSI SQL 标准定义。

原型

```
string sasql_sqlstate( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
返回一个包含当前 SQLSTATE 代码的由 5 个字符构成的字符串。结果 "00000" 表示没有错误。

sasql_use_result

为在连接上执行的最后一次查询启动结果集检索。

原型

```
sasql_result sasql_use_result( sasql_conn $conn )
```

参数

$conn – 由连接函数返回的连接资源。

返回值
FALSE（如果查询未返回结果对象或结果集对象）。该结果在客户端未被高速缓存。

Ruby 支持

SAP Sybase IQ 支持三种不同的 Ruby 应用程序编程接口。第一种是 SAP Sybase IQ Ruby API。此 API 通过 SAP Sybase IQ C API 公开的接口提供了一个 Ruby 包装。第二种是对 ActiveRecord 的支持，ActiveRecord 是一个对象相关映射程序，作为 Ruby on Rails Web 开发框架的一部分而为人所知。第三种是对 Ruby DBI 的支持。SAP Sybase IQ 提供了 Ruby 数据库驱动程序 (DBD)，可与 DBI 配合使用。

Ruby API 支持

SAP Sybase IQ Ruby 项目中提供三个单独的程序包。安装其中任何一个程序包最简单的方法是使用 `RubyGems`。

SAP Sybase IQ Ruby 项目的主页为 http://sqlanywhere.rubyforge.org/。

本地 Ruby 驱动程序

sqlanywhere – 此程序包是低层驱动程序，允许 Ruby 代码与 SAP Sybase IQ 数据库结合。此程序包通过 SAP Sybase IQ C API 公开的接口提供了一个 Ruby 包装。此程序包使用 C 语言编写，可以源代码或预编译 gem 的形式提供，适用于 Windows 和 Linux。如果已安装 `RubyGems`，可以通过运行以下命令获得此程序包：

```
gem install sqlanywhere
```

此程序包是任何其它 SQL Anywhere Ruby 程序包的前提条件。

Rails

Rails 是用 Ruby 语言编写的 Web 开发框架。其优势在于 Web 应用程序开发。在尝试 Rails 开发之前，强烈建议您熟悉 Ruby 编程语言。请参见 http://www.rubyonrails.org/。

ActiveRecord 适配器

activerecord-sqlanywhere-adapter – 此程序包是一个允许 ActiveRecord 与 SAP Sybase IQ 进行通信的适配器。ActiveRecord 是一个对象相关映射程序，作为 Ruby on Rails Web 开发框架的一部分而为人所知。此程序包使用纯 Ruby 编写，以源代码或 gem 格式提供。此适配器使用（并依赖于）`sqlanywhere` gem。如果已安装 `RubyGems`，可以通过运行以下命令安装此程序包及其依赖包：

```
gem install activerecord-sqlanywhere-adapter
```

SQL Anywhere Ruby/DBI 驱动程序

dbi – 此程序包是 Ruby 的 DBI 驱动程序。如果已安装 RubyGems，可以通过运行以下命令安装此程序包及其依赖包：

```
gem install dbi
```

dbd-sqlanywhere – 此程序包是一个允许 Ruby/DBI 与 SAP Sybase IQ 进行通信的驱动程序。Ruby/DBI 是模仿流行的 Perl DBI 模块的通用数据库接口。此程序包使用纯 Ruby 编写，以源代码或 gem 格式提供。此驱动程序使用（并依赖于）sqlanywhere gem。如果已安装 RubyGems，可以通过运行以下命令安装此程序包及其依赖包：

```
gem install dbd-sqlanywhere
```

有关其中任何一个程序包的反馈信息，请使用邮件列表 sqlanywhere-users@rubyforge.com。

在 SAP Sybase IQ 中配置 Rails 支持

可在 SAP Sybase IQ 中配置 Ruby on Rails 支持。

前提条件

执行此任务没有前提条件。

过程

1. 安装 RubyGems。它可以简化 Ruby 程序包的安装。Ruby on Rails 下载页面将引导您安装正确版本。请参见 http://www.rubyonrails.org/。
2. 在系统中安装 Ruby 解释器。Ruby on Rails 下载页面会推荐要安装的版本。请参见 http://www.rubyonrails.org/。
3. 运行以下命令安装 Ruby Rails 及其依赖包：

   ```
   gem install rails
   ```
4. 安装 Ruby 开发工具包 (`DevKit`)。从 http://rubyinstaller.org/downloads/ 下载 `DevKit` 并按照 http://github.com/oneclick/rubyinstaller/wiki/Development-Kit 中的操作说明进行操作。
5. 运行以下命令安装 SQL Anywhere ActiveRecord 支持 (activerecord-sqlanywhere-adapter)：

   ```
   gem install activerecord-sqlanywhere-adapter
   ```
6. 将 SAP Sybase IQ 添加到 Rails 支持的数据库管理系统集。编写本文档时，Rails 3.1.3 是当时发布的版本。
   a. 在 Rails `configs\databases` 目录中创建 `sqlanywhere.yml` 文件，以配置数据库。如果已将 Ruby 安装在 `\Ruby` 目录中，并且安装了 3.1.3 版本的 Rails，则此文件的路径为 `\Ruby\lib\ruby\gems\1.9.1\gems\railties-3.1.3\lib\rails\generators\rails\app\templates\config\databases`。此文件的内容应该为：

      ```
      #
      # SQL Anywhere database configuration
      #
      # This configuration file defines the patten used for
      # database filenames. If your application is called "blog",
      ```

```
# then the database names will be blog_development,
# blog_test, blog_production. The specified username and
# password should permit DBA access to the database.
#

development:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_development
  username: DBA
  password: sql

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run
"rake".
# Do not set this db to the same as development or production.
test:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_test
  username: DBA
  password: sql

production:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_production
  username: DBA
  password: sql
```

sqlanywhere.yml 文件提供了一个模板，用于在 Rails 项目中创建 database.yml 文件。可以指定以下数据库选项：

- **adapter** – （必选，无缺省值）。此选项必须设置为 sqlanywhere 才能使用 SQL Anywhere ActiveRecord 适配器。
- **database** – （必选，无缺省值）。此选项对应于连接字符串中的 DatabaseName。
- **server** – （可选，缺省为 database 选项）。此选项对应于连接字符串中的 ServerName。
- **username** – （可选，缺省为 'DBA'）。此选项对应于连接字符串中的 UserID。
- **password** – （可选，缺省为 'DBA'）。此选项对应于连接字符串中的 Password。
- **encoding** – （可选，缺省为 OS 字符集）。此选项对应于连接字符串中的 CharSet。
- **commlinks** – （可选）。此选项对应于连接字符串中的 CommLinks。
- **connection_name** – （可选）。此选项对应于连接字符串中的 ConnectionName。

b. 更新 Rails app_base.rb 文件。使用上一步骤中的相同假设条件，此文件位于 \Ruby\lib\ruby\gems\1.9.1\gems\railties-3.1.3\lib\rails

\generators\app_base.rb 路径下。编辑 app_base.rb 文件，并找到以下行：

```
DATABASES = %w( mysql oracle postgresql sqlite3 frontbase
ibm_db sqlserver )
```

将 sqlanywhere 添加到列表，如下所示：

```
DATABASES = %w( sqlanywhere mysql oracle postgresql sqlite3
frontbase ibm_db sqlserver )
```

7. 使用以下 SAP Sybase IQ 特定注释，按照 Ruby on Rails Web 站点 (http://guides.rails.info/getting_started.html) 上的教程进行操作：
   - 在此教程中，将介绍用于初始化 blog 项目的命令。使用以下命令可以初始化 blog 项目以用于 SAP Sybase IQ：

     ```
     rails new blog -d sqlanywhere
     ```
   - 创建 blog 应用程序后，切换到其文件夹以直接在该应用程序中继续工作：

     ```
     cd blog
     ```
   - 编辑 gemfile 文件以包括用于 SQL Anywhere ActiveRecord 适配器的 gem 指令。在下面指定的行后添加新指令：

     ```
     gem 'sqlanywhere'
     gem 'activerecord-sqlanywhere-adapter'
     ```
   - config\database.yml 文件引用了开发、测试和生产数据库。不按照教程的指示使用 rake 命令创建数据库，而是切换到项目的 db 目录并创建如下三个数据库。

     ```
     cd db
     iqinit -dba DBA,sql blog_development
     iqinit -dba DBA,sql blog_test
     iqinit -dba DBA,sql blog_production
     cd ..
     ```

已在 SAP Sybase IQ 中配置了 Rails 支持。

下一步

按如下所示启动数据库服务器和上述三个数据库。

```
iqsrv16 -n blog blog_development.db blog_production.db blog_test.db
```

命令行 (blog) 中的数据库服务器名称必须与 database.yml 文件中 server: 标签所指定的名称相匹配。sqlanywhere.yml 模板文件的配置可以保证数据库服务器名称与生成的所有 database.yml 文件中的项目名称相匹配。

Ruby-DBI 驱动程序

本节将概述如何编写使用 DBI 驱动程序的 Ruby 应用程序。

装载 DBI 模块

要在 Ruby 应用程序中使用 DBI:SQLAnywhere 接口，必须先告诉 Ruby 您打算使用 Ruby DBI 模块。为此，请在 Ruby 源代码文件的顶部附近包括以下行。

```
require 'dbi'
```

DBI 模块会根据需要自动装载 SQL Anywhere 数据库驱动程序 (DBD) 接口。

打开和关闭连接
通常，打开一个到数据库的连接，然后通过该连接运行一系列 SQL 语句来执行所有需要的操作。要打开连接，请使用 connect 函数。返回值是一个到数据库连接的句柄，使用该句柄可以在连接上执行后继操作。

对 connect 函数的调用采用一般形式：

```
dbh = DBI.connect('DBI:SQLAnywhere:server-name', user-id, password,
options)
```

- ***server-name*** – 是想要连接到的数据库服务器的名称。也可以按 "option1=value1;option2=value2;..." 的格式指定连接字符串。
- ***user-id*** – 是有效用户 ID。除非此字符串为空，否则将 ";UID=value" 附加到连接字符串。
- ***password*** – 是用户 ID 的相应口令。除非此字符串为空，否则将 ";PWD=value" 附加到连接字符串。
- ***options*** – 是一列附加连接参数，如 DatabaseName、DatabaseFile 和 ConnectionName。这些参数以 "option1=value1;option2=value2;..." 的格式附加到连接字符串。

要演示 connect 函数，需创建 iqdemo 数据库，并在运行示例 Ruby 脚本之前启动数据库服务器和 iqdemo 数据库。

```
$IQDIR16/demo
 mkiqdemo.sh
start_iq @iqdemo.cfg iqdemo.db
```

以下代码示例将打开 iqdemo 数据库的连接，然后将其关闭。下面示例中的字符串 "myserver" 是服务器名。

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:myserver', '<user_id>', '<password>')
do |dbh|
    if dbh.ping
       print "Successfully Connected\n"
       dbh.disconnect()
    end
end
```

也可以指定一个连接字符串替换服务器名。例如，在上面的脚本中，可以通过替换 connect 函数的第一个参数进行更改，如下所示：

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:SERVER=myserver;DBN=iqdemo',
'<user_id>', '<password>') do |dbh|
    if dbh.ping
       print "Successfully Connected\n"
       dbh.disconnect()
    end
end
```

用户 ID 和口令无法在连接字符串中指定。如果忽略这些参数，Ruby DBI 会自动用缺省值填写用户名和口令，因此绝对不要在连接字符串中包含 UID 或 PWD 连接参数。否则将抛出异常。

以下示例说明如何将附加连接参数作为散列键/值对传递到 connect 函数。

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:myserver', '<user_id>', '<password>',
       { :ConnectionName => "RubyDemo",
         :DatabaseFile => "iqdemo.db",
         :DatabaseName => "iqdemo" }
        ) do |dbh|
    if dbh.ping
       print "Successfully Connected\n"
       dbh.disconnect()
    end
end
```

选择数据

获得了打开的连接的句柄后，您可以访问和修改存储在数据库中的数据。可能最简单的操作是检索某些行并输出它们。

必须先执行 SQL 语句。如果语句返回结果集，可以使用得到的语句句柄检索有关结果集和结果集行的元信息。以下示例从元数据获得列名，并显示获取的每一行的列名和值。

```
require 'dbi'

def db_query( dbh, sql )
    sth = dbh.execute(sql)
    print "# of Fields:  #{sth.column_names.size}\n"
    sth.fetch do |row|
        print "\n"
        sth.column_info.each_with_index do |info, i|
            unless info["type_name"] == "LONG VARBINARY"
                print "#{info["name"]}=#{row[i]}\n"
            end
        end
    end
    sth.finish
end

begin
    dbh = DBI.connect('DBI:SQLAnywhere:demo', '<user_id>',
'<password>')
    db_query(dbh, "SELECT * FROM Products")
rescue DBI::DatabaseError => e
    puts "An error occurred"
    puts "Error code: #{e.err}"
    puts "Error message: #{e.errstr}"
    puts "Error SQLSTATE: #{e.state}"
ensure
    dbh.disconnect if dbh
end
```

显示的输出的前几行如下所示。

```
# of Fields:  8

ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28
UnitPrice=9.00

ID=301
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00
```

完成后调用 finish 释放语句句柄十分重要。如果不释放句柄，则可能收到如下所示的错误：

```
Resource governor for 'prepared statements' exceeded
```

为检测句柄泄漏，缺省情况下 SAP Sybase IQ 数据库服务器将允许的游标和准备好的语句数量限制为每个连接最多 50 个。如果超过这些限制，资源调控器自动生成错误。如果收到此错误，请检查未被释放的语句句柄。请谨慎使用 prepare_cached，因为语句句柄并没有被释放。

如有必要，可通过设置 max_cursor_count 和 max_statement_count 选项来修改这些限制。

插入行

插入行需要打开的连接的句柄。插入行最简单的方法是使用参数化的 INSERT 语句，这意味着问号用作值的占位符。首先准备好语句，然后对每一新行执行一次。新行的值作为参数提供给 execute 方法。

```
require 'dbi'

def db_query( dbh, sql )
    sth = dbh.execute(sql)
    print "# of Fields:  #{sth.column_names.size}\n"
    sth.fetch do |row|
        print "\n"
        sth.column_info.each_with_index do |info, i|
            unless info["type_name"] == "LONG VARBINARY"
                print "#{info["name"]}=#{row[i]}\n"
            end
        end
    end
    sth.finish
end

def db_insert( dbh, rows )
```

```
    sql = "INSERT INTO Customers (ID, GivenName, Surname,
                Street, City, State, Country, PostalCode,
                Phone, CompanyName)
           VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
    sth = dbh.prepare(sql);
    rows.each do |row|
        sth.execute(row[0],row[1],row[2],row[3],row[4],
                    row[5],row[6],row[7],row[8],row[9])
    end
end

begin
    dbh = DBI.connect('DBI:SQLAnywhere:demo', '<user_id>',
'<password>')
    rows = [
        [801,'Alex','Alt','5 Blue Ave','New York','NY','USA',
            '10012','5185553434','BXM'],
        [802,'Zach','Zed','82 Fair St','New York','NY','USA',
            '10033','5185552234','Zap']
       ]
    db_insert(dbh, rows)
    dbh.commit
    db_query(dbh, "SELECT * FROM Customers WHERE ID > 800")
rescue DBI::DatabaseError => e
    puts "An error occurred"
    puts "Error code: #{e.err}"
    puts "Error message: #{e.errstr}"
    puts "Error SQLSTATE: #{e.state}"
ensure
    dbh.disconnect if dbh
end
```

SAP Sybase IQ Ruby API 参考

SAP Sybase IQ 提供了 SAP Sybase IQ C API 的低层接口。以下几节中介绍的 API 允许快速开发 SQL 应用程序。要演示 Ruby 应用程序开发能力，可考虑以下 Ruby 程序示例。该程序装载 SAP Sybase IQ Ruby 扩展，连接到示例数据库，列出 Products 表中的列值，断开连接并终止。

```
begin
    require 'rubygems'
    gem 'sqlanywhere'
    unless defined? SQLAnywhere
        require 'sqlanywhere'
    end
end
api = SQLAnywhere::SQLAnywhereInterface.new()
SQLAnywhere::API.sqlany_initialize_interface( api )
api.sqlany_init()
conn = api.sqlany_new_connection()
api.sqlany_connect( conn, "DSN=Sybase IQ Demo" )
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Products" )
num_rows = api.sqlany_num_rows( stmt )
```

```
num_rows.times {
    api.sqlany_fetch_next( stmt )
    num_cols = api.sqlany_num_cols( stmt )
    for col in 1..num_cols do
        info = api.sqlany_get_column_info( stmt, col - 1 )
        unless info[3]==1 # Don't do binary
            rc, value = api.sqlany_get_column( stmt, col - 1 )
            print "#{info[2]}=#{value}\n"
        end
    end
    print "\n"
}
api.sqlany_free_stmt( stmt )
api.sqlany_disconnect(conn)
api.sqlany_free_connection(conn)
api.sqlany_fini()
SQLAnywhere::API.sqlany_finalize_interface( api )
```

此 Ruby 程序输出的结果集的前两行如下所示：

```
ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28
UnitPrice=9.00

ID=301
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00
```

以下几节介绍每个支持的函数。

sqlany_affected_rows

返回受执行预准备语句影响的行数。

语法

```
sqlany_affected_rows ( $stmt )
```

参数

- **$stmt** – 预准备并成功执行的语句，其中不返回任何结果集。例如，执行了 INSERT、UPDATE 或 DELETE 语句。

返回值

返回标量值（受影响的行数）或返回 -1（失败时）。

示例

```
affected = api.sqlany_affected( stmt )
```

sqlany_bind_param 函数

将用户提供的缓冲区作为参数绑定到预准备语句。

语法

```
sqlany_bind_param ( $stmt, $index, $param )
```

参数

- **$stmt** – 成功执行 sqlany_prepare 所返回的语句对象。
- **$index** – 参数的索引。该数字必须在 0 到 sqlany_num_params() - 1 之间。
- **$param** – 检索自 sqlany_describe_bind_param 的填充的绑定对象。

返回值
返回标量值，成功时为 1，不成功时为 0。

示例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

sqlany_clear_error 函数

清除上次存储的错误代码。

语法

```
sqlany_clear_error ( $conn )
```

参数

- **$conn** – 从 sqlany_new_connection 返回的连接对象。

返回值
返回 nil。

示例

```
api.sqlany_clear_error( conn )
```

sqlany_client_version 函数

返回当前的客户端版本。

语法

```
sqlany_client_version ( )
```

返回值
返回标量值，即客户端版本字符串。

示例

```
buffer = api.sqlany_client_version()
```

sqlany_commit 函数

提交当前事务。

语法

```
sqlany_commit ( $conn )
```

参数

- **$conn** – 要执行提交操作的连接对象。

返回值
返回标量值，成功时为 1，不成功时为 0。

示例

```
rc = api.sqlany_commit( conn )
```

sqlany_connect 函数

使用指定的连接对象和连接字符串创建与某个 SQL Anywhere 数据库服务器的连接。

语法

```
sqlany_connect ( $conn, $str )
```

参数

- **$conn** – 由 sqlany_new_connection 创建的连接对象。
- **$str** – 一个 SQL Anywhere 连接字符串。

返回值
返回标量值，成功建立连接为 1，连接失败则为 0。使用 sqlany_error 检索错误代码和消息。

示例

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Connection status = #{status}\n"
```

sqlany_describe_bind_param 函数

描述预准备语句的绑定参数。

语法

```
sqlany_describe_bind_param ( $stmt, $index )
```

参数

- **$stmt** – 使用 sqlany_prepare 成功预准备的语句。
- **$index** – 参数的索引。该数字必须在 0 到 sqlany_num_params() - 1 之间。

返回值

返回 2 个元素的数组，其中第一个参数中 1 表示成功，0 表示失败；第二个是描述性参数。

注释

此函数允许调用者确定有关预准备语句参数的信息。预准备语句（存储过程或 DML）的类型确定所提供的信息量。始终会提供参数的方向（输入、输出或输入-输出）。

示例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "\n"
print "Param dir = ", param.get_direction(), "\n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

sqlany_disconnect 函数

断开 SQL Anywhere 连接。回退所有未提交的事务。

语法

```
sqlany_disconnect ( $conn )
```

参数

- **$conn** – 已使用 sqlany_connect 建立连接的连接对象。

返回值
返回标量值，成功时为 1，不成功时为 0。

示例

```
# Disconnect from the database
status = api.sqlany_disconnect( conn )
print "Disconnect status = #{status}\n"
```

sqlany_error 函数

返回存储在连接对象中的上一错误代码和消息。

语法

```
sqlany_error ( $conn )
```

参数

- **$conn** – 从 sqlany_new_connection 返回的连接对象。

返回值
返回 2 个元素的数组，其中第一个参数表示 SQL 错误代码，第二个参数表示错误消息字符串。

对于错误代码，正值表示警告，负值表示错误，0 表示成功执行。

示例

```
code, msg = api.sqlany_error( conn )
print "Code=#{code} Message=#{msg}\n"
```

sqlany_execute 函数

执行预准备语句。

语法

```
sqlany_execute ( $stmt )
```

参数

- **$stmt** – 使用 sqlany_prepare 成功预准备的语句。

返回值
返回标量值，成功时为 1，不成功时为 0。

注释
可使用 sqlany_num_cols 验证语句是否返回结果集。

示例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

sqlany_execute_direct 函数

执行由字符串参数指定的 SQL 语句。

语法

```
sqlany_execute_direct ( $conn, $sql )
```

参数

- **$conn** – 已使用 sqlany_connect 建立连接的连接对象。
- **$sql** – SQL 字符串。SQL 字符串不应包含诸如 ? 之类的参数。

返回值
返回语句对象或 nil（失败时）。

注释
使用此函数只需一步便可准备和执行语句。不要使用此函数执行带参数的 SQL 语句。

示例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_execute_immediate 函数

立即执行指定的 SQL 语句，不返回结果集。这对不返回结果集的语句很有用。

语法

```
sqlany_execute_immediate ( $conn, $sql )
```

参数

- **$conn** – 已使用 sqlany_connect 建立连接的连接对象。

- **$sql** – SQL 字符串。SQL 字符串不应包含诸如 ? 之类的参数。

返回值
返回标量值，成功时为 1，不成功时为 0。

示例

```
rc = api.sqlany_execute_immediate(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= 50" )
```

sqlany_fetch_absolute 函数

将结果集中的当前行移动到指定的行编号处，然后读取该行的数据。

语法

```
sqlany_fetch_absolute ( $stmt, $row_num )
```

参数

- **$stmt** – 由 sqlany_execute 或 sqlany_execute_direct 执行的语句对象。
- **$row_num** – 要读取的行编号。第一行为 1，最后一行为 -1。

返回值
返回标量值，成功时为 1，不成功时为 0。

示例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_absolute( stmt, 2 )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_fetch_next 函数

返回结果集的下一行。此函数首先前移行指针，然后读取新行的数据。

语法

```
sqlany_fetch_next ( $stmt )
```

参数

- **$stmt** – 由 sqlany_execute 或 sqlany_execute_direct 执行的语句对象。

返回值
返回标量值，成功时为 1，不成功时为 0。

示例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_fini 函数

释放由 API 分配的资源。

语法

```
sqlany_fini ( )
```

返回值
返回 nil。

示例

```
# Disconnect from the database
api.sqlany_disconnect( conn )

# Free the connection resources
api.sqlany_free_connection( conn )

# Free resources the api object uses
api.sqlany_fini()

# Close the interface
SQLAnywhere::API.sqlany_finalize_interface( api )
```

sqlany_free_connection 函数

释放与连接对象关联的资源。

语法

```
sqlany_free_connection ( $conn )
```

参数

- **$conn** – 由 sqlany_new_connection 创建的连接对象。

返回值
返回 nil。

示例

```
# Disconnect from the database
api.sqlany_disconnect( conn )

# Free the connection resources
api.sqlany_free_connection( conn )

# Free resources the api object uses
api.sqlany_fini()

# Close the interface
SQLAnywhere::API.sqlany_finalize_interface( api )
```

sqlany_free_stmt 函数

释放与语句对象关联的资源。

语法

```
sqlany_free_stmt ( $stmt )
```

参数

- **$stmt** – 成功执行 sqlany_prepare 或 sqlany_execute_direct 所返回的语句对象。

返回值
返回 nil。

示例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
rc = api.sqlany_free_stmt( stmt )
```

sqlany_get_bind_param_info 函数

检索使用 sqlany_bind_param 绑定的参数的相关信息。

语法

```
sqlany_get_bind_param_info ( $stmt, $index )
```

参数

- **$stmt** – 使用 sqlany_prepare 成功预准备的语句。
- **$index** – 参数的索引。该数字必须在 0 到 sqlany_num_params() - 1 之间。

返回值
返回 2 个元素的数组，其中第一个参数中 1 表示成功，0 表示失败；第二个是描述性参数。

示例

```
# Get information on first parameter (0)
rc, param_info = api.sqlany_get_bind_param_info( stmt, 0 )
print "Param_info direction = ", param_info.get_direction(), "\n"
print "Param_info output = ", param_info.get_output(), "\n"
```

sqlany_get_column 函数

返回为指定列读取的值。

语法

```
sqlany_get_column ( $stmt, $col_index )
```

参数

- **$stmt** – 由 sqlany_execute 或 sqlany_execute_direct 执行的语句对象。
- **$col_index** – 要检索的列编号。列编号在 0 到 sqlany_num_cols() - 1 之间。

返回值
返回 2 个元素的数组，其中第一个参数中 1 表示成功，0 表示失败；第二个是列值。

示例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "\n"
```

sqlany_get_column_info 函数

获取指定结果集列的列信息。

语法

```
sqlany_get_column_info ( $stmt, $col_index )
```

参数

- **$stmt** – 由 sqlany_execute 或 sqlany_execute_direct 执行的语句对象。
- **$col_index** – 列编号在 0 到 sqlany_num_cols() - 1 之间。

返回值
返回 9 个元素的数组，其中包含用于描述结果集中一列的信息。第一个元素包含 1（成功时）或 0（失败时）。数组元素在下表中进行介绍。

元素号	类型	说明
0	整数	成功则为 1，失败则为 0。
1	整数	列索引（0 到 sqlany_num_cols() - 1）。
2	字符串	列名。
3	整数	列类型。
4	整数	列本地类型。
5	整数	列精度（用于数字类型）。
6	整数	列小数位数（用于数字类型）。
7	整数	列大小。
8	整数	列可为空（1 = 可为空，0 = 不可为空）。

示例

```
# Get column info for first column (0)
rc, col_num, col_name, col_type, col_native_type, col_precision,
col_scale,
    col_size, col_nullable = api.sqlany_get_column_info( stmt, 0 )
```

sqlany_get_next_result 函数

前进到多结果集查询中的下一结果集。

语法

```
sqlany_get_next_result ( $stmt )
```

参数

- **$stmt** – 由 sqlany_execute 或 sqlany_execute_direct 执行的语句对象。

返回值
返回标量值，成功时为 1，不成功时为 0。

示例

```
stmt = api.sqlany_prepare(conn, "call two_results()" )
rc = api.sqlany_execute( stmt )
# Fetch from first result set
rc = api.sqlany_fetch_absolute( stmt, 3 )
# Go to next result set
rc = api.sqlany_get_next_result( stmt )
# Fetch from second result set
rc = api.sqlany_fetch_absolute( stmt, 2 )
```

sqlany_init 函数

初始化接口。

语法

```
sqlany_init ( )
```

返回值

返回 2 个元素的数组，其中第一个参数中 1 表示成功，0 表示失败，第二个是 Ruby 接口版本。

示例

```
# Load the SQLAnywhere gem
begin
    require 'rubygems'
    gem 'sqlanywhere'
    unless defined? SQLAnywhere
        require 'sqlanywhere'
    end
end
# Create an interface
api = SQLAnywhere::SQLAnywhereInterface.new()
# Initialize the interface (loads the DLL/SO)
SQLAnywhere::API.sqlany_initialize_interface( api )
# Initialize our api object
api.sqlany_init()
```

sqlany_new_connection 函数

创建连接对象。

语法

```
sqlany_new_connection ()
```

返回值

返回标量值，即连接对象。

注释

必须先创建连接对象，然后才能建立数据库连接。可以从连接对象中检索错误。一个连接中每次只能处理一个请求。

示例

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Status=#{status}\n"
```

sqlany_num_cols 函数

返回结果集中的列数。

语法

```
sqlany_num_cols ( $stmt )
```

参数

- **$stmt** – 由 sqlany_execute 或 sqlany_execute_direct 执行的语句对象。

返回值
返回标量值，即结果集中的列数或 -1（失败时）。

示例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of result set columns
num_cols = api.sqlany_num_cols( stmt )
```

sqlany_num_params 函数

返回预期用于预准备语句的参数数目。

语法

```
sqlany_num_params ( $stmt )
```

参数

- **$stmt** – 成功执行 sqlany_prepare 所返回的语句对象。

返回值
返回标量值，即预准备语句中的参数数或 -1（失败时）。

示例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
      SET Contacts.ID = Contacts.ID + 1000
      WHERE Contacts.ID >= ?" )
num_params = api.sqlany_num_params( stmt )
```

sqlany_num_rows 函数

返回结果集中的行数。

语法

```
sqlany_num_rows ( $stmt )
```

参数

- **$stmt** – 由 sqlany_execute 或 sqlany_execute_direct 执行的语句对象。

返回值
返回标量值，即结果集中的行数。如果行数是估计值，则返回负数，且估计值为所返回整数的绝对值。如果行数为准确值，则返回值为正。

注释
缺省情况下，此函数仅返回一个估计值。要返回准确计数，请在连接中设置 ROW_COUNTS 选项。

只能针对返回多个结果集的语句中的第一个结果集返回结果集中的行数计数。如果使用 sqlany_get_next_result 移动到下一个结果集，sqlany_num_rows 将仍返回第一个结果集中的行数。

示例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of rows in result set
num_rows = api.sqlany_num_rows( stmt )
```

sqlany_prepare 函数

准备所提供的 SQL 字符串。

语法

```
sqlany_prepare ( $conn, $sql )
```

参数

- **$conn** – 已使用 sqlany_connect 建立连接的连接对象。
- **$sql** – 要准备的 SQL 语句。

返回值
返回标量值，即语句对象或 nil（失败时）。

注释
由 sqlany_execute 执行与该语句对象相关联的语句。可使用 sqlany_free_stmt 释放与语句对象关联的资源。

示例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

sqlany_rollback 函数

回退当前事务。

语法

```
sqlany_rollback ( $conn )
```

参数

- **$conn** – 要执行回退操作的连接对象。

返回值

返回标量值，成功时为 1，不成功时为 0。

示例

```
rc = api.sqlany_rollback( conn )
```

sqlany_sqlstate 函数

检索当前的 SQLSTATE。

语法

```
sqlany_sqlstate ( $conn )
```

参数

- **$conn** – 从 sqlany_new_connection 返回的连接对象。

返回值

返回标量值，即当前 5 个字符的 SQLSTATE。

示例

```
sql_state = api.sqlany_sqlstate( conn )
```

列类型

下面的 Ruby 类定义了由一些 SQL Anywhere Ruby 函数返回的列类型。

```
class Types
    A_INVALID_TYPE = 0
    A_BINARY       = 1
```

```
    A_STRING        = 2
    A_DOUBLE        = 3
    A_VAL64         = 4
    A_UVAL64        = 5
    A_VAL32         = 6
    A_UVAL32        = 7
    A_VAL16         = 8
    A_UVAL16        = 9
    A_VAL8          = 10
    A_UVAL8         = 11
end
```

本地列类型

下表定义了由一些 SQL Anywhere 函数返回的本地列类型。

本地类型值	本地类型
384	DT_DATE
388	DT_TIME
390	DT_TIMESTAMP_STRUCT
392	DT_TIMESTAMP
448	DT_VARCHAR
452	DT_FIXCHAR
456	DT_LONGVARCHAR
460	DT_STRING
480	DT_DOUBLE
482	DT_FLOAT
484	DT_DECIMAL
496	DT_INT
500	DT_SMALLINT
524	DT_BINARY
528	DT_LONGBINARY
600	DT_VARIABLE
604	DT_TINYINT
608	DT_BIGINT
612	DT_UNSINT

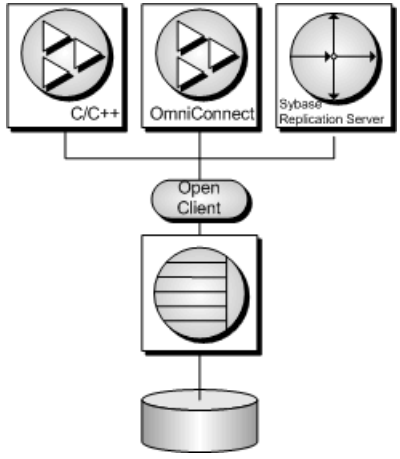
本地类型值	本地类型
616	DT_UNSSMALLINT
620	DT_UNSBIGINT
624	DT_BIT
628	DT_NSTRING
632	DT_NFIXCHAR
636	DT_NVARCHAR
640	DT_LONGNVARCHAR

Sybase Open Client 支持

Sybase Open Client 为客户应用程序、第三方产品及其它 Sybase 产品提供了与 SAP Sybase IQ 及其它 Open Server 进行通信所需要的接口。

何时使用 Open Client
如果您需要考虑与 Adaptive Server Enterprise 兼容或要使用其它支持 Open Client 的 Sybase 产品，那么您应考虑使用 Open Client 接口。

Open Client 应用程序
您可以用 C 或 C++ 开发应用程序，然后使用 Open Client API 将这些应用程序连接到 SAP Sybase IQ。其它 Sybase 应用程序（例如 OmniConnect）使用 Open Client。Open Client API 也是 Sybase Adaptive Server Enterprise 支持的接口。



Open Client 体系结构

本节介绍面向 SAP Sybase IQ 的 Open Client 编程接口。Sybase Open Client 应用程序开发的主要文档为 Sybase Open Client 文档，您可从 SAP 获得。本节介绍的是专门面向 SAP Sybase IQ 的功能，而不是 Sybase Open Client 应用程序编程的详尽指南。

Sybase Open Client 具有两种组件：编程接口和网络服务。

DB-Library 和 Client Library。
Sybase Open Client 提供了两个核心编程接口以供编写客户端应用程序时使用：DB-Library 和 Client-Library。

Open Client DB-Library 为早期版本的 Open Client 应用程序提供支持，是与 Client-Library 完全分开的编程接口。Sybase Open Client 产品附带的 Open Client DB-Library/C 参考手册对 DB-Library 进行了介绍。

Client-Library 程序还依赖于 CS-Library，在 Client-Library 和 Server-Library 应用程序中都使用 CS-Library 提供的例程。Client-Library 应用程序还可以使用 Bulk-Library 中的例程来促进数据的高速传输。

CS-Library 和 Bulk-Library 都包含在 Sybase Open Client 中，两者分开使用。

网络服务

Open Client 网络服务包括 Sybase Net-Library，Sybase Net-Library 为特定网络协议（如 TCP/IP 和 DECnet）提供支持。Net-Library 接口对于应用程序开发人员来说是不可见的。但在某些平台上，针对不同的系统网络配置，应用程序可能需要另外一种 Net-Library 驱动程序。根据您的主机平台，Net-Library 驱动程序由系统的 Sybase 配置指定，或者在您编译和链接程序时指定。

有关驱动程序配置的说明，请参见 Open Client/Server 配置指南。

有关构建 Client-Library 程序的说明，请参见 Open Client/Server 程序员补充材料。

建立 Open Client 应用程序的要求

要运行 Open Client 应用程序，则必须在运行该应用程序的计算机上安装和配置 Sybase Open Client 组件。这些组件可以作为您安装的其它 Sybase 产品的一部分进行安装，您也可以随 SAP Sybase IQ 一同安装这些库，这取决于许可协议条款的规定。

Open Client 应用程序不需要在运行数据库服务器的计算机上有任何 Open Client 组件。

要构建 Open Client 应用程序，您需要 Open Client 的开发版本，这可由 SAP 提供。

缺省情况下，SAP Sybase IQ 数据库创建后不区分大小写，而 Adaptive Server 数据库区分大小写。

Open Client 数据类型映射

Sybase Open Client 有其自己的内部数据类型，它们在某些细节上与 SAP Sybase IQ 中提供的数据类型有所区别。出于这个原因，SAP Sybase IQ 会在 Open Client 应用程序使用的数据类型和 SAP Sybase IQ 中的数据类型之间对某些数据类型进行内部映射。

要构建 Open Client 应用程序，您需要 Open Client 的开发版本。要使用 Open Client 应用程序，则必须在运行它的计算机上安装和配置 Open Client 运行时版本。

SAP Sybase IQ 服务器不需要任何外部通信运行时环境即可支持 Open Client 应用程序。

每个 Open Client 数据类型都映射到 SAP Sybase IQ 中对等的数据类型。所有的 Open Client 数据类型均受支持。

Open Client 中没有直接对等项的 SAP Sybase IQ 数据类型
下表列出了在 SAP Sybase IQ 中受支持但在 Open Client 中没有直接对等项的数据类型的映射关系。

SAP Sybase IQ 数据类型	Open Client 数据类型
unsigned short	int
unsigned int	bigint
unsigned bigint	numeric(20,0)
string	varchar
timestamp	datetime

Open Client 数据类型映射中的范围限制

某些数据类型在 SAP Sybase IQ 中与在 Open Client 中的范围不同。这种情况下，在检索或插入数据过程中可能会发生溢出错误。

下表列出了可映射到 SAP Sybase IQ 数据类型但对可能值范围具有某些限制的 Open Client 应用程序数据类型。

Open Client 数据类型通常会映射到可能值范围更大的 SAP Sybase IQ 数据类型。因此，可能会出现这种情况：向 SAP Sybase IQ 传递的值被接受并存储在数据库中，而这个值由于过大而无法被 Open Client 应用程序读取。

数据类型	Open Client 下限	Open Client 上限	SAP Sybase IQ 下限	SAP Sybase IQ 上限
MONEY	-922 377 203 685 477.5808	922 377 203 685 477.5807	-999 999 999 999 999.9999	999 999 999 999 999.9999
SMALLMONEY	-214 748.3648	214 748.3647	-999 999.9999	-999 999.9999
DATETIME [1]	January 1, 1753	December 31, 9999	January 1, 0001	December 31, 9999
SMALLDATETIME	January 1, 1900	June 6, 2079	January 1, 0001	December 31, 9999

[1] 对于早于 OpenClient 15.5 的版本而言；否则支持从 0001-01-01 到 9999-12-31 的完整范围。

示例

例如，Open Client MONEY 和 SMALLMONEY 数据类型不会跨越它们底层的 SAP Sybase IQ 实现的整个数值范围。因此，SAP Sybase IQ 列中的值有可能会超出 Open

Client 数据类型 MONEY 的界限。当客户端通过 SAP Sybase IQ 读取这种违规值时，就会产生错误。

时间戳
如果客户端使用的是 Open Client 15.1 或更早的版本，则插入到或获取自 SAP Sybase IQ 的时间戳值的日期部分将被限制在 1753 年 1 月 1 日或之后，且时间版本限制在 1/300 秒的精度。然而，如果客户端使用的是 Open Client 15.5 或更新的版本，则不会对时间戳的值作出任何限制。

Open Client 应用程序中的 SQL

本节简要介绍了在 Open Client 应用程序中使用 SQL 的相关信息，特别重点说明了专门针对 SAP Sybase IQ 的问题。

有关完整说明，请参见 http://www.sybase.com/products/databasemanagement/openserver 上的 Open Client 文档。

Open Client SQL 语句执行

可通过将 SQL 语句放到 Client Library 函数调用中将其发送到数据库服务器。例如，下面的两个调用将执行 DELETE 语句：

```
ret = ct_command(cmd, CS_LANG_CMD,
                  "DELETE FROM Employees
                   WHERE EmployeeID=105"
                 CS_NULLTERM,
                 CS_UNUSED);
ret = ct_send(cmd);
```

Open Client 预准备语句

ct_dynamic 函数用于管理预准备语句。此函数采用 *type* 参数描述您要执行的操作。

要在 Open Client 中使用预准备语句，可执行以下任务：

1. 使用带 CS_PREPARE *type* 参数的 ct_dynamic 函数准备语句。
2. 使用 ct_param 函数设置语句参数。
3. 使用带 CS_EXECUTE *type* 参数的 ct_dynamic 执行语句。
4. 使用带 CS_DEALLOC *type* 参数的 ct_dynamic 释放与该语句关联的资源。

有关在 Open Client 中使用预准备语句的详细信息，请参见您的 Open Client 文档。

Open Client 游标管理

ct_cursor 函数用于管理游标。此函数采用 *type* 参数描述您要执行的操作。

支持的游标类型
并非 SAP Sybase IQ 支持的所有游标类型都能通过 Open Client 接口使用。您不能通过 Open Client 使用滚动游标、动态滚动游标或不敏感游标。

唯一性和可更新性是游标的两个属性。游标可以唯一（每个行都带有主键或唯一性信息，无论应用程序是否使用它），也可以不唯一。游标可以是只读游标，也可以是可更新游标。如果游标是可更新游标且不唯一，则性能可能会受到影响，因为在这种情况下，无论 CS_CURSOR_ROWS 设置是什么，都不会对任何行执行预取。

使用游标的步骤

与其它某些接口不同（例如嵌入式 SQL），Open Client 会将游标与一个以字符串表示的 SQL 语句相关联。嵌入式 SQL 首先准备一个语句，然后游标使用该语句句柄进行声明。

要在 Open Client 中使用游标，可执行以下任务：

1. 要在 Open Client 中声明游标，请使用将 CS_CURSOR_DECLARE 作为 *type* 参数的 ct_cursor。
2. 声明游标之后，可以使用将 CS_CURSOR_ROWS 作为 *type* 参数的 ct_cursor 函数来控制每次从服务器读取一行时要预取多少行到客户端。

   在客户端存储预取的行会减少对服务器的调用，从而提高总体吞吐量，同时缩短周转时间。预取的行不会立即传递到应用程序，而是会存储在客户端的缓冲区内备用。

   PREFETCH 数据库选项的设置控制其它接口的行预取。Open Client 连接会忽略该设置。对于非唯一的可更新游标，将忽略 CS_CURSOR_ROWS 设置。
3. 要在 Open Client 中打开游标，请使用将 CS_CURSOR_OPEN 作为 *type* 参数的 ct_cursor 函数。
4. 使用 ct_fetch 函数可将每行读取到应用程序中。
5. 要关闭游标，请使用带 CS_CURSOR_CLOSE 的 ct_cursor。
6. 在 Open Client 中，您还需要释放与游标相关联的资源。使用带 CS_CURSOR_DEALLOC 的 ct_cursor 函数可以完成此任务。您还可以使用带附加参数 CS_DEALLOC 的 CS_CURSOR_CLOSE 通过单个步骤执行这些操作。

通过游标进行 Open Client 行修改

借助 Open Client，只要游标针对的是单个表，您就可以在游标中删除或更新行。用户必须具有更新表的权限，且游标必须标记为可更新。

您可以不执行读取，而是分别使用带 CS_CURSOR_DELETE 或 CS_CURSOR_UPDATE 的 ct_cursor 来删除或更新游标的当前行。

在 Open Client 应用程序中不能通过游标插入行。

Open Client 结果集

Open Client 处理结果集的方式与其它某些 SAP Sybase IQ 接口不同。

在嵌入式 SQL 和 ODBC 中，*描述*查询或存储过程是为了设置正确数目和类型的变量来接收结果。描述是对语句其本身执行。

在 Open Client 中，您不需要描述语句。从服务器返回的每个行都可以带有对其内容的描述。如果使用 ct_command 函数和 ct_send 函数执行语句，则可以使用 ct_results 函数来处理查询中所返回行的各个方面。

如果不想用这种逐行方式来处理结果集，则可以使用 ct_dynamic 函数来准备 SQL 语句，并使用 ct_describe 函数来描述其结果集。这种方法更接近于其它接口中对 SQL 语句的描述方式。

SAP Sybase IQ 的已知 Open Client 限制

通过 Open Client 接口，您基本上就可以像使用 Adaptive Server Enterprise 数据库那样使用 SAP Sybase IQ 数据库。但还是有一些限制，其中包括以下方面：

- SAP Sybase IQ 不支持 Adaptive Server Enterprise 提交服务。
- 客户端/服务器连接的*功能*决定了该连接所允许的客户端请求和服务器响应的类型。不支持以下功能：

  CS_CSR_ABS
  CS_CSR_FIRST
  CS_CSR_LAST
  CS_CSR_PREV
  CS_CSR_REL
  CS_DATA_BOUNDARY
  CS_DATA_SENSITIVITY
  CS_OPT_FORMATONLY
  CS_PROTO_DYNPROC
  CS_REG_NOTIF
  CS_REQ_BCP
- 不支持安全性组件，例如 SSL。但是，支持口令加密。
- Open Client 应用程序可使用 TCP/IP 连接到 SAP Sybase IQ。
  有关功能的详细信息，请参见 Open Server Server-Library C 参考手册。
- 如果将 CS_DATAFMT 与 CS_DESCRIBE_INPUT 一起使用，在某个参数化变量作为输入发送到 SAP Sybase IQ 时，CS_DATAFMT 不会返回列的数据类型。

HTTP Web 服务

开发使用 SAP Sybase IQ 作为 HTTP web 服务器的 web 服务应用程序。

SAP Sybase IQ 作为 HTTP Web 服务器

SAP Sybase IQ 包含内置的 HTTP web 服务器，可用于在 SAP Sybase IQ 数据库中创建在线 web 服务。SAP Sybase IQ web 服务器支持 web 浏览器和客户端应用程序发送的 HTTP 和 SOAP over HTTP 请求。Web 服务是嵌入到数据库中的，因此 Web 服务器的性能得到了优化。

SAP Sybase IQ Web 服务为客户端应用程序提供替代接口，来替代诸如 JDBC 和 ODBC 之类的传统接口。由于不需要额外的组件，它们很容易部署，还能通过多个平台上由多种语言（包括脚本编写语言如 Perl 和 Python）编写的客户端应用程序进行访问。

除了通过 HTTP Web 服务器提供 Web 服务，SAP Sybase IQ 还可以作为 SOAP 或 HTTP 客户端应用程序访问 Internet 和其它 SAP Sybase IQ HTTP Web 服务器提供的标准 Web 服务。

使用 SAP Sybase IQ 作为 HTTP Web 服务器的快速入门

本节介绍了如何启动 SAP Sybase IQ HTTP Web 服务器、如何创建 Web 服务以及如何使用 Web 浏览器访问它。但没有对 SAP Sybase IQ Web 服务功能（如 SOAP over HTTP 支持和应用程序开发）进行全面介绍。虽然许多 SAP Sybase IQ Web 服务功能都可用，但超出了本主题的范畴。

执行以下任务以创建 SAP Sybase IQ HTTP Web 服务器和 HTTP Web 服务：

1. 在装载 SAP Sybase IQ 数据库时启动 SAP Sybase IQ HTTP web 服务器。
   在命令提示符处运行以下命令：

   ```
   iqsrv16 -xs http(port=8082) iqdemo.db
   ```

   注意： 使用 iqsrv16 命令启动可在网络上访问的数据库服务器。

   -xs http(port=8082) 选项指示服务器在端口 8082 监听 HTTP 请求。如果已有 Web 服务器在端口 8082 上运行，请使用其它端口号。
2. 使用 CREATE SERVICE 语句创建 Web 服务，该服务响应进来的 Web 浏览器请求。
   a. 使用 Interactive SQL 运行以下命令来连接 `demo.db` 数据库：

      ```
      dbisql -c "dbf=iqdemo.db;uid=<user_id>;pwd=<password>"
      ```

   b. 在数据库中创建新 Web 服务。
      在 Interactive SQL 中执行以下 SQL 语句：

      ```
      CREATE SERVICE SampleWebService
          TYPE 'web-service-type-clause'
      ```

```
    AUTHORIZATION OFF
    USER DBA
    AS SELECT 'Hello world!';
```

使用所需 web 服务类型替换 *web-service-type-clause*。出于 Web 浏览器的兼容性考虑，建议使用 HTML 类型的子句。其它通用 HTTP Web 服务类型子句包括 XML、RAW 和 JSON。

CREATE SERVICE 语句创建 SampleWebService Web 服务，返回 SELECT 语句的结果集。在本例中，该语句返回 "Hello world!"

AUTHORIZATION OFF 子句表示不需授权即可访问 Web 服务。

USER DBA 语句表示服务语句应在 DBA 登录名下运行。

AS SELECT 子句允许服务从表或函数中选择数据，或直接查看数据。使用 AS CALL 子句同样也可以调用存储过程。

3. 在 Web 浏览器中查看 Web 服务。

   在运行 SAP Sybase IQ HTTP Web 服务器的计算机上，打开 Web 浏览器（例如 Internet Explorer 或 Firefox），然后转到以下 URL：

   ```
   http://localhost:8082/demo/SampleWebService
   ```

   此 URL 将使 Web 浏览器直接转到端口 8082 上的 HTTP Web 服务器。SampleWebService 输出 "Hello world"。结果集输出的显示格式由步骤 2 中的 *web-service-type-clause* 指定。

其它示例资源

示例包括在 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\http 目录下。

CodeXchange 上提供了其它示例，网址是 http://www.sybase.com/developer/codexchange。

如何启动 HTTP Web 服务器

如果用 -xs 服务器选项启动数据库服务器，SAP Sybase IQ HTTP Web 服务器将自动启动。此选项允许执行以下任务：

启用 Web 服务协议来监听 Web 服务请求。

配置网络协议选项，如服务器端口、记录、超时条件以及最大请求大小。

命令行的一般格式如下：

```
iqsrv16 -xs protocol-type(protocol-options) your-database-name.db
```

采用下列受支持的协议和任何适用的协议选项之一来替换 *protocol-type* 和 *protocol-options*：

- **HTTP** – 使用此协议监听 HTTP 连接。下面是一个示例。

  ```
  iqsrv16 -xs HTTP(PORT=8082) services.db
  ```

- **HTTPS** – 使用此协议监听 HTTPS 连接。支持 SSL 3.0 版和 TLS 1.0/1.1 版。下面是一个示例。

```
iqsrv16 -xs "HTTPS(FIPS=N;PORT=8082;IDENTITY="%ALLUSERSPROFILE
%"\SybaseIQ\samples\Certificates
\rsaserver.id;IDENTITY_PASSWORD=test)" services.db
```

注意： 网络协议选项对每个支持的协议均可用。这些选项可以控制协议行为，并可以在启动数据库服务器时在命令行中配置。

配置网络协议选项

网络协议选项都是可选的设置，提供对指定 Web 服务协议的控制。这些设置可在使用 -xs 数据库服务器选项启动数据库服务器时在命令行中配置。

例如，以下命令行可在指定了 PORT、FIPS、Identity 和 Identity_Password 网络协议选项的情况下配置 HTTPS 监听器：

```
iqsrv16 -xs https(PORT=544;FIPS=YES;
    IDENTITY=certificate.id;IDENTITY_PASSWORD=password) your-
database-name.db
```

该命令启动数据库服务器，从而为 `your-database-name.db` 数据库启用 HTTPS Web 服务协议。网络协议选项表示 Web 服务器应执行以下任务：

监听端口 544 而不是监听缺省 HTTPS 端口 (443)。
启用 FIPS 认可的安全算法对通信进行加密。
找到指定的标识文件 `certificate.id`，其中包含公共证书及其专用密钥。
将专用密钥与指定标识口令 `password` 进行比对校验。

下表标识了常用于 Web 服务协议的网络协议选项：

网络协议选项	可用 Web 服务协议	说明
DatabaseName (DBN) 协议选项	HTTP、HTTPS	指定处理 Web 请求时要使用的数据库名称，或者使用 REQUIRED 或 AUTO 关键字指定是否需要在 URL 中使用数据库名称。
FIPS 协议选项	HTTPS	启用经 FIPS 认可的安全算法来加密数据库文件、数据库客户端/服务器的通信和 Web 服务。
Identity 协议选项	HTTPS	指定用于安全 HTTPS 连接的标识文件的名称。
Identity_Password 协议选项	HTTPS	指定加密证书的口令。
LocalOnly (LO) 协议选项	HTTP、HTTPS	允许客户端选择只连接到本地计算机上的服务器（如果存在）。
LogFile (LOG) 协议选项	HTTP、HTTPS	指定数据库服务器用来记录 Web 请求信息的文件的名称。
LogFormat (LF) 协议选项	HTTP、HTTPS	控制写入日志文件（数据库服务器使用日志文件写入 Web 请求相关信息）的消息的格式，并指定出现在消息中的字段。

网络协议选项	可用 Web 服务协议	说明
LogOptions (LOPT) 协议选项	HTTP、HTTPS	指定数据库服务器用来写入 Web 请求信息的日志中所记录的消息类型。
ServerPort (PORT) 协议选项	HTTP、HTTPS	指定数据库服务器所监听的端口。

如何启动多个 HTTP Web 服务器

多 HTTP Web 服务器配置允许跨数据库创建 Web 服务，以使这些 Web 服务看起来像是单个 Web 站点的一部分。可利用 -xs 数据库服务器选项的多个实例来启动多个 HTTP Web 服务器。通过为每个 HTTP Web 服务器指定唯一的端口号来执行该任务。

示例

在本例中，以下命令行启动两个 HTTP web 服务 - 一个用于 `your-first-database.db`，而另一个用于 `your-second-database.db`：

```
iqsrv16 -xs http(port=80;dbn=your-first-
database),http(port=8800;dbn=your-second-database)
    your-first-database.db your-second-database.db
```

什么是 Web 服务

Web 服务指的是能促进计算机间数据传送和互操作性的软件。它们通过 Internet 提供业务逻辑段。在 HTTP Web 服务器中管理 Web 服务时，URL 将对客户端可用。指定 URL 时使用的约定决定了服务器与 Web 客户端的通信方式。

Web 服务管理包含下列任务：

选择要管理的 Web 服务类型。
创建和维护这些 Web 服务

Web 服务可以在 SQL Anywhere 数据库中创建和存储。

Web 服务类型

当 web 浏览器或客户端应用程序向 SAP Sybase IQ web 服务发出 web 服务请求时，请求将被处理并在响应中返回结果集。SAP Sybase IQ 支持数种 web 服务类型，这些服务类型控制着结果集的格式和结果集的返回方式。选择适当的 Web 服务类型后，即可用 CREATE SERVICE 或 ALTER SERVICE 语句的 TYPE 子句来指定 Web 服务器类型。

支持以下 Web 服务类型：

- **HTML** – 将语句、函数或过程的结果集设置为包含表的 HTML 文档格式。Web 浏览器显示 HTML 文档的主体。
- **XML** – 语句、函数或过程的结果集将以 XML 文档的形式返回。非 XML 设置格式的结果集会自动设置为 XML 格式。Web 浏览器显示原始 XML 代码，包括标记和属性。

XML 设置格式等同于在 SELECT 语句中使用 FOR XML RAW 子句，例如下面的 SQL 语句示例：

```
SELECT * FROM table-name FOR XML RAW
```

- **RAW** – 返回语句、函数或过程的结果集，但不自动设置格式。

  此服务类型可对结果集进行最大程度的控制。然而，必须在存储过程中显式写入必要的标记 (HTML、XML) 来生成响应。可以使用 SA_SET_HTTP_HEADER 系统过程来设置 HTTP Content-Type 标头以指定 MIME 类型，使 Web 浏览器能够正确地显示结果集。

- **JSON** – 以 JSON (JavaScript Object Notation) 格式返回语句、函数或过程的结果集。JavaScript Object Notation (JSON) 是一种针对 JavaScript 数据序列化而开发的基于文本且与语言无关的数据交换格式。JSON 表示四种基本类型：字符串、数字、布尔型和 NULL。JSON 还表示两种结构化类型：对象和数组。有关 JSON 的详细信息，请参见 http://www.json.org。

  AJAX 使用该服务来向 Web 应用程序进行 HTTP 调用。有关 JSON 类型的示例，请参见 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP\json_sample.sql`。

- **SOAP** – 语句、函数或过程的结果集将以 SOAP 响应的形式返回。SOAP 服务提供公共数据交换标准，以向支持 SOAP 的异构客户端应用程序提供数据访问。用 HTTP (SOAP over HTTP) 将 SOAP 请求和响应封装作为 XML 载荷传输。对 SOAP 服务的请求必须是有效 SOAP 请求，而不仅是一般的 HTTP 请求。SOAP 服务的输出可使用 CREATE 或 ALTER SERVICE 语句的 FORMAT 和 DATATYPE 属性进行调整。

- **DISH** – DISH (Determine SOAP Handler) 服务是一个 SAP Sybase IQ SOAP 端点。DISH 服务将公开 WSDL（Web Services Description Language，Web 服务描述语言）文档，该文档描述通过它能够访问到的所有 SOAP 操作（SAP Sybase IQ SOAP 服务）。SOAP 客户端工具箱用基于 WSDL 的接口构建客户端应用程序。SOAP 客户端应用程序将所有 SOAP 请求导引至 SOAP 端点（SAP Sybase IQ DISH 服务）。

示例

以下示例说明了如何创建使用 RAW 服务类型的通用 HTTP Web 服务：

```
CREATE PROCEDURE sp_echotext(str LONG VARCHAR)
BEGIN
    CALL sa_set_http_header( 'Content-Type', 'text/plain' );
    SELECT str;
END;

CREATE SERVICE SampleWebService
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL sp_echotext ( :str );
```

Web 服务维护

Web 服务维护包含下列任务：

- **创建或变更 Web 服务** – 创建或变更 Web 服务以提供支持 Web 浏览器界面的 Web 应用程序，并通过 REST 和 SOAP 方法来提供跨 Web 的数据交换。
- **删除 Web 服务** – 删除 Web 服务会导致随后发送给该服务的请求将返回 `404 Not Found` HTTP 状态消息。如果存在 `root` Web 服务，所有未解决的请求，不论是预期请求还是非预期请求，都将被处理。
- **对 Web 服务进行注释** – 注释是可选的，并允许为 Web 服务提供文档。
- **创建和自定义 root Web 服务** – 可以创建 `root` Web 服务来处理与任何其它 Web 服务请求都不匹配的 HTTP 请求。
- **启用与禁用 Web 服务** – 禁用的 Web 服务会返回 `404 Not Found` HTTP 状态消息。METHOD 子句指定可以为特定 Web 服务调用的 HTTP 方法。

如何创建或变更 Web 服务

创建或变更 Web 服务分别要求使用 CREATE SERVICE 或 ALTER SERVICE 语句。本节将说明如何用 Interactive SQL 执行这些语句来创建不同类型的 Web 服务。本节中的示例假定已通过 Interactive SQL 使用以下命令连接到 SAP Sybase IQ 数据库 *your-database.db*：

```
dbisql -c "dbf=your-database.db;uid=your-userid;pwd=your-password"
```

如何创建 HTTP Web 服务

HTTP Web 服务分为 HTML、XML 或 RAW。所有 HTTP Web 服务都可以用同样的 CREATE SERVICE 和 ALTER SERVICE 语句语法来创建或变更。

示例

在 Interactive SQL 中执行以下语句以在 HTTP Web 服务器中创建示例性通用 HTTP Web 服务：

```
CREATE SERVICE SampleWebService
    TYPE 'web-service-type-clause'
    URL OFF
    USER DBA
    AUTHORIZATION OFF
    AS sql-statement;
```

CREATE SERVICE 语句创建名为 `SampleWebService` 的新 Web 服务，并返回 *sql-statement* 的结果集。可以用 SELECT 或 CALL 语句替代 *sql-statement*，前者用于从表中选择数据或直接查看，后者用于调用数据库中的存储过程。

使用所需 web 服务类型替换 *web-service-type-clause*。HTTP Web 服务的有效子句包括 `HTML`、`XML`、`RAW` 和 `JSON`。

可以通过在 Web 浏览器中访问服务来查看 `SampleWebService` 服务的生成的结果集。

如何创建 SOAP over HTTP 服务
SOAP 是受多种开发环境支持的数据交换标准。

SOAP 载荷由 XML 文档组成，称为 SOAP 封装。SOAP 请求封装包含 SOAP 操作（SOAP 服务）并指定了所有相应参数。SOAP 服务对请求封装进行分析以获得参数，并像任何其它服务一样调用或选择存储过程或函数。SOAP 服务的表示层将结果集放置在 SOAP 封装中发送回客户端，结果集的格式是按照 DISH 服务的 WSDL 的指定预先定义好的。有关 SOAP 标准的详细信息，请参见 http://www.w3.org/TR/2000/NOTE-SOAP-20000508/。

缺省情况下，SOAP 服务参数和结果数据都作为 XmlSchema 字符串参数。DATATYPE ON 指定输入的参数和响应数据应使用 TRUE 类型。指定 DATATYPE 相应地更改了 WSDL 说明，以使客户端 SOAP 工具箱以相应类型的参数和响应对象生成接口。

FORMAT 子句用于以多种功能定向特定的 SOAP 工具箱。DNET 提供 Microsoft .NET 客户端应用程序以将 SOAP 服务响应作为 System.Data.DataSet 对象来使用。CONCRETE 公开更加通用的结构，这种结构允许面向对象的应用程序，如 .NET 或 Java 生成包含行和列的响应对象。XML 将整个响应返回为 XML 文档，将其公开为字符串。客户端可使用 XML 分析程序进一步处理数据。CREATE SERVICE 语句的 FORMAT 子句支持多种客户端应用程序类型。

注意： DATATYPE 子句只和 SOAP 服务有关（HTML 中没有数据分类）。FORMAT 子句可以为 SOAP 或 DISH 服务指定。SOAP 服务 FORMAT 规定会替换 DISH 服务的规定。

示例

在 Interactive SQL 中执行下列语句来创建 SOAP over HTTP 服务：

```
CREATE SERVICE SampleSOAPService
    TYPE 'SOAP'
    DATATYPE ON
    FORMAT 'CONCRETE'
    USER DBA
    AUTHORIZATION OFF
    AS sql-statement;
```

如何创建 DISH 服务
SAP Sybase IQ 允许创建 DISH 服务，这些服务为 SOAP 服务组充当 SOAP 端点。DISH 服务还会自动构造 WSDL（Web Services Description Language，Web 服务描述语言）文档，允许 SOAP 客户端工具箱生成与 WSDL 中描述的 SOAP 服务进行数据交换所需的接口。

不需对 DISH 服务进行维护即可对 SOAP 服务进行添加或删除，因为当前运行的 SOAP over HTTP 服务集始终是公开的。

示例

在 Interactive SQL 中执行以下 SQL 语句以在 HTTP Web 服务器中创建示例性 SOAP 和 DISH 服务：

```
CREATE SERVICE "Samples/TestSoapOp"
    TYPE 'SOAP'
    DATATYPE ON
    USER DBA
    AUTHORIZATION OFF
    AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
RESULT( ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR )
BEGIN
    SELECT i, f, s;
END;

CREATE SERVICE "dnet_endpoint"
    TYPE 'DISH'
    GROUP "Samples"
    FORMAT 'DNET';
```

第一个 CREATE SERVICE 语句创建一个名为 `Samples/TestSoapOp` 的新 SOAP 服务。

第二个 CREATE SERVICE 语句创建一个名为 `dnet_endpoint` 的新 DISH 服务。GROUP 子句的 `Samples` 部分标识了要公开的 SOAP 服务组。可以查看 DISH 服务生成的 WSDL 文档。在计算机上运行 SAP Sybase IQ Web 服务器时，可以用 `http://localhost:`*port-number*`/dnet_endpoint` URL 来访问服务，其中 *port-number* 为运行服务器的端口号。

在本例中，SOAP 服务没有包含决定 SOAP 响应格式的 FORMAT 子句。因此，SOAP 响应格式由 DISH 服务来决定，但它不会替代 SOAP 服务的 FORMAT 子句。本功能允许创建同类 DISH 服务，这样每个 DISH 端点就能以多种功能为 SOAP 客户端提供服务。

创建同类 DISH 服务

当 SOAP 服务定义将 FORMAT 子句的规范交给 DISH 服务决定时，一套 SOAP 服务可以在定义格式的 DISH 服务内集合成组。多个 DISH 服务则可以用不同的 FORTMAT 规范来公开同一组 SOAP 服务。如果要详述示例 `TestSoapOp`，可以使用下列 SQL 语句来创建另一个名为 `java_endpoint` 的 DISH 服务：

```
CREATE SERVICE "java_endpoint"
    TYPE 'DISH'
    GROUP "Samples"
    FORMAT 'CONCRETE';
```

在本例中，当 SOAP 客户端通过 `java_endpoint` DISH 服务为 `TestSoapOp` 操作发出 Web 服务请求时，它将接收到一个名为 `TestSoapOp_Dataset` 的响应对象。

可以检查 WSDL 以比较 dnet_endpoint 和 java_endpoint 之间的差异。使用此技术，可以快速构建满足特定 SOAP 客户端工具箱需要的 SOAP 端点。

如何删除 Web 服务

删除 Web 服务会导致随后发送给该服务的请求将返回 404 Not Found HTTP 状态消息。如果存在 root Web 服务，所有未解决的请求，不论是预期请求还是非预期请求，都将被处理。

示例

执行下列 SQL 语句来删除名为 SampleWebService 的 Web 服务：

```
DROP SERVICE SampleWebService;
```

如何对 Web 服务进行注释

为 Web 服务提供文档要求使用 COMMENT ON SERVICE 语句。将 *statement* 子句设置为空值可以删除注释。

示例

例如，执行下列 SQL 语句为名为 SampleWebService 的 Web 服务创建新的注释：

```
COMMENT ON SERVICE SampleWebService
    IS "This is a comment on my web service.";
```

如何创建和自定义 Root Web 服务

如果定义了 root Web 服务，不匹配任何 Web 服务请求的 HTTP 客户端请求将由 root Web 服务处理。

root Web 服务提供简便且灵活的方法来处理在构建应用程序时 URL 不必知道的任意 HTTP 请求，以及处理无法识别的请求。

示例

本示例说明如何使用 root Web 服务（存储在数据库中的一个表中）为 Web 浏览器和其它 HTTP 客户端提供内容。假设在单一数据库上启动了本地 HTTP Web 服务器，并监听端口 80。所有的脚本都在 Web 服务器上运行。

通过 Interactive SQL 连接数据库服务器，并执行以下 SQL 语句来创建 root Web 服务，从而向名为 PageContent 的过程传递 URL 主机参数（由客户端提供）：

```
CREATE SERVICE root
    TYPE 'RAW'
    AUTHORIZATION OFF
    SECURE OFF
    URL ON
    USER DBA
    AS CALL PageContent(:url);
```

URL ON 部分指定完整的路径组成部分可通过名为 URL 的 HTTP 变量访问。

执行以下 SQL 语句来创建存储页面内容的表。在本例中，页面内容由其 URL、MIME 类型和内容自身决定。

```
CREATE TABLE Page_Content (
    url          VARCHAR(1024) NOT NULL PRIMARY KEY,
    content_type VARCHAR(128)  NOT NULL,
    image        LONG VARCHAR  NOT NULL
);
```

执行以下 SQL 语句来填充该表。在本例中，意图是在 index.html 页面被请求后定义向 HTTP 客户端提供的内容。

```
INSERT INTO Page_Content
VALUES(
    'index.html',
    'text/html',
    '<html><body><h1>Hello World</h1></body></html>'
);
COMMIT;
```

执行以下 SQL 语句来实施 PageContent 过程，此过程接受一并传递给 root Web 服务的 URL 主机变量：

```
CREATE PROCEDURE PageContent(IN @url LONG VARCHAR)
RESULT ( html_doc LONG VARCHAR )
BEGIN
    DECLARE @status CHAR(3);
    DECLARE @type   VARCHAR(128);
    DECLARE @image  LONG VARCHAR;

    SELECT content_type, image INTO @type, @image
        FROM Page_Content
        WHERE url = @url;

    IF @image is NULL THEN
        SET @status = '404';
        SET @type = 'text/html';
        SET @image =  '<html><body><h1>404 - Page Not Found</h1>'
            || '<p>There is no content located at the URL "'
            || html_encode( @url ) || '" on this server.<p>'
            || '</body></html>';
    ELSE
        SET @status = '200';
    END IF;
    CALL sa_set_http_header( '@HttpStatus', @status );
    CALL sa_set_http_header( 'Content-Type', @type );
    SELECT @image;
END;
```

当向 HTTP 服务器发送的请求不与任何其它已定义的 Web 服务 URL 匹配时，root Web 服务将调用 PageContent 过程。该过程检查客户端提供的 URL 是否和 Page_Content 表中的 URL 匹配。SELECT 语句向客户端发送响应。如果无法在表中找到客户端提供的 URL，则构建通用的 404 - Page Not Found html 页面，并将其发送给客户端。

有些浏览器会用其自身页面响应 404 状态，所以不能保证一定会显示通用页面。

在错误消息中，HTML_ENCODE 函数用于对客户端提供的 URL 中的特殊字符进行编码。

@HttpStatus 标头用于设置请求所返回的状态代码。404 状态表示 Not Found 错误，而 200 状态表示 OK。"Content-Type" 标头用于设置请求所返回的内容类型。在本例中，index.html 页面的内容 (MIME) 类型为 text/html。

Web 服务 SQL 语句
以下 SQL 语句有助于开发 Web 服务：

Web 服务器相关 SQL 语句	说明
CREATE SERVICE 语句 [HTTP Web 服务]	创建新的 HTTP Web 服务。
CREATE SERVICE 语句 [SOAP Web 服务]	创建新的 SOAP over HTTP 或 DISH 服务。
ALTER SERVICE 语句 [HTTP Web 服务]	变更现有 HTTP Web 服务。
ALTER SERVICE 语句 [SOAP Web 服务]	变更现有 SOAP over HTTP 或 DISH 服务。
COMMENT 语句	在系统表中存储关于数据库对象的注释。 使用以下语法对 Web 服务进行注释： `COMMENT ON SERVICE 'web-service-name' IS 'your comments'`
DROP SERVICE 语句	删除 Web 服务。

Web 服务的连接池
每个公开了 Web 服务的数据库都可以访问数据库连接池。池用用户名进行分组，从而在给定 USER 子句下定义的所有服务共享同一个连接池组。

第一次执行查询的服务请求必须经历优化阶段才能建立执行计划。如果计划可以高速缓存并重新使用，则可以为后续执行跳过优化阶段。HTTP 连接池通过尽可能地重新使用数据库连接中的计划高速缓存来利用它们。每个服务都维护自己要优化重用的连接列表。然而，在负载高峰期间，服务会在同一连接用户组中挪用使用最少的连接。

随着时间推移，给定的连接可以获得高速缓存的计划，这些计划能够为多个服务的执行优化性能。

在连接池中，给定服务的 HTTP 请求会尝试从池中获取数据库连接。与 HTTP 会话不同，池中的连接可以通过重置连接范围环境（例如连接范围变量和临时表）来清除。

就授权而言，存放在 HTTP 连接池中的数据库连接不计为使用中的连接。从池中获取连接时，它们将计为授权连接。当 HTTP 请求在从池中获取连接时，如果超过了许可的限制，将返回 503 Service Temporarily Unavailable 状态。

当定义为 AUTHORIZATION OFF 时，Web 服务只能使用一个连接池。

当数据库和连接选项出现变动时，池中的数据库连接不会更新。

如何在 HTTP Web 服务器中开发 Web 服务应用程序

本节将概述 Web 页面的创建和自定义。将介绍如何为 HTTP Web 服务器开发存储过程。本节假设您懂得如何启动 SAP Sybase IQ HTTP Web 服务器以及创建调用存储过程的 Web 服务。

有关 Web 服务应用程序的详细示例，请参见 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP 目录。

如何自定义 Web 页

必须先确定 HTTP Web 服务器调用的 Web 服务格式以自定义 Web 页面。例如，当 Web 服务指定 HTML 类型时，Web 页面将是 HTML 格式。

RAW Web 服务类型能提供最多的自定义选项，因为它需要 Web 服务过程和函数显式地要求代码来提供需求的标记，如 HTML 或 XML。使用 RAW 类型时，必须执行以下任务来自定义 Web 页面：

- 在调用存储过程中，将 HTTP Content-Type 标头字段设置为适当的 MIME 类型，例如 text/html。
- 当从调用存储过程中生成 Web 页面输出时，为 MIME 类型应用适当的标记。

示例

以下示例将说明如何通过指定 RAW 类型来创建新的 Web 服务：

```
CREATE SERVICE WebServiceName
    TYPE 'RAW'
    AUTHORIZATION OFF
    URL ON
    USER DBA
    AS CALL HomePage( :url );
```

在本例中，Web 服务调用了 HomePage 存储过程，这在定义单个接收 URL PATH 组成部分的 URL 参数时是必需的。

设置 Content-Type 标头字段

使用 sa_set_http_header 系统过程定义 HTTP Content-Type 标头以确保 Web 浏览器能正确地显示其内容。

以下示例将说明如何通过 sa_set_http_header 系统过程使用 text/html MIME 类型来设置 HTML 输出的 Web 页面格式：

```
CREATE PROCEDURE HomePage (IN url LONG VARCHAR)
    RESULT (html_doc XML)
```

```
    BEGIN
        CALL sa_set_http_header ( 'Content-Type', 'text/html' );
        -- Your SQL code goes here.
        ...
    END
```

应用 MIME 类型的标记约定
在存储过程中必须应用 Content-Type 标头指定的 MIME 类型的标记约定。SAP Sybase IQ 提供了几种函数来创建标记。

以下示例将说明如何使用 XMLCONCAT 和 XMLELEMENT 函数来生成 HTML 内容，前提是使用了 sa_set_http_header 系统过程将 Content-Type 标头设置为 text/html MIME 类型：

```
XMLCONCAT(
   CAST('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">' AS XML),
    XMLELEMENT(
         'HTML',
         XMLELEMENT(
             'HEAD',
             XMLELEMENT('TITLE', 'My Home Page')
         ),
         XMLELEMENT(
             'BODY',
             XMLELEMENT('H1', 'My home on the web'),
             XMLELEMENT('P', 'Thank you for visiting my web site!')
         )
    )
)
```

由于元素内容始终会被转义（除非数据类型为 XML），所以上例使用 CAST 函数。否则会转义出特殊字符（例如用 < 来表示 <）。

如何访问客户端提供的 HTTP 变量与标头
HTTP 客户端请求中的变量和标头可以用下列方式之一访问：

- 将它们作为存储函数和过程调用的主机参数进行传递的 Web 服务语句声明。
- 在存储函数或过程中调用 HTTP_VARIABLE、NEXT_HTTP_VARIABLE、HTTP_HEADER、NEXT_HTTP_HEADER 函数。

如何使用主机参数访问 HTTP 变量
在把客户端提供的变量当作函数或过程调用的主机参数传递时，可以引用它们。

示例

以下示例说明如何访问在名为 ShowTable 的 Web 服务中使用的主机参数：

```
CREATE SERVICE ShowTable
    TYPE 'RAW'
    AUTHORIZATION ON
    AS CALL ShowTable( :user_name, :table_name );

CREATE PROCEDURE ShowTable(IN username VARCHAR(128), IN tblname
VARCHAR(128))
```

```
BEGIN
    -- write SQL code utilizing the username and tblname variables
here.
END;
```

服务主机参数按照过程参数的声明顺序映射。在上例中，user_name 和 table_name 主机参数分别映射至 username 和 tblname 参数。

如何使用 Web 服务函数访问 HTTP 变量和标头

HTTP_VARIABLE、NEXT_HTTP_VARIABLE、HTTP_HEADER 和 NEXT_HTTP_HEADER 函数可用于遍历客户端提供的变量和标头。

使用 HTTP_VARIABLE 和 HTTP_NEXT_VARIABLE 访问变量

可使用存储过程中的 NEXT_HTTP_VARIABLE 和 HTTP_VARIABLE 函数遍历所有客户端提供的变量。

使用 HTTP_VARIABLE 函数可得到变量名的值。

NEXT_HTTP_VARIABLE 函数允许您遍历客户端发送的所有变量。第一次调用它去获取第一个变量名时，它将传递 NULL 值。使用返回的变量名作为调用 HTTP_VARIABLE 函数的一个参数来得到它的值。将前面的变量名传递给 next_http_variable 调用将得到下一个变量名。最后一个变量名被传递后，将返回 Null。

遍历变量名可以保证每个变量名都刚好返回一次，但变量名的顺序不一定与它们在客户端请求中出现的顺序相同。

以下示例说明如何使用 HTTP_VARIABLE 函数来检索客户端请求提供的参数的值，该请求访问 ShowDetail 服务：

```
CREATE SERVICE ShowDetail
    TYPE 'HTML'
    URL PATH OFF
    AUTHORIZATION OFF
    USER DBA
    AS CALL ShowDetail();

CREATE PROCEDURE ShowDetail()
BEGIN
    DECLARE v_customer_id LONG VARCHAR;
    DECLARE v_product_id LONG VARCHAR;
    SET v_customer_id = HTTP_VARIABLE( 'customer_id' );
    SET v_product_id = HTTP_VARIABLE( 'product_id' );
    CALL ShowSalesOrderDetail( v_customer_id, v_product_id );
END;
```

以下示例说明如何从与 image 变量关联的标头字段值中检索三个属性。

```
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );
```

提供整数作为第二个参数使您可以检索其它值。第三个参数允许您从包含多个部分的请求中检索标头字段值。提供标头字段的名称以检索其值。

使用 HTTP_HEADER 和 NEXT_HTTP_HEADER 访问标头
可使用 NEXT_HTTP_HEADER 和 HTTP_HEADER 函数从请求中获得 HTTP 请求标头。

HTTP_HEADER 函数将返回指定 HTTP 标头字段的值。

NEXT_HTTP_HEADER 函数将遍历 HTTP 标头并返回下一个 HTTP 标头名。如果用 NULL 调用该函数，该函数会返回第一个标头的名称。通过向此函数传递上一个标头的名称来检索后续的标头。最后一个标头名被调用后，将返回 NULL。

下表列出了一些通用 HTTP 请求标头和典型值：

标头名	标头值
Accept	image/gif、image/x-xbitmap、image/jpeg、image/pjpeg、application/x-shockwave-flash、application/vnd.ms-excel、application/vnd.ms-powerpoint、application/msword、*/*
Accept-Language	en-us
Accept-Charset	utf-8, iso-8859-5;q=0.8
Accept-Encoding	gzip、deflate
User-Agent	Mozilla/4.0（兼容；MSIE 7.0；Windows NT 5.2；WOW64；SV1；.NET CLR 2.0.50727）
Host	localhost:8080
Connection	Keep-Alive

下表列出了特殊标头和典型值：

标头名	标头值
@HttpMethod	GET
@HttpURI	/demo/ShowHTTPHeaders
@HttpVersion	HTTP/1.1
@HttpQueryString	id=-123&version=109&lang=en

可以使用 @HttpStatus 特殊标头来设置正在处理的请求的状态代码。

以下示例将说明如何在 HTML 表中设置标头名和值的格式。

创建 `ShowHTTPHeaders` Web 服务：

```
CREATE SERVICE ShowHTTPHeaders
    TYPE 'RAW'
```

```
    AUTHORIZATION OFF
    USER DBA
    AS CALL HTTPHeaderExample();
```

创建 HTTPHeaderExample 过程，该过程使用 NEXT_HTTP_HEADER 函数获取标头名，然后使用 HTTP_HEADER 函数检索其值：

```
CREATE PROCEDURE HTTPHeaderExample()
RESULT ( html_string LONG VARCHAR )
BEGIN
    declare header_name LONG VARCHAR;
    declare header_value LONG VARCHAR;
    declare header_query LONG VARCHAR;
    declare table_rows XML;
    set header_name = NULL;
    set table_rows = NULL;
header_loop:
    LOOP
        SET header_name = NEXT_HTTP_HEADER( header_name );
        IF header_name IS NULL THEN
            LEAVE header_loop
        END IF;
        SET header_value = HTTP_HEADER( header_name );
        SET header_query = HTTP_HEADER( '@HttpQueryString' );
        -- Format header name and value into an HTML table row
        SET table_rows = table_rows ||
            XMLELEMENT( name "tr",
                XMLATTRIBUTES( 'left' AS "align",
                               'top' AS "valign" ),
                XMLELEMENT( name "td", header_name ),
                XMLELEMENT( name "td", header_value ),
                XMLELEMENT( name "td", header_query ) );

    END LOOP;
    SELECT XMLELEMENT( name "table",
                XMLATTRIBUTES( '' AS "BORDER",
                             '10' AS "CELLPADDING",
                              '0' AS "CELLSPACING" ),
                XMLELEMENT( name "th",
                    XMLATTRIBUTES( 'left' AS "align",
                                   'top'  AS "valign" ),
                                   'Header Name' ),
                XMLELEMENT( name "th",
                    XMLATTRIBUTES( 'left' AS "align",
                                   'top'  AS "valign" ),
                                   'Header Value' ),
                XMLELEMENT( name "th",
                    XMLATTRIBUTES( 'left' AS "align",
                                   'top'  AS "valign" ),
                                   'HTTP Query String' ),
                table_rows );
END;
```

在 Web 浏览器中访问 ShowHTTPHeaders 来观察 HTML 表中设置的请求标头。

如何访问客户端提供的 SOAP 请求标头

可将 NEXT_SOAP_HEADER 和 SOAP_HEADER 函数结合使用来获得 SOAP 请求中的标头。

NEXT_SOAP_HEADER 函数将遍历包含在 SOAP 请求封装中的 SOAP 标头并返回下一个 SOAP 标头名。如果用 NULL 调用该函数，该函数会返回第一个标头的名称。通过向 NEXT_SOAP_HEADER 函数传递上一个标头的名称来检索后续的标头。当用最后一个标头的名称调用该函数时，该函数返回 NULL。

下面的示例将说明 SOAP 标头检索：

```
SET hd_key = NEXT_SOAP_HEADER( hd_key );
    IF hd_key IS NULL THEN
      -- no more header entries
      LEAVE header_loop;
    END IF;
```

重复调用该函数将会返回所有的标头字段且仅返回一次，但不一定与其出现在 SOAP 请求中的顺序相同。

SOAP_HEADER 函数将返回指定的 SOAP 标头字段的值或 NULL（如果不是从 SOAP 服务调用）。当通过 Web 服务处理 SOAP 请求时，将使用该函数。如果给定字段名的标头不存在，则会返回值 NULL。

该示例将搜索名为 Authentication 的 SOAP 标头。当它找到此标头时，便会抽取整个 SOAP 标头的值以及 `@namespace` 和 `mustUnderstand` 属性的值。SOAP 标头值可能类似于下面的 XML 字符串：

```
<Authentication xmlns="CustomerOrderURN" mustUnderstand="1">
  <userName pwd="none">
    <first>John</first>
    <last>Smith</last>
  </userName>
</Authentication>
```

对于此标头，`@namespace` 属性值将为 `CustomerOrderURN`

同时，`mustUnderstand` 属性值将为 1

使用设置为 `/*:Authentication/*:userName` 的 XPath 字符串来通过 OPENXML 函数分析此 XML 字符串的内部。

```
SELECT * FROM OPENXML( hd_entry, xpath )
    WITH ( pwd LONG VARCHAR '@*:pwd',
        first_name LONG VARCHAR '*:first/text()',
        last_name LONG VARCHAR '*:last/text()' );
```

通过使用上面所示的示例 SOAP 标头值，SELECT 语句将会创建一个如下所示的结果集：

pwd	first_name	last_name
none	John	Smith

在此结果集上会声明一个游标，且会将三个列值读取到三个变量中。此时，您获取了传递给 Web 服务的所有相关信息。

示例

下面的例子将说明 Web 服务器如何处理带有参数和 SOAP 标头的 SOAP 请求。本示例实现了一个需要两个参数的 addItem SOAP 操作：amount 是整数类型而 item 是字符串类型。sp_addItems 过程处理了一个 Authentication SOAP 标头，即将用户的名和姓抽取出来。这些值用于通过 sa_set_soap_header 系统过程来填充 SOAP 响应的 Validation 标头。响应是由三列组成的结果：quantity、item 和 status，分别为 INT、LONG VARCHAR 和 LONG VARCHAR 类型。

```
// create the SOAP service
CREATE SERVICE addItems
    TYPE 'SOAP'
    FORMAT 'CONCRETE'
    AUTHORIZATION OFF
    USER DBA
    AS CALL sp_addItems( :amount, :item );

// create SOAP endpoint for related services
CREATE SERVICE itemStore
    TYPE 'DISH'
    AUTHORIZATION OFF
    USER DBA;

// create the procedure that will process the SOAP requests for the
addItems service
CREATE PROCEDURE sp_addItems(count INT, item LONG VARCHAR)
RESULT(quantity INT, item LONG VARCHAR, status LONG VARCHAR)
BEGIN
    DECLARE hd_key LONG VARCHAR;
    DECLARE hd_entry LONG VARCHAR;
    DECLARE pwd LONG VARCHAR;
    DECLARE first_name LONG VARCHAR;
    DECLARE last_name LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE authinfo LONG VARCHAR;
    DECLARE namespace LONG VARCHAR;
    DECLARE mustUnderstand LONG VARCHAR;

  header_loop:
    LOOP
        SET hd_key = next_soap_header( hd_key );
        IF hd_key IS NULL THEN
            // no more header entries.
            leave header_loop;
        END IF;
        IF hd_key = 'Authentication' THEN
            SET hd_entry = soap_header( hd_key );
            SET xpath = '/*:' || hd_key || '/*:userName';
            SET namespace = soap_header( hd_key, 1, '@namespace' );
            SET mustUnderstand = soap_header( hd_key, 1,
'mustUnderstand' );
```

```
            BEGIN
                // parse for the pieces that you are interested in
                DECLARE crsr CURSOR FOR SELECT * FROM
                    OPENXML( hd_entry, xpath )
                        WITH ( pwd LONG VARCHAR '@*:pwd',
                           first_name LONG VARCHAR '*:first/text()',
                            last_name LONG VARCHAR '*:last/text()');
                OPEN crsr;
                FETCH crsr INTO pwd, first_name, last_name;
                CLOSE crsr;
            END;
           // build a response header, based on the pieces from the
request header
            SET authinfo = XMLELEMENT( 'Validation',
                XMLATTRIBUTES(
                    namespace as xmlns,
                    mustUnderstand as mustUnderstand ),
                    XMLELEMENT( 'first', first_name ),
                    XMLELEMENT( 'last', last_name ) );
            CALL sa_set_soap_header( 'authinfo', authinfo);
        END IF;
    END LOOP header_loop;
    //  code to validate user/session and check item goes here...
    SELECT count, item, 'available';
END;
```

在 HTTP 服务器上管理 HTTP 会话

Web 应用程序可以各种方式支持会话。在 HTML 窗体中的隐藏字段，可用于跨多个请求保存客户端/服务器数据。另外，Web 2.0 技术（如启用了 AJAX 的客户端 JavaScript）也可以基于客户端状态作出异步 HTTP 请求。SAP Sybase IQ 提供了额外的功能，可以保留数据库连接专供会话 HTTP 请求使用。

任何在 HTTP 会话内部创建和变更的连接范围变量和临时表都可以为指定了给定 SessionID 的后续 HTTP 请求所访问。SessionID 可以通过 GET 或 POST HTTP 请求方法指定，或在 HTTP Cookie 标头内指定。当 HTTP 请求与 SessionID 变量一起被接收时，服务器将检查其会话存储库以寻找匹配上下文。如果找到一个会话，服务器将使用它的数据库连接来处理请求。如果会话在使用中，服务器会将 HTTP 请求加入队列，并在会话被释放时激活它。

sa_set_http_option 可用于创建、删除和更改会话 ID。

HTTP 会话要求对管理会话标准进行特殊处理。一个给定的 SessionID 只能有一个数据库连接供其使用，所以向该 SessionID 发出的连续请求将被服务器序列化。一个给定 SessionID 的队列中最多可以有 16 个请求。如果会话队列已满，则随后向该 SessionID 发出的请求将用 `503 Service Unavailable` 状态拒绝掉。

第一次创建 SessionID 时，此 SessionID 将立即被系统注册。随后要修改或删除该 SessionID 的请求仅在给定 HTTP 请求终止后生效。如果请求处理结果导致回退，或如果应用程序删除或重置了该 SessionID，此方法可以促成一致的行为。

如果 HTTP 请求更改了 SessionID，当前的会话将被删除，并由待执行的会话替换。被会话高速缓存的数据库连接会高效移动到新的会话上下文，且会保留所有状态数据(如临时表和创建的变量）。

有关 HTTP 会话用法的完整示例，请参见 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP\session.sql`。

注意： 应删除失效的会话，并应当设置适当的超时时间使未完成的连接数目降到最低，因为每个客户端应用程序连接都持有一个许可座席。与 HTTP 会话关联的连接在连接期间将维持它们对服务器数据库的持有。

有关许可的详细信息，请访问 http://www.sybase.com/detail?id=1056242。

如何创建 HTTP 会话

使用 sa_set_http_option 系统过程的 SessionID 选项可以创建会话。会话 ID 可以由任何非空字符串定义。

URL 和 Cookie 支持会话状态管理。要访问 HTTP 会话，可以用 HTTP Cookie，也可以通过 GET 请求的 URL，或者从 POST (x-www-form-urlencoded) 请求的主体内部来访问。例如，以下 URL 在执行时使用了 `XYZ` 数据库连接：

```
http://localhost/sa_svc?SESSIONID=XYZ
```

如果 `XYZ` 数据库连接不存在，请求将作为标准的非会话请求处理。

示例

下列代码将说明如何创建一个能够创建和删除会话的 `RAW` Web 服务。名为 `request_count` 的连接范围变量将随着指定有效 SessionID 时发出的 HTTP 请求递增。

```
CREATE SERVICE mysession
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL mysession_proc();

CREATE PROCEDURE mysession_proc()
BEGIN
    DECLARE body LONG VARCHAR;
    DECLARE hostname LONG VARCHAR;
    DECLARE svcname LONG VARCHAR;
    DECLARE sesid LONG VARCHAR;

    CALL sa_set_http_header ( 'Content-Type', 'text/html' );
    SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
    SELECT CONNECTION_PROPERTY('HttpServiceName') INTO svcname;
    SELECT HTTP_HEADER( 'Host' ) INTO hostname;
    IF HTTP_VARIABLE('delete') IS NOT NULL THEN
        CALL sa_set_http_option( 'SessionID', NULL );
        SET body = '<html><body>Deleted ' || sesid
          || '</BR><a href="http://' || hostname || '/' || svcname ||
'">Start Again</a>';
```

```
        SELECT body;
    END IF;
    IF sesid = '' THEN
        SET sesid = set_session_url();
        CREATE VARIABLE request_count INT;
        SET request_count = 0;

        SET body = '<html><body> Created session ID ' || sesid
           || '</br><a href="http://' || hostname || '/' || svcname
           || '?SessionID=' || sesid || '"> Enter into Session</a>';
    ELSE
        SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
        SET request_count = request_count +1;
        SET body = '<html><body>Session ' || sesid || '</br>'
          || 'created ' || CONNECTION_PROPERTY('SessionCreateTime')
|| '</br>'
           || 'last access ' ||
CONNECTION_PROPERTY('SessionLastTime') || '</br>'
           || 'connection ID ' || CONNECTION_PROPERTY('Number') ||
'</br>'
         || '<h3>REQUEST COUNT is '|| request_count || '</h3><hr></
br>'
           || '<a href="http://' || hostname || '/' || svcname
           || '?SessionID=' || sesid || '">Enter into Session</a></
br>'
           || '<a href="http://' || hostname || '/' || svcname
           || '?SessionID=' || sesid || '&delete">Delete Session</
a>';
    END IF;

    SELECT body;
END;
```

如何使用 URL 管理会话

在 URL 会话状态管理系统中，客户端应用程序或 Web 浏览器在 URL 中提供会话 ID。

示例

以下示例将说明如何在 HTTP Web 服务器 SQL 函数中创建唯一的会话 ID，其中会话 ID 只能由 URL 提供：

```
CREATE FUNCTION set_session_url()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
       CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND,
tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;
```

如果没有为连接指定 session_id，则会形成一个非会话连接，SessionID 会被表示为一个空字符串。

如果 session_id 为另外一个 HTTP 请求所拥有，则 sa_set_http_option 系统过程将返回一条错误。

如何使用 Cookie 管理会话
在 Cookie 会话状态管理系统中，客户端应用程序或 Web 浏览器在 HTTP Cookie 标头而非 URL 中提供会话 ID。sa_set_http_header 系统过程的 "Set-Cookie" HTTP 响应标头支持 Cookie 会话管理。

注意： 如果 Cookie 在客户端应用程序或 Web 浏览器上被禁用，将无法依赖 Cookie 状态管理。建议同时支持 URL 和 Cookie 状态管理。如果 URL 和 Cookie 都提供了会话 ID，将使用 URL 提供的会话 ID。

示例

以下示例将说明如何在 HTTP Web 服务器 SQL 函数中创建唯一的会话 ID，其中会话 ID 可由 URL 或 Cookie 提供：

```
CREATE FUNCTION set_session_cookie()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
       CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND,
tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    CALL sa_set_http_header( 'Set-Cookie',
        'sessionid=' || session_id || ';' ||
        'max-age=60;' ||
        'path=/session;' );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;
```

如何检测不活动的 HTTP 会话
SessionCreateTime 和 SessionLastTime 连接属性可以用于决定当前连接是否位于会话上下文中。如果两个连接属性查询中的任意一个返回了空字符串，则 HTTP 请求没有运行在会话上下文中。

SessionCreateTime 连接属性提供了用于确定给定会话创建时间的度量。调用 sa_set_http_option 系统过程建立 SessionID 时，它将初次被定义。

SessionLastTime 连接属性根据前一次请求的终止时刻来提供上次处理的会话请求释放数据库连接的时间。如果会话是第一次创建，在创建者请求释放连接之前，它都将返回为空字符串。

注意： 可以用 http_session_timeout 选项来调整会话超时时间。

示例

以下示例将说明如何使用 SessionCreateTime 和 SessionLastTime 连接属性来检测会话：

```
SELECT CONNECTION_PROPERTY( 'sessioncreatetime' ) INTO ses_create;
SELECT CONNECTION_PROPERTY( 'sessionlasttime' ) INTO ses_last;
```

如何删除 HTTP 会话或更改会话 ID

显式删除在会话上下文中进行高速缓存处理的数据库连接会导致会话被删除。用这种方式删除会话是一种取消操作；从该会话队列中释放的所有请求都将进入取消状态。该行为确保任何等待会话的未处理请求都被终止。类似地，关闭服务器或数据库也会取消所有的数据库连接。

将 sa_set_http_option 系统过程中的 SessionID 选项设置为 NULL 或空字符串可以删除会话。

以下代码可用于删除会话：

```
CALL sa_set_http_option( 'SessionID', null );
```

当 HTTP 会话被删除或 SessionID 出现变更，任何在会话队列中等待的待执行 HTTP 请求都将被释放，并允许脱离会话上下文。待处理的请求不会重新使用同样的数据库连接。

会话 ID 不能设置为已存在的会话 ID，如果 SessionID 发生变更，引用了旧 SessionID 的待处理请求将被释放并作为非会话的请求运行。引用了新 SessionID 的后续请求将重新使用旧 SessionID 实例化的相同数据库连接。

删除或更改 HTTP 会话时，以下条件适用：

- 行为是不同的，这取决于当前请求是否继承了会话而获得了属于会话的数据库连接，或者非会话请求是否实例化了新的会话。如果请求以非会话开始，则创建或删除会话的行为将立即发生。如果请求继承了会话，则会话状态中的变更（例如删除会话或改变 SessionID）只能在请求终止且其变更提交后发生。行为中的区别强调处理客户端使用同样的 SessionID 同时发出请求时可能发生的异常。
- 将会话更改为当前会话的 SessionID（没有待执行会话）不会出错，并且没有明显的影响。
- 将会话更改为另一个 HTTP 请求正在使用的 SessionID 会出错。
- 在某更改已为待执行状态时更改会话将导致待执行会话被删除，并创建新的待执行会话。待执行的会话只能在请求成功终止后激活。
- 将带有待执行会话的会话更改回其原始 SessionID 将导致待执行会话在没有被当前会话改变的情况下被删除。

HTTP 会话管理

HTTP 请求创建的会话会立即实例化，以便该会话将需要该会话上下文的任何后续 HTTP 请求进行排队。

在本例中，本地主机客户端可以访问会话，其指定的会话 ID 为 session_63315422814117，运行于数据库 dbname 中，且当会话在服务器上用 sa_set_http_option 过程被创建后会用下列 URL 运行 session_service 服务。

```
http://localhost/dbname/session_service?
sessionid=session_63315422814117
```

Web 应用程序可能需要一种用于跟踪 HTTP Web 服务器内活动会话使用情况的方法。使用 NEXT_CONNECTION 函数调用遍历活动数据库连接，并检查与会话相关的属性（如 SessionID）即可找到会话数据。

以下 SQL 语句将说明如何跟踪活动会话：

```
CREATE VARIABLE conn_id LONG VARCHAR;
CREATE VARIABLE the_sessionID LONG VARCHAR;
SELECT NEXT_CONNECTION( NULL, NULL ) INTO conn_id;
conn_loop:
    LOOP
        IF conn_id IS NULL THEN
            LEAVE conn_loop;
        END IF;
        SELECT CONNECTION_PROPERTY( 'SessionID', conn_id )
            INTO the_sessionID;
        IF the_sessionID != '' THEN
            PRINT 'conn_id = %1!, SessionID = %2!', conn_id,
the_sessionID;
        ELSE
            PRINT 'conn_id = %1!', conn_id;
        END IF;
        SELECT NEXT_CONNECTION( conn_id, NULL ) INTO conn_id;
    END LOOP conn_loop;
    PRINT '\n';
```

如果查看数据库服务器消息窗口，会看到类似以下输出的数据：

```
conn_id = 30
conn_id = 29, SessionID = session_63315442223323
conn_id = 28, SessionID = session_63315442220088
conn_id = 25, SessionID = session_63315441867629
```

显式删除属于某会话的连接会导致连接关闭并删除该会话。如果要被删除的连接当前正用于为 HTTP 请求提供服务，则将对该请求设置删除标记，并向连接发送一个取消信号以终止该请求。当请求终止时，会话即被删除，并且连接关闭。删除该会话会导致该会话队列上的待执行请求进行重新排队。

如果连接当前处于非活动状态，则对会话设置删除标记，并将其重新排队为会话超时队列的起点。在下一个超时周期中（通常在 5 秒内）将删除该会话和连接。新 HTTP 请求不可以使用具有删除标记的任何会话。

如果数据库停止，所有的会话都将丢失。

HTTP 会话错误代码
如果新请求尝试访问某会话而该会话上的待执行请求超过 16 个，或会话排队时出错，则会出现错误 `503 Service Unavailable`。

如果客户端 IP 地址或主机名与会话创建者的 IP 地址或主机名不匹配，则会发生错误 `403 Forbidden`。

如果请求规定的会话不存在，则该请求不会隐式生成错误。由 Web 应用程序负责检测此条件（通过检查 SessionID、SessionCreateTime 或 SessionLastTime 连接属性）并执行相应操作。

字符集转换注意事项
缺省情况下，将对文本类型的外发结果集自动执行字符集转换。其它类型的结果集(如二进制对象）不会受到影响。请求的字符集将转换为 HTTP Web 服务器的字符集，结果集则被转换为客户端应用程序的字符集。如果请求列出了多个字符集，服务器将使用的列表上第一个适合的字符集。

可通过设置 sa_set_http_option 系统过程的 HTTP 选项 "CharsetConversion" 来启用或禁用字符集转换。

下面的示例将说明如何关闭自动字符集转换：

```
CALL sa_set_http_option('CharsetConversion', 'OFF');
```

如果启用了字符集转换，可以使用 sa_set_http_option 系统过程的 "AcceptCharset" 选项来指定字符集编码首选项。

下面的示例将说明如何在支持 ISO-8859-5 的情况下将其指定为 Web 服务字符集编码的首选项，而在不支持的时候将首选项指定为 UTF-8：

```
CALL sa_set_http_option('AcceptCharset', 'iso-8859-5, utf-8');
```

虽然在字符集的选择上优先考虑服务器的首选项，但也会考虑客户端的 Accept-Charset 条件。当客户端的字符集也在该选项中时，最优先的是客户端的字符集。

跨站点脚本注意事项
在开发 Web 应用程序时，应确保其不易受跨站点脚本 (XSS) 攻击。当攻击者试图将脚本注入您的 Web 页时，会发生此种类型的攻击。

强烈建议应用程序开发人员和数据库管理员在其 Web 应用程序代码投入生产前，查看该代码是否存在潜在的安全漏洞。Open Web Application Security Project (https://www.owasp.org) 中包含有关如何保护 Web 应用程序安全的详细信息。

Web 服务系统过程
以下系统过程用于 Web 服务：

- sa_http_header_info 系统过程

sa_http_php_page 系统过程
sa_http_php_page_interpreted 系统过程
sa_http_variable_info 系统过程
sa_set_http_header 系统过程
sa_set_http_option 系统过程
sa_set_soap_header 系统过程

Web 服务函数

Web 服务函数帮助在 Web 服务内部处理 HTTP 和 SOAP 请求。

提供以下函数:

HTML_DECODE 函数 [Miscellaneous]
HTML_ENCODE 函数 [Miscellaneous]
HTTP_BODY 函数 [Web 服务]
HTTP_DECODE 函数 [Web 服务]
HTTP_ENCODE 函数 [Web 服务]
HTTP_HEADER 函数 [Web 服务]
HTTP_RESPONSE_HEADER 函数 [Web 服务]
HTTP_VARIABLE 函数 [Web 服务]
NEXT_HTTP_HEADER 函数 [Web 服务]
NEXT_HTTP_RESPONSE_HEADER 函数 [Web 服务]
NEXT_HTTP_VARIABLE 函数 [Web 服务]
NEXT_SOAP_HEADER 函数 [SOAP]
SOAP_HEADER 函数 [SOAP]

针对 Web 服务还提供了许多系统过程。

Web 服务连接属性

Web 服务器连接属性可以是通过 CONNECTION_PROPERTY 函数访问的数据库属性。

使用以下语法来将一个来自 HTTP 服务器的连接属性值存储到 SQL 函数或过程的本地变量中:

```
SELECT CONNECTION_PROPERTY('connection-property-name') INTO
variable_name;
```

下面列出了常用于 Web 服务应用程序的有用运行时 HTTP 请求连接属性:

- **HttpServiceName** – 返回 Web 应用程序的服务名起源。
- **AuthType** – 返回连接时所使用的验证类型。
- **ServerPort** – 返回数据库服务器的 TCP/IP 端口号或 0。
- **ClientNodeAddress** – 返回客户端/服务器连接中客户端的节点。

- **ServerNodeAddress** – 返回客户端/服务器连接中服务器的节点。
- **BytesReceived** – 返回在客户端/服务器通信期间所收到的字节数。

Web 服务选项

Web 服务选项控制着 HTTP 服务器行为的多个方面。

使用以下语法在 HTTP 服务器中设置公共选项：

```
SET TEMPORARY OPTION PUBLIC.http_session_timeout=100;
```

下面列出了常用于 HTTP 服务器应用程序配置的选项：

- **http_connection_pool_basesize** – 指定数据库连接的额定阈值大小。
- **http_connection_pool_timeout** – 指定未使用的连接可以在连接池中保留的最大持续时间。
- **http_session_timeout** – 指定 HTTP 会话在非活动期间持续时间（单位为分钟）的缺省超时期限。
- **request_timeout** – 控制单个请求可以运行的最长时间。
- **webservice_namespace_host** – 指定在 DISH 服务规范中用作 XML 命名空间的主机名。

如何浏览 SAP Sybase IQ HTTP Web 服务器

Web 服务的命名和设计定义了可用的 URL 名称。每个 Web 服务都提供了它自己的 Web 内容集。通常这种内容都是由数据库中的自定义函数和过程生成的，不过内容也可以由指定 SQL 语句的 URL 生成。

还可以换种方式，或两者结合，可以定义 `root` Web 服务，它可以处理所有无指定服务处理的 HTTP 请求。`root` Web 服务通常会检查请求的 URL 和标头来决定如何处理该请求。

资源，如通过 HTTP 或安全 HTTPS 请求可用的 html 内容，由 URL 唯一地指定。本节将说明如何在 Web 浏览器中设置 URL 语法的格式，以便访问 SAP Sybase IQ HTTP Web 服务器上定义的 Web 服务。

注意： 本节中的信息适用于使用通用 HTTP Web 服务类型（如 RAW、XML 和 HTML）以及使用 DISH 服务的 HTTP Web 服务器。不能使用浏览器来发出 SOAP 请求。JSON 服务返回结果集，供使用 AJAX 的 Web 服务应用程序使用。

语法

```
{http|https}://host-name[:port-number][/dbn]/service-name[/path-name|?url-query]
```

参数

- **host-name 和 port-number** – 指定 Web 服务器的位置，而在端口号未定义为缺省 HTTP 或 HTTPS 端口号时也可以指定端口号，但不是必须的。*host-name* 可以是运行

Web 服务器的计算机 IP 地址。*port-number* 必须与启动 Web 服务器时使用的端口号匹配。

- **dbn** – 指定数据库名称。此数据库必须在 Web 服务器上运行，并且含有 Web 服务。

  如果 Web 服务器只运行一个数据库，或者如果为协议选项的给定 HTTP/HTTPS 监听者指定了数据库名称，则不需要指定 *dbn*。
- **service-name** – 指定要访问的 Web 服务的名称。此 Web 服务必须存在于 *dbn* 指定的数据库中。创建或变更 Web 服务时允许使用斜线字符 (/)，因此它们可以用在 *service-name* 中。SAP Sybase IQ 会将 URL 的其余部分与定义的服务进行匹配。

  如果未指定 *service-name* 而定义了 `root` Web 服务，客户端请求将被处理。如果服务器无法标识适用的服务来处理请求，将会返回 `404 Not Found` 错误。副作用是，如果 `root` Web 服务不存在而无法根据 URL 条件来处理请求，它将负责生成 `404 Not Found` 错误。
- **path-name** – 解析服务名称后，余下的斜线隔开的路径可以用 Web 服务过程访问。如果服务是用 URL ON 创建的，使用指定的 `URL` HTTP 变量就可以访问整个路径。如果服务是用 URL ELEMENTS 创建的，使用指定的 HTTP 变量 `URL1` 至 `URL10` 就可以访问每个路径元素。

  路径元素变量可以定义为服务语句定义的参数声明中的主机变量。作为上述方法的替代或补充，在存储过程内使用 HTTP_VARIABLE 函数调用也可以访问 HTTP 变量。

  下例将说明创建 Web 服务所用的 SQL 语句，其 URL 子句设置为 ELEMENTS：

```
CREATE SERVICE TestWebService
    TYPE 'HTML'
    URL ELEMENTS
    AUTHORIZATION OFF
    USER DBA
    AS CALL TestProcedure ( :url1, :url2 );
```

  此 `TestWebService` Web 服务调用一个过程，该过程显式地引用了 `url1` 与 `url2` 主机变量。

  假设 `TestWebService` 在 `localhost` 上的 `demo` 数据库上通过缺省端口运行，可以用以下 URL 来访问该 Web 服务：

```
http://localhost/demo/TestWebService/Assignment1/Assignment2/
Assignment3
```

  该 URL 可以访问 `TestWebService`，此服务运行 `TestProcedure`，将值 `Assignment1` 指派给 `url1`，并将值 `Assignment2` 指派给 `url2`。（可选步骤）`TestProcedure` 能通过 HTTP_VARIABLE 函数访问其它路径元素。例如，`HTTP_VARIABLE( 'url3' )` 函数调用返回 `Assignment3`。
- **url-query** – HTTP GET 请求可能会跟在路径后面，此路径带有指定 HTTP 变量的查询组件。类似地，使用标准 application/x-www-form-urlencoded Content-Type 的

POST 请求主体可以在请求主体内部传递 HTTP 变量。在这两种情况下，HTTP 变量都以名称/值对的形式传递，其中变量名用等号与其值隔开。变量用和号隔开。

HTTP 变量可以在服务语句的参数列表中作为主机变量显式声明，也可以在服务语句的存储过程中用 HTTP_VARIABLE 函数访问。

例如，以下 SQL 语句将创建需要两个主机变量的 Web 服务。主机变量用冒号 (:) 作为前缀来标识。

```
CREATE SERVICE ShowSalesOrderDetail
    TYPE 'HTML'
    URL OFF
    AUTHORIZATION OFF
    USER DBA
    AS CALL ShowSalesOrderDetail( :customer_id, :product_id );
```

假设 ShowSalesOrderDetail 在 localhost 上的演示数据库上通过缺省端口运行，可以用以下 URL 来访问该 Web 服务：

```
http://localhost/demo/ShowSalesOrderDetail?
customer_id=101&product_id=300
```

该 URL 可以访问 ShowSalesOrderDetail，将值 101 指派给 customer_id，并将值 300 指派给 product_id。输出结果将以 HTML 格式在 Web 浏览器中显示。

注释
要求连接到服务器时，Web 浏览器将提示输入用户名和口令。随后，浏览器 base64 在 Authorization 请求标头中对用户输入编码，并重新发送请求。

如果 Web 服务 URL 子句设置为 ON 或 ELEMENTS，*path-name* 和 *url-query* 的 URL 语法属性可同时使用，从而可使用几种不同格式设置选项的其中一个便能访问 Web 服务。如果同时使用这些语法属性，*path-name* 格式必须首先使用，后跟 *url-query* 格式。

在下例中，此 SQL 语句将创建一个 Web 服务，其中 URL 子句设置为 ON，定义了 url 变量：

```
CREATE SERVICE ShowSalesOrderDetail
    TYPE 'HTML'
    URL ON
    AUTHORIZATION OFF
    USER DBA
    AS CALL ShowSalesOrderDetail( :product_id, :url );
```

下面列出的是可接受的能够指派值 101 给 url 并指派值 300 给 product_id 的示例 URL：

```
http://localhost:80/demo/ShowSalesOrderDetail2/101?
product_id=300
http://localhost:80/demo/ShowSalesOrderDetail2?
url=101&product_id=300
```

```
http://localhost:80/demo/ShowSalesOrderDetail2?
product_id=300&url=101
```

如果在 *path-name* 和 *url-query* 的上下文中，一个主机变量名被多次指派，则最后的指派始终优先。例如，以下示例 URL 将指派值 101 给 `url`，并指派值 300 给 `product_id`:

```
http://localhost:80/demo/ShowSalesOrderDetail2/302?
url=101&product_id=300
http://localhost:80/demo/ShowSalesOrderDetail2/String?
product_id=300&url=101
```

示例

以下 URL 语法用于访问名为 `gallery_image` 的 Web 服务，该服务在本地 HTTP 服务器上名为 `demo` 的数据库中通过缺省端口运行。假设 `gallery_image` 服务用 `URL ON` 来定义:

```
http://localhost/demo/gallery_image/sunset.jpg
```

该 URL 看似要从传统 Web 服务器的目录中请求图形文件，但它却指定 `sunset.jpg` 为 HTTP Web 服务器的输入参数来访问 `gallery_image` 服务。

以下 SQL 语句将说明如何在 HTTP 服务器上定义 gallery 服务来实现此行为:

```
CREATE SERVICE gallery_image
    TYPE 'RAW'
    URL ON
    AUTHORIZATION OFF
    USER DBA
    AS CALL gallery_image ( :url );
```

`gallery_image` 服务调用一个同名的过程，传递客户端提供的 URL。有关实现可被此 Web 服务定义访问的 `gallery_image` 过程的示例，请参见 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP\gallery.sql`。

使用 Web 客户端访问 Web 服务

SAP Sybase IQ 可用作 Web 客户端来访问由 SAP Sybase IQ Web 服务器或第三方 Web 服务器（如 Apache 或 IIS）托管的 Web 服务。

除了将 SAP Sybase IQ 用作 Web 客户端，SAP Sybase IQ Web 服务还为客户端应用程序提供替代接口，来替代诸如 JDBC 和 ODBC 之类的传统接口。由于不需要额外的组件，它们很容易部署，还能通过多个平台上由多种语言（包括脚本编写语言如 Perl 和 Python）编写的客户端应用程序进行访问。

将 SAP Sybase IQ 用作 Web 客户端的快速入门

本节介绍如何将 SAP Sybase IQ 用作 Web 客户端应用程序来连接 SAP Sybase IQ HTTP 服务器和访问通用 HTTP Web 服务，其中并没有全面介绍 SAP Sybase IQ 的 Web 客

户端功能。虽然许多 SAP Sybase IQ Web 客户端功能都可用，但超出了本主题的范畴。

可以开发连接任何类型在线 Web 服务器的 SAP Sybase IQ Web 客户端应用程序，但本节假定在 8082 端口上启动了一个本地 SAP Sybase IQ HTTP 服务器，并打算连接由以下 SQL 语句创建的名为 SampleHTMLService 的 Web 服务：

```
CREATE SERVICE SampleHTMLService
    TYPE 'HTML'
    USER DBA
    AUTHORIZATION OFF
    AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
RESULT(ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR)
BEGIN
    SELECT i, f, s;
END;
```

执行以下任务创建 SAP Sybase IQ Web 客户端应用程序：

1. 如果尚未创建 SAP Sybase IQ 客户端数据库，运行以下命令来创建：

   ```
   iqinit -dba DBA,sql client-database-name
   ```

   将 *client-database-name* 替换为新客户端数据库的名称。
2. 运行以下命令启动客户端数据库：

   ```
   iqsrv16 client-database-name.db
   ```
3. 运行以下命令来通过 Interactive SQL 连接客户端数据库：

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=client-database-name"
   ```
4. 使用以下 SQL 语句来创建一个连接到 SampleHTMLService Web 服务的新客户端过程：

   ```
   CREATE PROCEDURE client_post(f REAL, i INTEGER, s VARCHAR(16), x
   VARCHAR(16))
       URL 'http://localhost:8082/SampleHTMLService'
       TYPE 'HTTP:POST'
       HEADER 'User-Agent:SATest';
   ```
5. 执行以下 SQL 语句来调用客户端过程，并向 Web 服务器发送 HTTP 请求：

   ```
   CALL client_post(3.14, 9, 's varchar', 'x varchar');
   ```

由 client_post 创建的 HTTP POST 请求与以下输出相似：

```
POST /SampleHTMLService HTTP/1.0
ASA-Id: ea1746b01cd0472eb4f0729948db60a2
User-Agent: SATest
Accept-Charset: windows-1252, UTF-8, *
Date: Wed, 9 Jun 2010 21:55:01 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded;
charset=windows-1252
Content-Length: 58

&f=3.1400001049041748&i=9&s=s%20varchar&x=x%20varchar
```

在 Web 服务器上运行的 Web 服务 SampleHTMLService 从 POST 请求中为 i、f 和 s 抽取参数值，并将它们作为参数传递至 sp_echo 过程。参数值 x 将被忽略。sp_echo 过程创建一个可返回至 Web 服务的结果集。客户端与 Web 服务器之间的参数名称的一致性对正确映射至关重要。

Web 服务会创建发回客户端的响应。Interactive SQL 显示的输出应当和以下输出类似：

属性	值
Status	HTTP /1.1 200 OK
Body	<html> <head> <title>/SampleHTMLService</title></head> <body> /SampleHTMLService <table border=1> <tr class="header"><th>ret_i</th> <th>ret_f</th> <th>ret_s</th> </tr> <tr><td>9</td><td>3.1400001049041748</td><td>s var-char</td>
Date	Wed, 09 Jun 2010 21:55:01 GMT
Connection	close
Expires	Wed, 09 Jun 2010 21:55:01 GMT
Content-Type	text/html; charset=windows-1252
Server	Sybase IQ/16.0.0

访问 SAP Sybase IQ HTTP Web 服务器的快速入门

本节介绍如何使用两种不同类型的客户端应用程序（Python 和 C#）来访问 SAP Sybase IQ HTTP Web 服务器，其中并没有全面介绍 SAP Sybase IQ 的 Web 服务应用程序功能。虽然许多 SAP Sybase IQ Web 服务功能都可用，但超出了本主题的范畴。

可以开发连接任何类型在线 Web 服务器的 SAP Sybase IQ Web 客户端应用程序，但本指南假定在 8082 端口上启动了一个本地 SAP Sybase IQ HTTP 服务器，并打算连接由以下 SQL 语句创建的名为 SampleXMLService 的 Web 服务：

```
CREATE SERVICE SampleXMLService
    TYPE 'XML'
    USER DBA
    AUTHORIZATION OFF
    AS CALL sp_echo2(:i, :f, :s);

CREATE PROCEDURE sp_echo2(i INTEGER, f NUMERIC(6,2), s LONG VARCHAR )
RESULT( ret_i INTEGER, ret_f NUMERIC(6,2), ret_s LONG VARCHAR )
```

```
BEGIN
    SELECT i, f, s;
END;
```

执行以下任务，使用 C# 或 Python 访问 XML Web 服务：

1. 创建连接 HTTP 服务器上的 Web 服务的过程。
   编写访问 `SampleXMLService` Web 服务的代码。
   - 对于 C#，使用以下代码：

```
using System;
using System.Xml;

public class WebClient
{
    static void Main(string[] args)
    {
        XmlTextReader reader = new XmlTextReader(
            "http://localhost:8082/SampleXMLService?
i=5&f=3.14&s=hello");
        while (reader.Read())
        {
            switch (reader.NodeType)
            {
                case XmlNodeType.Element:
                    if (reader.Name == "row")
                    {
                    Console.Write(reader.GetAttribute("ret_i")
+ " ");
                    Console.Write(reader.GetAttribute("ret_s")
+ " ");

Console.WriteLine(reader.GetAttribute("ret_f"));
                    }
                    break;
            }
        }
        reader.Close();
    }
}
```

   将此代码保存在名为 `DocHandler.cs` 的文件中。
   要编译程序，请在命令提示符下运行以下命令：

```
csc /out:DocHandler.exe DocHandler.cs
```

   - 对于 Python，使用以下代码：

```
import xml.sax

class DocHandler( xml.sax.ContentHandler ):
    def startElement( self, name, attrs ):
        if name == 'row':
            table_int = attrs.getValue( 'ret_i' )
            table_string = attrs.getValue( 'ret_s' )
            table_numeric = attrs.getValue( 'ret_f' )
            print('%s %s %s' % ( table_int, table_string,
table_numeric ))
```

```
parser = xml.sax.make_parser()
parser.setContentHandler( DocHandler() )
parser.parse('http://localhost:8082/SampleXMLService?
i=5&f=3.14&s=hello')
```

将此代码保存在名为 DocHandler.py 的文件中。

2. 在 HTTP 服务器发送的结果集上执行操作。
   - 对于 C#，运行以下命令：

     ```
     DocHandler
     ```
   - 对于 Python，运行以下命令：

     ```
     python DocHandler.py
     ```

应用程序显示以下输出：

```
5 hello 3.14
```

Web 客户端应用程序开发

SAP Sybase IQ 数据库可以用作 Web 客户端应用程序来访问 SAP Sybase IQ 或第三方 Web 服务器托管的 Web 服务。SAP Sybase IQ Web 客户端应用程序通过使用配置子句（例如指定 Web 服务目标端点的 URL 子句）编写存储过程和函数来创建。Web 客户端过程不包含主体，不过其它方面与任何存储过程都没有区别。Web 客户端过程在被调用时发出出站 HTTP 或 SOAP 请求。Web 客户端过程不能向自己发送出站 HTTP 请求；它不能调用在同一数据库上运行的本地主机 SAP Sybase IQ Web 服务。

有关 Web 服务应用程序的详细示例，请参见 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP 目录。

Web 客户端函数和过程的要求和建议

Web 服务客户端过程和函数要求定义 URL 子句来标识 Web 服务的端点。Web 服务客户端过程或函数除了对于配置有专用的子句，在其它方面同任何存储过程用法都相同。

可使用 CREATE PROCEDURE 和 CREATE FUNCTION 语句来创建 Web 客户端函数和过程，以向 Web 服务器发送 SOAP 或 HTTP 请求。

下表简要介绍了对于创建或变更 Web 客户端函数和过程的要求和建议。在创建或变更 Web 客户端函数或过程时可以指定以下信息：

- URL 子句，它需要一个绝对 URL 来指定 Web 服务端点。（必需）
- TYPE 子句，用于指定请求是 HTTP 还是 SOAP over HTTP。（建议使用）
- 可以被客户端应用程序访问的端口。（可选）
- HEADER 子句，用于指定 HTTP 请求的标头。（可选）
- SOAPHEADER 子句，用于指定 SOAP 请求封装中的 SOAP 标头条件。（可选。仅用于 SOAP 请求）
- 命名空间 URI。（仅用于 SOAP 请求）

Web 客户端 URL 子句

必须指定 Web 服务端点的位置，Web 客户端函数或过程才能访问它。CREATE PROCEDURE 和 CREATE FUNCTION 语句中的 URL 子句提供了要访问的 Web 服务 URL。

指定 HTTP 服务 URL

在 URL 子句中指定 HTTP 方案，将配置用于通过 HTTP 协议进行非安全通信的过程或函数。

以下语句演示了如何创建一个过程，该过程向名为 `SampleHTMLService` 的 Web 服务发送请求，而此服务位于由 `localhost` 的 HTTP Web 服务器在端口 8082 上托管的名为 `dbname` 的数据库中：

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
    URL 'http://localhost:8082/dbname/SampleHTMLService'
    TYPE 'HTTP:POST'
    HEADER 'User-Agent:SATest';
```

只有在 HTTP 服务器托管了多个数据库时，数据库名才是必需的。可以用 HTTP 服务器的主机名或 IP 地址替换 `localhost`。

指定 HTTPS 服务 URL

在 URL 子句中指定 HTTPS 方案，将配置用于通过安全套接字层（Secure Socket Layer，简称 SSL）进行安全通信的过程或函数。

Web 客户端应用程序必须有权访问 RSA 服务器证书或为服务器证书签名的证书，才能发布安全 HTTPS 请求。客户端过程需要证书来验证服务器，以防止中间人攻击。

使用 CREATE PROCEDURE 和 CREATE FUNCTION 语句中的 CERTIFICATE 子句可以验证服务器并建立安全数据通道。可以将证书放置在文件中并提供文件名，或者将整个证书作为字符串值提供，但只能选择其一。

以下语句演示了如何创建一个过程，该过程向名为 `SecureHTMLService` 的 Web 服务发送请求，而该服务位于由 `localhost` 的 HTTPS 服务器在端口 8082 上托管的名为 `dbname` 的数据库中：

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
    URL 'HTTPS://localhost:8082/dbname/SecureHTMLService'
    CERTIFICATE 'file=%ALLUSERSPROFILE%/SybaseIQ/demo\\Certificates\
\rsaroot.crt'
    TYPE 'HTTP:POST'
    HEADER 'User-Agent:SATest';
```

本例中 CERTIFICATE 子句表示 RSA 服务器证书位于 `%ALLUSERSPROFILE%\SybaseIQ\samples\Certificates\rsaroot.crt` 文件中。

注意： 指定 HTTPS_FIPS 会强制系统使用 FIPS 库。如果指定了 HTTPS_FIPS，但不存在 FIPS 库，则会改为使用非 FIPS 库。

指定代理服务器 URL
某些请求需要通过代理服务器来发送。使用 CREATE PROCEDURE 和 CREATE FUNCTION 语句中的 PROXY 子句来指定代理服务器 URL 并将请求重定向到该 URL。代理服务器会将请求转发到最终目标，获取响应，然后将响应转发回 SAP Sybase IQ。

Web 服务请求类型
在创建 Web 客户端函数或过程时，可以指定向 Web 服务器发送的客户端请求类型。CREATE PROCEDURE 和 CREATE FUNCTION 语句的 TYPE 子句在设置请求的格式后才将它们发送到 Web 服务器。

指定 HTTP 请求格式
如果 TYPE 子句中指定的格式以 HTTP 前缀开头，Web 客户端函数和过程将发送 HTTP 请求。

例如，在 Web 客户端数据库中执行以下 SQL 语句来创建名为 PostOperation 的 HTTP 过程，该过程向指定的 URL 发送 HTTP 请求：

```
CREATE PROCEDURE PostOperation(a INTEGER, b CHAR(128))
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    TYPE 'HTTP:POST';
```

在本例中，将请求的格式设置为 HTTP:POST 请求，使得生成的请求与下列内容类似：

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:02:49 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded;
charset=windows-1252
Content-Length: 12

a=123&b=data
```

指定 SOAP 请求格式
如果 TYPE 子句中指定的格式以 SOAP 前缀开头，Web 客户端函数和过程将发送 HTTP 请求。

例如，在 Web 客户端数据库中执行以下语句来创建名为 SoapOperation 的 SOAP 过程，该过程向指定的 URL 发送 SOAP 请求：

```
CREATE PROCEDURE SoapOperation(intVariable INTEGER, charVariable
CHAR(128))
    URL 'HTTP://localhost:8082/dbname/SampleSoapService'
    TYPE 'SOAP:DOC';
```

在本例中，调用此过程时向 URL 发送了一个 SOAP:DOC 请求，使得生成的请求与下列内容类似：

```
POST /dbname/SampleSoapService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:05:13 GMT
Host: localhost:8082
Connection: close
Content-Type: text/xml; charset=windows-1252
Content-Length: 428
SOAPAction: "HTTP://localhost:8082/SoapOperation"

<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="HTTP://localhost:8082">
  <SOAP-ENV:Body>
    <m:SoapOperation>
      <m:intVariable>123</m:intVariable>
      <m:charVariable>data</m:charVariable>
    </m:SoapOperation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

该过程名将出现在主体的 <m:SoapOperation> 标记中。该过程的两个参数 intVariable 和 charVariable 分别成为 <m:intVariable> 和 <m:charVariable>。

缺省情况下，在生成 SOAP 请求时，将存储过程名用作 SOAP 操作名。参数名出现在 SOAP 封装的标记名称中。定义 SOAP 存储过程时必须正确引用这些名称，因为服务器需要 SOAP 请求中带有这些名称。SET 子句可用于为给定的过程指定替代 SOAP 操作名称。WSDLC 用于从文件或 URL 指定中读取 WSDL 并生成 SQL 存根函数或过程。除非是最简单的情况（例如，返回单个字符串值的 SOAP RPC 调用），否则建议统一使用函数定义，而不使用过程。SOAP 函数返回完整的 SOAP 响应封装，这可以使用 OPENXML 进行解析。

Web 客户端端口

有时在通过防火墙打开一条服务器连接时，有必要指明要用的端口。可以使用 CREATE PROCEDURE 和 CREATE FUNCTION 语句的 CLIENTPORT 子句来指定客户端应用程序通过 TCP/IP 进行通信所用的端口号。除非防火墙限制了对特定范围端口的访问，否则建议不要使用此功能。

例如，在 Web 客户端数据库中执行以下语句来创建名为 SomeOperation 的过程，它使用 5050 - 5060 范围内的端口或 5070 端口来向指定的 URL 发送请求：

```
CREATE PROCEDURE SomeOperation()
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    CLIENTPORT '5050-5060,5070';
```

建议在必要时指定端口号的范围。如果只指定一个端口号，每次只能维持一个连接；客户端应用程序将尝试访问所有指定的端口号，直至找到要绑定的端口号。连接关闭后，会有一个长达几分钟的超时期，在此期间无法建立与同一服务器和端口的任何新连接。

此特性与设置 ClientPort 网络协议选项类似。

HTTP 请求标头管理

HTTP 请求标头可以用 CREATE PROCEDURE 和 CREATE FUNCTION 语句的 HEADER 子句来添加、更改或删除。通过引用名称来取消 HTTP 请求标头。可以通过在标头名称后加上冒号并紧跟值来添加或更改 HTTP 请求标头的值。指定标头值为可选操作。

例如，在 Web 客户端数据库中执行以下 SQL 语句来创建名为 `SomeOperation2` 的过程，该过程向指定的 URL（限制 HTTP 请求标头）发送请求：

```
CREATE PROCEDURE SomeOperation2()
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    TYPE 'HTTP:GET'
    HEADER 'SOAPAction\nDate\nFrom:\nCustomAlias:John Doe';
```

在本例中，由 SAP Sybase IQ 自动生成的标头 `Date` 被取消。`From` 标头存在，但没有指派任何值。HTTP 请求包含了名为 `CustomAlias` 的新标头，并指派了值 `John Doe`。GET 请求与下列内容类似：

```
GET /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SybaseIQ/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
From:
Host: localhost:8082
Connection: close
CustomAlias: John Doe
```

支持长标头值的折叠，只要在 `\n` 后紧跟一个或多个空格。

下面的示例演示了对长标头值的支持：

```
CREATE PROCEDURE SomeOperation3()
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    TYPE 'HTTP:POST'
    HEADER 'heading1: This long value\n is really long for a header.
\n
    heading2:shortvalue';
```

POST 请求与下列内容类似：

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SybaseIQ/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:26:04 GMT
heading1: This long value is really long for a header.
heading2:shortvalue
```

```
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded;
charset=windows-1252
Content-Length: 0
```

注意： 创建 SOAP 函数或过程时，必须将 `SOAPAction` HTTP 请求标头设置为在 WSDL 中指定的给定 SOAP 服务 URI。

自动生成的 HTTP 请求标头

修改自动生成的标头可能会导致意外的结果。在无预防措施的情况下不得对以下 HTTP 请求标头进行修改：

HTTP 标头	说明
Accept-Charset	始终自动生成。如果更改或删除该标头，可能导致意外的数据转换错误。
ASA-Id	始终自动生成。此标头确保客户端应用程序不会连接它自己形成死锁。
Authorization	当 URL 包含证书时自动生成。如果更改或删除此标头，可能导致请求失败。仅支持 BASIC 授权。仅当通过 HTTPS 进行连接时才能包括用户和口令信息。
Connection	Connection:close 始终自动生成。客户端应用程序不支持持久连接。连接如果更改将被挂起。
Host	始终自动生成。如果 HTTP/1.1 客户端未提供 Host 标头，则需要 HTTP/1.1 服务器响应 400 错误请求。
Transfer-Encoding	在块模式下发布请求时自动生成。当客户端使用 CHUNK 模式时，删除此标头或删除块值将导致失败。
Content-Length	当发布请求且不在块模式下时自动生成。要将主体的内容长度通知服务器，必须提供此标头。如果内容长度错误，则连接可能挂起或发生数据丢失。

SOAP 请求标头管理

SOAP 请求标头是 SOAP 封装中的一个 XML 片段。当 SOAP 操作及其参数可以被看作远程过程调用 (Remote Procedure Call，RPC) 时，SOAP 请求标头可用于在特定的请求或响应中传输元信息。SOAP 请求标头传输应用程序的元数据，如授权或会话标准。

SOAPHEADER 子句的值必须是有效并符合 SOAP 请求标头条目的 XML 片段。可以指定多个 SOAP 请求标头条目。存储过程或函数将 SOAP 请求标头条目自动注入 SOAP 标头元素 (`SOAP-ENV:Header`)。SOAPHEADER 值指定可声明为静态常量或使用参数替代机制进行动态设置的 SOAP 标头。以下是一个 SOAP 请求示例的代码段。它包含两个 XML 标头，分别称为 `Authentication` 和 `Session`。

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
```

```
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="HTTP://localhost:8082">
  <SOAP-ENV:Header>
    <Authentication xmlns="CustomerOrderURN">
      <userName pwd="none" mustUnderstand="1">
        <first>John</first>
        <last>Smith</last>
      </userName>
    </Authentication>
    <Session xmlns="SomeSession">123456789</Session>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:SoapOperation>
      <m:intVariable>123</m:intVariable>
      <m:charVariable>data</m:charVariable>
    </m:SoapOperation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

函数和过程在处理 SOAP 响应标头（由 SOAP 调用返回）上是不同的。使用函数是最灵活也是建议采用的方法，此时将接收到整个 SOAP 响应封装。响应封装就可以通过 OPENXML 运算符进行处理，从而抽取 SOAP 标头和 SOAP 主体数据。而使用过程时，SOAP 响应标头只能通过用映射到 IN 或 INOUT 变量的替代参数来抽取。SOAP 过程允许的 IN 或 INOUT 参数的最大数量为一个。

Web 服务函数必须分析响应 SOAP 封装以获得标头条目。

示例

以下示例说明了如何创建发送参数和 SOAP 标头的 SOAP 过程和函数。使用包装过程来填充 Web 服务过程调用，并处理响应。`soapAddItemProc` 过程说明了 SOAP Web 服务过程的使用，`soapAddItemFunc` 函数说明了 SOAP Web 服务函数的使用，而 `httpAddItemFunc` 函数则说明了 SOAP 载荷如何被传递到 HTTP Web 服务过程。

下例说明了使用替代参数发送 SOAP 标头的 SOAP 客户端过程。可使用单个 INOUT 参数来接收 SOAP 标头。调用 `soapAddItemProc` 的包装存储过程 `addItemProcWrapper` 演示了如何发送和接收包括参数的 SOAP 标头。

```
CREATE PROCEDURE soapAddItemProc(amount INT, item LONG VARCHAR,
        INOUT inoutheader LONG VARCHAR, IN inheader LONG VARCHAR)
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP( OP=addItems )'
    TYPE 'SOAP:DOC'
    SOAPHEADER '!inoutheader!inheader';

CREATE PROCEDURE addItemProcWrapper(amount INT, item LONG VARCHAR,
        first_name LONG VARCHAR, last_name LONG VARCHAR)
BEGIN
    DECLARE io_header LONG VARCHAR;     // inout (write/read) soap
header
    DECLARE resxml LONG VARCHAR;
    DECLARE soap_header_sent LONG VARCHAR;
```

```
   DECLARE i_header LONG VARCHAR;      // in (write) only soap header
    DECLARE err int;
    DECLARE crsr CURSOR FOR
        CALL soapAddItemProc( amount, item, io_header, i_header );

    SET io_header = XMLELEMENT( 'Authentication',
                        XMLATTRIBUTES('CustomerOrderURN' as xmlns),
                        XMLELEMENT('userName', XMLATTRIBUTES(
                                        'none' as pwd,
                                        '1' as mustUnderstand ),
                            XMLELEMENT( 'first', first_name ),
                            XMLELEMENT( 'last', last_name ) ) );
    SET i_header = '<Session xmlns="SomeSession">123456789</
Session>';
    SET soap_header_sent = io_header || i_header;
    OPEN crsr;
    FETCH crsr INTO resxml, err;
    CLOSE crsr;

    SELECT resxml, err, soap_header_sent, io_header AS
soap_header_received;
END;

/* example call to addItemProcWrapper */
CALL addItemProcWrapper( 5, 'shirt', 'John', 'Smith' );
```

下例说明了使用替代参数发送 SOAP 标头的 SOAP 客户端函数。返回整个 SOAP 响应封装。可使用 OPENXML 运算符分析 SOAP 标头。调用 `soapAddItemFunc` 的包装函数 `addItemFuncWrapper` 演示了如何发送和接收包括参数的 SOAP 标头。它也显示了如何使用 OPENXML 运算符处理响应。

```
CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR,
        IN inheader1 LONG VARCHAR, IN inheader2 LONG VARCHAR )
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP(OP=addItems)'
    TYPE 'SOAP:DOC'
    SOAPHEADER '!inheader1!inheader2';

CREATE PROCEDURE addItemFuncWrapper(amount INT, item LONG VARCHAR,
        first_name LONG VARCHAR, last_name LONG VARCHAR )
BEGIN
    DECLARE i_header1 LONG VARCHAR;
    DECLARE i_header2 LONG VARCHAR;
    DECLARE res LONG VARCHAR;
    DECLARE ns LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE header_entry LONG VARCHAR;
    DECLARE localname LONG VARCHAR;
    DECLARE namespaceuri LONG VARCHAR;
    DECLARE r_quantity int;
    DECLARE r_item LONG VARCHAR;
    DECLARE r_status LONG VARCHAR;

    SET i_header1 = XMLELEMENT( 'Authentication',
```

```
                        XMLATTRIBUTES('CustomerOrderURN' as xmlns),
                        XMLELEMENT('userName', XMLATTRIBUTES(
                                        'none' as pwd,
                                        '1' as mustUnderstand ),
                            XMLELEMENT( 'first', first_name ),
                            XMLELEMENT( 'last',  last_name ) ) );
    SET i_header2 = '<Session xmlns="SessionURN">123456789</
Session>';

    SET res = soapAddItemFunc( amount, item, i_header1, i_header2 );

    SET ns = '<ns xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/"'
            || ' xmlns:mp="urn:ianywhere-com:sa-xpath-metaprop"'
            || ' xmlns:customer="CustomerOrderURN"'
            || ' xmlns:session="SessionURN"'
            || ' xmlns:tns="http://localhost:8082"></ns>';

    // Process headers...
    SET xpath = '//SOAP-ENV:Header/*';
    BEGIN
         DECLARE crsr CURSOR FOR SELECT * FROM
                OPENXML( res, xpath, 1, ns )
                  WITH (  "header_entry" LONG VARCHAR '@mp:xmltext',
                             "localname"    LONG VARCHAR
'@mp:localname',
                             "namespaceuri" LONG VARCHAR
'@mp:namespaceuri' );
        OPEN crsr;
        FETCH crsr INTO "header_entry", "localname", "namespaceuri";
        CLOSE crsr;
    END;

    // Process body...
    SET xpath = '//tns:row';
    BEGIN
         DECLARE crsr1 CURSOR FOR SELECT * FROM
                OPENXML( res, xpath, 1, ns )
                    WITH (  "r_quantity" INT 'tns:quantity/text()',
                             "r_item"     LONG VARCHAR 'tns:item/
text()',
                             "r_status"   LONG VARCHAR 'tns:status/
text()' );
        OPEN crsr1;
        FETCH crsr1 INTO "r_quantity", "r_item", "r_status";
        CLOSE crsr1;
    END;

    SELECT r_item, r_quantity, r_status, header_entry, localname,
namespaceuri;
END;

/* example call to addItemFuncWrapper */
CALL addItemFuncWrapper( 6, 'shorts', 'Jack', 'Smith' );
```

下例演示了如何才能使用 HTTP:POST 传输整个 SOAP 载荷。此方法创建了传输 SOAP 载荷的 Web 服务 HTTP 过程，而不是 Web 服务客户端 SOAP 过程。包装过程 `addItemHttpWrapper` 调用 `httpAddItemFunc` 演示 POST 函数的使用方法。它显示了如何发送和接收包括参数的 SOAP 标头以及如何接收响应。

```
CREATE FUNCTION httpAddItemFunc(soapPayload XML)
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    TYPE 'HTTP:POST:text/xml'
    HEADER 'SOAPAction: "http://localhost:8082/addItems"';

CREATE PROCEDURE addItemHttpWrapper(amount INT, item LONG VARCHAR)
RESULT(response XML)
BEGIN
    DECLARE payload XML;
    DECLARE response XML;

    SET payload =
'<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>' || amount || '</m:amount>
      <m:item>' || item || '</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>';

    SET response = httpAddItemFunc( payload );
    /* process response as demonstrated in addItemFuncWrapper */
    SELECT response;
END;

/* example call to addItemHttpWrapper */
CALL addItemHttpWrapper( 7, 'socks' );
```

限制

服务器端 SOAP 服务目前无法定义输入和输出 SOAP 标头要求。因此，SOAP 标头元数据在 DISH 服务的 WSDL 输出中不可用。SOAP 客户端工具箱无法为 SAP Sybase IQ SOAP 服务端点自动生成 SOAP 标头接口。

不支持 Soap 标头错误。

SOAP 命名空间 URI 要求

命名空间 URI 为给定的 SOAP 操作指定用于组成 SOAP 请求封装的 XML 命名空间。如果命名空间 URI 未被定义，将使用 URL 子句中的域组件。

服务器端的 SOAP 处理器使用此 URI 来了解请求的消息主体中各种实体的名称。CREATE PROCEDURE 和 CREATE FUNCTION 语句的 NAMESPACE 子句指定命名空间 URI。

在过程调用成功之前可能需要指定命名空间 URI。本信息通常在公共 Web 服务器文档中有所说明，但也可以从 Web 服务器上可用的 WSDL 中获得要求的命名空间 URI。在尝试与 SAP Sybase IQ Web 服务器通信时，可以通过访问 DISH 服务来生成 WSDL。

通常，可以从 WSDL 文档开头的 `targetNamespace` 属性中复制 NAMESPACE，该属性在 `wsdl:definition` 元素中指定。包含尾随的 "/" 时请务必谨慎，因为它们非常重要。第二，检查给定 SOAP 操作的 `soapAction` 属性。它应当和下面将要说明的生成的 `SOAPAction` HTTP 标头相对应。

NAMESPACE 子句满足两个函数。它为 SOAP 封装的主体指定了命名空间，并且如果为过程指定了 TYPE 'SOAP:DOC'，它将用作 `SOAPAction` HTTP 标头的域组件。

以下示例说明了 NAMESPACE 子句的用法：

```
CREATE FUNCTION an_operation(a_parameter LONG VARCHAR)
    RETURNS LONG VARCHAR
    URL 'http://wsdl.domain.com/fictitious.asmx'
    TYPE 'SOAP:DOC'
    NAMESPACE 'http://wsdl.domain.com/'
```

在 Interactive SQL 中执行以下 SQL 语句：

```
SELECT an_operation('a_value');
```

此语句生成类似以下输出的 SOAP 请求：

```
POST /fictitious.asmx HTTP/1.0
SOAPAction: "http://wsdl.domain.com/an_operation"
Host: wsdl.domain.com
Content-Type: text/xml
Content-Length: 387
Connection: close

<?xml version="1.0"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:m="http://wsdl.domain.com/">
  <SOAP-ENV:Body>
   <m:an_operation>
    <m:a_parameter>a_value</m:a_parameter>
   </m:an_operation>
```

```
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

前缀 "m" 的命名空间设置为 `http://wsdl.domain.com/` 且 `SOAPAction` HTTP 标头为 SOAP 操作指定一个完全限定的 URL。

尾随斜线并非 SAP Sybase IQ 进行正确操作所必需，但会导致一个难以诊断的响应失效。不论尾随斜线是否存在，都会正确生成 `SOAPAction` HTTP 标头。

如果未指定 NAMESPACE，将使用 URL 子句的域组件作为 SOAP 主体的命名空间，如果过程为 TYPE 'SOAP:DOC'，它将用于生成 HTTP `SOAPAction` HTTP 标头。如果在上例中 NAMESPACE 子句被省略，将用 `http://wsdl.domain.com` 作为命名空间。细微差别在于不存在尾随斜线 "/"。该 SOAP 请求的其它任何方面，包括 `SOAPAction` HTTP 标头都将和上例相同。

NAMESPACE 子句用于为 SOAP 主体按照上面 SOAP:DOC 示例所述指定命名空间。然而，`SOAPAction` HTTP 标头生成时值为空：SOAPAction: ""

使用 SOAP:DOC 请求类型时，命名空间也用于组成 `SOAPAction` HTTP 标头。

Web 客户端 SQL 语句

以下 SQL 语句有助于开发 Web 客户端：

Web 客户端相关 SQL 语句	说明
CREATE FUNCTION 语句 [Web 服务]	创建发出 HTTP 或 SOAP over HTTP 请求的 Web 客户端函数。
ALTER FUNCTION 语句	修改函数。
CREATE PROCEDURE 语句 [Web 服务]	创建向 HTTP 服务器发出 HTTP 或 SOAP 请求的用户定义的 Web 客户端过程。
ALTER PROCEDURE 语句	修改过程。

为 Web 服务提供的变量

根据 Web 服务的类型，可以多种方式向 Web 服务提供变量。

Web 客户端应用程序可以通过以下任意手段向通用 HTTP Web 服务提供变量：

URL 的后缀
HTTP 请求的主体

将变量作为标准 SOAP 封装的一部分，可以将它们提供给 `SOAP` 服务类型。

在 URL 中向 Web 服务提供的变量

HTTP Web 服务器可使用 Web 浏览器管理 URL 中提供的变量。这些变量可用以下任意约定来表示：

- 将它们附加在 URL 的末尾，并用斜线 (/) 将每个参数值隔开，如以下示例所示：

```
http://localhost/database-name/param1/param2/param3
```

- 在 URL 参数表中显式地定义它们，如以下示例所示：

```
http://localhost/database-name/?
arg1=param1&arg2=param2&arg3=param3
```

- 附加在 URL 上并在参数表中定义它们，如以下示例所示：

```
http://localhost/database-name/param4/param5?
arg1=param1&arg2=param2&arg3=param3
```

Web 服务器对 URL 的解释取决于如何指定 Web 服务 URL 子句。

在 HTTP 请求主体中提供的变量

可在 Web 客户端函数或过程中的 TYPE 子句中指定 HTTP:POST 来在 HTTP 请求的主体中提供变量。

缺省情况下，TYPE HTTP:POST 使用 application/x-www-form-urlencoded MIME 类型。所有的参数都是 URL 编码形式，并在请求的主体内传递。（可选步骤）如果给出介质类型，请求 Content-Type 的标头将自动调整为适应给定的介质类型，且一个参数值也将上载到请求的主体中。

示例

以下示例假设一个名为 `XMLService` 的 Web 服务存在于 `localhost` Web 服务器上。设置 SAP Sybase IQ 客户端数据库，通过 Interactive SQL 连接到它，然后执行以下 SQL 语句：

```
CREATE PROCEDURE SendXMLContent(xmlcode LONG VARCHAR)
    URL 'http://localhost/XMLService'
    TYPE 'HTTP:POST:text/xml';
```

该语句创建了一个允许在 HTTP 请求主体中以 text/xml 格式发送变量的过程。

在 Interactive SQL 中执行以下 SQL 语句以向 `XMLService` Web 服务发送 HTTP 请求：

```
CALL SendXMLContent('<title>Hello World!</title>');
```

过程调用向 `xmlcode` 参数指派一个值并将它发送给 Web 服务。

在 SOAP 封装中提供的变量

可通过 Web 客户端函数或过程的 SET SOAP 选项在 SOAP 封装中提供变量，以设置 SOAP 操作。

以下代码说明了如何在 Web 客户端函数中设置 SOAP 操作：

```
CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR)
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP(OP=addItems)'
    TYPE 'SOAP:DOC';
```

在本示例中，`addItems` 是包含了 `amount` 和 `item` 值的 SOAP 操作，它将作为参数被传递到 `soapAddItemFunc` 函数。

可以运行以下示例脚本来发送请求：

```
SELECT soapAddItemFunc(5, 'shirt');
```

调用 soapAddItemFunc 函数产生类似以下示例的 SOAP 封装：

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>5</m:amount>
      <m:item>shirt</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

作为前一种方法的替代，可以创建自己的 SOAP 载荷，并将其发送到 HTTP 包装中的服务器。

SOAP 服务的变量必须作为标准 SOAP 请求的一部分提供。以其它方式提供的值将被忽略。

以下代码说明如何创建用于构建自定义 SOAP 封装的 HTTP 包装过程：

```
CREATE PROCEDURE addItemHttpWrapper(amount INT, item LONG VARCHAR)
RESULT(response XML)
BEGIN
    DECLARE payload XML;
    DECLARE response XML;

    SET payload =
'<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>' || amount || '</m:amount>
      <m:item>' || item || '</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>';

    SET response = httpAddItemFunc( payload );
    /* process response as demonstrated in addItemFuncWrapper */
    SELECT response;
END;
```

以下代码说明了用于发送请求的 Web 客户端函数：

```
CREATE FUNCTION httpAddItemFunc(soapPayload XML)
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    TYPE 'HTTP:POST:text/xml'
    HEADER 'SOAPAction: "http://localhost:8082/addItems"';
```

可以运行以下示例脚本来发送请求：

```
CALL addItemHttpWrapper( 7, 'socks' );
```

从结果集中访问的变量

可以使用存储函数或过程进行 Web 服务客户端调用。如果从函数进行调用，则返回类型必须是字符数据类型，如 CHAR、VARCHAR 或 LONG VARCHAR。返回值是 HTTP 响应的主体。不包含任何标头信息。关于请求的附加信息（包括 HTTP 状态信息）由过程返回。这样，在需要访问附加信息时最好使用过程。

SOAP 过程

来自 SOAP 函数的响应是一个包含 SOAP 响应的 XML 文档。

SOAP 响应是结构化的 XML 文档，因此 SAP Sybase IQ 在缺省情况下会尝试利用此信息，并构造一个更实用的结果集。返回的响应文档中的每个顶级标记都将被抽取并用作列名。每个此类标记下的子树的内容将用作该列的对应行值。

例如，假设 SOAP 响应如下所示，SAP Sybase IQ 将构造出所显示的数据集：

```
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
     <ElizaResponse xmlns:SOAPSDK4="SoapInterop">
        <Eliza>Hi, I'm Eliza. Nice to meet you.</Eliza>
     <ElizaResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Eliza
Hi, I'm Eliza.Nice to meet you.

在本示例中，响应文档使用 <SOAP-ENV:Body> 标记中出现的 <ElizaResponse> 标记进行分隔。

结果集的列数与顶级标记数量相同。因为 SOAP 响应中只有一个顶级标记，所以此结果集只有一列。此单个顶级标记 Eliza 将成为列名。

XML 处理工具

使用 OPENXML 过程也可以访问 XML 结果集中的信息（包括 SOAP 响应）。

以下示例使用 OPENXML 过程来抽取 SOAP 响应的各部分。此示例使用 Web 服务将 SYSWEBSERVICE 表的内容公开为 SOAP 服务：

```
CREATE SERVICE get_webservices
    TYPE 'SOAP'
    AUTHORIZATION OFF
    USER DBA
    AS SELECT * FROM SYSWEBSERVICE;
```

以下 Web 客户端函数（必须在另一个 SAP Sybase IQ 数据库中创建）发出对此 Web 服务的调用。该函数的返回值是整个 SOAP 响应文档。响应采用 .NET `DataSet` 格式，因为 `DNET` 是缺省的 SOAP 服务格式。

```
CREATE FUNCTION get_webservices()
    RETURNS LONG VARCHAR
    URL 'HTTP://localhost/get_webservices'
    TYPE 'SOAP:DOC';
```

以下语句演示了如何使用 OPENXML 过程抽取结果集的两列。`service_name` 和 `secure_required` 列分别指明哪些 SOAP 服务是安全的以及何处需要 HTTPS。

```
SELECT *
FROM OPENXML( get_webservices(), '//row' )
WITH ("Name"    CHAR(128) 'service_name',
      "Secure?" CHAR(1)   'secure_required' );
```

此语句通过选择 `row` 节点的子节点生效。WITH 子句基于相关的两个元素构造结果集。假定只存在 `get_webservices` Web 服务，此函数将返回以下结果集：

Name	Secure?
get_webservices	N

通过 Web 服务检索结果集

HTTP 类型的 Web 服务过程在一个两列式结果集中返回关于响应的所有信息。此结果集包含响应状态、标头信息和主体。第一列名为 `Attribute`，第二列名为 `Value`。两列的数据类型都是 LONG VARCHAR。

结果集为每个响应标头字段都提供一个数据行，并且分别为 HTTP 状态行（Status 属性）和响应主体（Body 属性）提供了一个数据行。

以下示例代表一个典型响应：

属性	值
Status	HTTP /1.0 200 OK
Body	<!DOCTYPE HTML ...><HTML> ...</HTML>
Content-Type	text/html
Server	GWS/2.1
Content-Length	2234
Date	Mon, 18 Oct 2004, 16:00:00 GMT

创建以下 Web 服务存储过程，作为示例。

```
CREATE OR REPLACE PROCEDURE SybaseWebPage()
URL 'http://www.sybase.com/mobilize'
TYPE 'HTTP';
```

执行以下 SELECT 查询，从 Web 服务获取结果集形式的响应。

```
SELECT * FROM SybaseWebPage()
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR);
```

由于 Web 服务过程不会描述结果集的形状，因此，需要使用 WITH 子句定义临时视图。

查询结果可存储在表中。执行以下 SQL 语句，创建用于存储结果集值的表。

```
CREATE TABLE StoredResults(
     Attribute LONG VARCHAR,
     Value     LONG VARCHAR
);
```

可将结果集按以下所示插入到 StoredResults 表中：

```
INSERT INTO StoredResults
    SELECT * FROM SybaseWebPage()
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR);
```

可以根据 SELECT 语句的通常语法添加子句。例如，如果只需要结果集中的特定行，则可以添加 WHERE 子句将 SELECT 的结果限定为仅一行。

```
SELECT * FROM SybaseWebPage()
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR)
    WHERE Attribute = 'Status';
```

此 SELECT 语句仅从结果集中检索状态信息。它可用于验证调用是否成功。

SOAP 数据类型

缺省情况下，参数输入的 XML 编码为字符串，SOAP 服务格式的结果集输出中不包含明确描述结果集中列的数据类型的信息。对于所有格式，参数数据类型都为字符串。对于 DNET 格式，在响应的模式部分中所有列的数据类型均设置为字符串。CONCRETE 和 XML 格式不包含响应中的数据类型信息。可使用 DATATYPE 子句处理此缺省行为。

SAP Sybase IQ 允许使用 DATATYPE 子句设置数据类型。数据类型信息包含在参数输入和结果集输出或所有 SOAP 服务格式的响应的 XML 编码中。这样一来便不再需要客户端代码将参数显式转换为字符串，从而简化了从 SOAP 工具箱进行参数传递。例如，可将整数作为整型进行传递。XML 编码的数据类型允许 SOAP 工具箱分析数据并将其归到相应类型。

以独占方式使用字符串数据类型时，应用程序需要隐式地了解结果集内每列的数据类型。当 Web 服务器请求数据类型时，无需执行此操作。在定义 Web 服务时，可使用 DATATYPE 子句控制是否包括数据类型信息。

下面是征用结果集响应数据类型的 Web 服务定义的示例。

```
CREATE SERVICE "SASoapTest/EmployeeList"
    TYPE 'SOAP'
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA
    DATATYPE OUT
    AS SELECT * FROM Employees;
```

在此示例中，由于此服务没有参数，因此仅为结果集响应请求数据类型信息。

数据类型适用于所有定义为 'SOAP' 类型的 SAP Sybase IQ Web 服务。

输入参数的数据类型
只要将参数数据类型公开为其在 WSDL 中由 DISH 服务生成的真实数据类型就可支持输入参数的数据类型。

典型字符串参数定义（或非类型参数）应如下所示：

```
<s:element minOccurs="0" maxOccurs="1" name="a_varchar"
nillable="true" type="s:string" />
```

字符串参数可以为空，即它可以发生，也可以不发生。

对于确定类型的参数（例如整数），必须有该参数且不能为空。以下是一个示例。

```
<s:element minOccurs="1" maxOccurs="1" name="an_int"
nillable="false" type="s:int" />
```

输出参数的数据类型
所有 'SOAP' 类型的 SAP Sybase IQ Web 服务都可公开响应数据内的数据类型信息。将数据类型公开为行集列元素内的属性。

以下是一个来自 SOAP FORMAT 'CONCRETE' Web 服务的 SimpleDataSet 类型响应的示例。

```
<SOAP-ENV:Body>
  <tns:test_types_concrete_onResponse>
   <tns:test_types_concrete_onResult xsi:type='tns:SimpleDataset'>
    <tns:rowset>
      <tns:row>
       <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
       <tns:i xsi:type="xsd:int">99</tns:i>
       <tns:ii xsi:type="xsd:long">99999999</tns:ii>
       <tns:f xsi:type="xsd:float">3.25</tns:f>
       <tns:d xsi:type="xsd:double">.55555555555555582</tns:d>
       <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
       <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
      </tns:row>
     </tns:rowset>
   </tns:test_types_concrete_onResult>
  <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_concrete_onResponse>
</SOAP-ENV:Body>
```

以下是一个返回字符串格式 XML 数据的 SOAP FORMAT 'XML' Web 服务的响应示例。内部行集由编码的 XML 组成，为便于理解在此处以其解码形式显示。

```
<SOAP-ENV:Body>
 <tns:test_types_XML_onResponse>
  <tns:test_types_XML_onResult xsi:type='xsd:string'>
   <tns:rowset
     xmlns:tns="http://localhost/satest/dish"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tns:row>
     <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
     <tns:i xsi:type="xsd:int">99</tns:i>
     <tns:ii xsi:type="xsd:long">99999999</tns:ii>
     <tns:f xsi:type="xsd:float">3.25</tns:f>
     <tns:d xsi:type="xsd:double">.55555555555555582</tns:d>
     <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
     <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
    </tns:row>
   </tns:rowset>
  </tns:test_types_XML_onResult>
  <tns:sqlcode>0</tns:sqlcode>
 </tns:test_types_XML_onResponse>
</SOAP-ENV:Body>
```

除数据类型信息外，这些元素的命名空间和 XML 模式还提供 XML 分析程序进行后处理所必需的全部信息。如果结果集中不存在数据类型信息（DATATYPE OFF 或 IN），则忽略 xsi:type 和 XML 模式命名空间声明。

返回 SimpleDataSet 类型的 SOAP FORMAT 'DNET' Web 服务的示例如下：

```
<SOAP-ENV:Body>
  <tns:test_types_dnet_outResponse>
   <tns:test_types_dnet_outResult
xsi:type='sqlresultstream:SqlRowSet'>
    <xsd:schema id='Schema2'
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       xmlns:msdata='urn:schemas-microsoft.com:xml-msdata'>
     <xsd:element name='rowset' msdata:IsDataSet='true'>
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element name='row' minOccurs='0' maxOccurs='unbounded'>
         <xsd:complexType>
          <xsd:sequence>
          <xsd:element name='lvc' minOccurs='0' type='xsd:string' />
           <xsd:element name='ub' minOccurs='0'
type='xsd:unsignedByte' />
           <xsd:element name='s' minOccurs='0' type='xsd:short' />
           <xsd:element name='us' minOccurs='0'
type='xsd:unsignedShort' />
           <xsd:element name='i' minOccurs='0' type='xsd:int' />
           <xsd:element name='ui' minOccurs='0'
type='xsd:unsignedInt' />
           <xsd:element name='l' minOccurs='0' type='xsd:long' />
           <xsd:element name='ul' minOccurs='0'
type='xsd:unsignedLong' />
```

```
           <xsd:element name='f' minOccurs='0' type='xsd:float' />
           <xsd:element name='d' minOccurs='0' type='xsd:double' />
           <xsd:element name='bin' minOccurs='0'
type='xsd:base64Binary' />
           <xsd:element name='bool' minOccurs='0'
type='xsd:boolean' />
         <xsd:element name='num' minOccurs='0' type='xsd:decimal' />
          <xsd:element name='dc' minOccurs='0' type='xsd:decimal' />
           <xsd:element name='date' minOccurs='0' type='xsd:date' />
          </xsd:sequence>
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
    </xsd:schema>
    <diffgr:diffgram xmlns:msdata='urn:schemas-microsoft-com:xml-
msdata' xmlns:diffgr='urn:schemas-microsoft-com:xml-diffgram-v1'>
     <rowset>
      <row>
       <lvc>Hello World</lvc>
       <ub>128</ub>
       <s>-99</s>
       <us>33000</us>
       <i>-2147483640</i>
       <ui>4294967295</ui>
       <l>-9223372036854775807</l>
       <ul>18446744073709551615</ul>
       <f>3.25</f>
       <d>.5555555555555582</d>
       <bin>QUJD</bin>
       <bool>1</bool>
       <num>123456.123457</num>
       <dc>-1.756000</dc>
       <date>2006-05-29-04:00</date>
      </row>
     </rowset>
    </diffgr:diffgram>
   </tns:test_types_dnet_outResult>
  <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_dnet_outResponse>
</SOAP-ENV:Body>
```

将 SAP Sybase IQ 类型映射到 XML 模式类型

SAP Sybase IQ 类型	XML 模式类型	XML 示例
CHAR	string	Hello World
VARCHAR	string	Hello World
LONG VARCHAR	string	Hello World
TEXT	string	Hello World

SAP Sybase IQ 类型	XML 模式类型	XML 示例
NCHAR	string	Hello World
NVARCHAR	string	Hello World
LONG NVARCHAR	string	Hello World
NTEXT	string	Hello World
UNIQUEIDENTIFIER	string	12345678-1234-5678-9012-123456789012
UNIQUEIDENTI-FIERSTR	string	12345678-1234-5678-9012-123456789012
XML	此项由用户定义。假定参数是表示复杂类型（例如 base64Binary、SOAP 数组、struct）的有效 XML。	<inputHexBinary xsi:type="xsd:hexBinary"> 414243 </inputHexBinary> （解释为 'ABC'）
BIGINT	long	-9223372036854775807
UNSIGNED BIGINT	unsignedLong	18446744073709551615
BIT	boolean	1
VARBIT	string	11111111
LONG VARBIT	string	00000000000000001000000000000000
DECIMAL	decimal	-1.756000
DOUBLE	double	.555555555555555582
FLOAT	float	12.3456792831420898
INTEGER	int	-2147483640
UNSIGNED INTEGER	unsignedInt	4294967295
NUMERIC	decimal	123456.123457
REAL	float	3.25
SMALLINT	short	-99
UNSIGNED SMALLINT	unsignedShort	33000
TINYINT	unsignedByte	128

SAP Sybase IQ 类型	XML 模式类型	XML 示例
MONEY	decimal	12345678.9900
SMALLMONEY	decimal	12.3400
DATE	date	2006-11-21-05:00
DATETIME	dateTime	2006-05-21T09:00:00.000-08:00
SMALLDATETIME	dateTime	2007-01-15T09:00:00.000-08:00
TIME	time	14:14:48.980-05:00
TIMESTAMP	dateTime	2007-01-12T21:02:14.420-06:00
TIMESTAMP WITH TIME ZONE	dateTime	2007-01-12T21:02:14.420-06:00
BINARY	base64Binary	AAAAZg==
IMAGE	base64Binary	AAAAZg==
LONG BINARY	base64Binary	AAAAZg==
VARBINARY	base64Binary	AAAAZg==

如果一个或多个参数属于 NCHAR、NVARCHAR、LONG NVARCHAR 或 NTEXT 类型，则响应输出采用 UTF8 格式。如果客户端数据库使用 UTF-8 字符编码，则不存在行为的变化（因为 NCHAR 和 CHAR 数据类型是相同的）。不过，如果数据库不使用 UTF-8 字符编码，则所有不属于 NCHAR 数据类型的参数都将转换为 UTF8。XML 声明编码和 Content-Type HTTP 标头的值将与使用的字符编码相对应。

将 XML 模式类型映射到 Java 类型

XML 模式类型	Java 数据类型
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double

XML 模式类型	Java 数据类型
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedInt	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration
xsd:NOTATION	javax.xml.namespace.QName

SOAP 结构化数据类型
使用函数或过程将 SAP Sybase IQ 服务器作为 Web 服务客户端与 Web 服务连接。

XML 返回值
对于简单的返回数据类型，结果集内的字符串表示形式就足够了。在这种情况下，可能有必要使用存储过程。

当返回复杂的数据（如数组或结构）时，使用 Web 服务函数是较好的选择。对于函数声明，RETURN 子句可指定 XML 数据类型。可使用 OPENXML 对返回的 XML 进行分析以抽取有用的元素。

返回的 XML 数据（如 dateTime）在结果集内逐字显示。例如，如果结果集内包括 TIMESTAMP 列，它的格式将为 XML dateTime 字符串 (2006-12-25T12:00:00.000-05:00) 而非字符串 (2006-12-25 12:00:00.000)。

XML 参数值

支持将 SAP Sybase IQ XML 数据类型用作 Web 服务函数和过程内的参数。对于简单类型，当生成 SOAP 请求主体时将自动构造参数元素。但对于 XML 类型的参数，将无法执行此操作，因为元素的 XML 表示形式可能需要提供其它数据的属性。因此，当为数据类型为 XML 的参数生成 XML 时，根元素名称必须与该参数名相对应。

```
<inputHexBinary xsi:type="xsd:hexBinary">414243</inputHexBinary>
```

该 XML 类型演示如何将参数作为 hexBinary XML 类型发送。SOAP 端点期望该参数名（在 XML 术语中称为根元素名称）为 "inputHexBinary"。

Cookbook 常量

构造复杂的结构和数组需要了解 SAP Sybase IQ 如何引用命名空间的知识。此处列出的前缀与为 SAP Sybase IQ SOAP 请求封装生成的命名空间的声明相对应。

SAP Sybase IQ XML 前缀	命名空间
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
SOAP-ENC	http://schemas.xmlsoap.org/soap/encoding/
m	NAMESPACE 子句中定义的命名空间

复杂数据类型示例

以下三个示例演示了如何使用表示数组、结构和结构数组的参数创建 Web 服务客户端函数。Web 服务函数将分别与名为 echoFloatArray、echoStruct 和 echoStructArray 的 SOAP 操作（或 RPC 函数名称）进行通信。用于互操作性测试的公共命名空间为 http://soapinterop.org/，这样，只需将 URL 子句更改为所选的 SOAP 端点，给定函数便可针对替代互操作性服务器进行测试。

该示例旨在向 Microsoft SOAP ToolKit 3.0 Round 2 互操作性测试服务器 (http://mssoapinterop.org/stkV3) 发布请求。

所有示例都使用一个表来生成 XML 数据。以下过程介绍如何设置该表。

```
CREATE LOCAL TEMPORARY TABLE SoapData
(
    seqno INT DEFAULT AUTOINCREMENT,
    i INT,
    f FLOAT,
    s LONG VARCHAR
) ON COMMIT PRESERVE ROWS;

INSERT INTO SoapData (i,f,s)
VALUES (99,99.999,'Ninety-Nine');

INSERT INTO SoapData (i,f,s)
VALUES (199,199.999,'Hundred and Ninety-Nine');
```

以下三个函数将 SOAP 请求发送到互操作性服务器。此示例向 Microsoft Interop 服务器发出请求：

```
CREATE FUNCTION echoFloatArray(inputFloatArray XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStruct(inputStruct XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStructArray(inputStructArray XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';
```

最终将显示三个示例语句及其参数的 XML 表示形式：

1. 以下示例中的参数表示一个数组。

```
SELECT echoFloatArray(
    XMLELEMENT( 'inputFloatArray',
       XMLATTRIBUTES( 'xsd:float[2]' as "SOAP-ENC:arrayType" ),
        (
        SELECT XMLAGG( XMLELEMENT( 'number', f ) ORDER BY seqno )
        FROM SoapData
        )
    )
);
```

存储过程 echoFloatArray 将以下 XML 发送到互操作性服务器。

```
<inputFloatArray SOAP-ENC:arrayType="xsd:float[2]">
<number>99.9990005493164</number>
<number>199.998992919922</number>
</inputFloatArray>
```

来自互操作性服务器的响应如下所示。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
    <SOAPSDK4:echoFloatArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result SOAPSDK3:arrayType="SOAPSDK1:float[2]"
        SOAPSDK3:offset="[0]"
        SOAPSDK2:type="SOAPSDK3:Array">
```

```
        <SOAPSDK3:float>99.9990005493164</SOAPSDK3:float>
        <SOAPSDK3:float>199.998992919922</SOAPSDK3:float>
      </Result>
    </SOAPSDK4:echoFloatArrayResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

如果该响应存储在变量中，则可使用 OPENXML 进行分析。

```
SELECT * FROM OPENXML( resp,'//*:Result/*' )
WITH ( varFloat FLOAT 'text()' );
```

varFloat
99.9990005493
199.9989929199

2. 以下示例中的参数表示一个结构。

```
SELECT echoStruct(
    XMLELEMENT('inputStruct',
        (
        SELECT XMLFOREST( s as varString,
                          i as varInt,
                          f as varFloat )
        FROM SoapData
        WHERE seqno=1
        )
    )
);
```

存储过程 echoStruct 将以下 XML 发送到互操作性服务器。

```
<inputStruct>
  <varString>Ninety-Nine</varString>
  <varInt>99</varInt>
  <varFloat>99.9990005493164</varFloat>
</inputStruct>
```

来自互操作性服务器的响应如下所示。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
    <SOAPSDK4:echoStructResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result href="#id1"/>
    </SOAPSDK4:echoStructResponse>
    <SOAPSDK5:SOAPStruct
      xmlns:SOAPSDK5="http://soapinterop.org/xsd"
      id="id1"
```

```
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK5:SOAPStruct">
      <varString>Ninety-Nine</varString>
      <varInt>99</varInt>
      <varFloat>99.9990005493164</varFloat>
    </SOAPSDK5:SOAPStruct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

如果该响应存储在变量中，则可使用 OPENXML 进行分析。

```
SELECT * FROM OPENXML( resp,'//*:Body/*:SOAPStruct' )
WITH (
varString LONG VARCHAR 'varString',
varInt INT 'varInt',
varFloat FLOAT 'varFloat' );
```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493

3. 以下示例中的参数表示一个结构数组。

```
SELECT echoStructArray(
    XMLELEMENT( 'inputStructArray',
       XMLATTRIBUTES( 'http://soapinterop.org/xsd' AS "xmlns:q2",
                      'q2:SOAPStruct[2]' AS "SOAP-ENC:arrayType" ),
            (
            SELECT XMLAGG(
                XMLElement('q2:SOAPStruct',
                XMLFOREST( s as varString,
                           i as varInt,
                           f as varFloat )
                )
            ORDER BY seqno
            )
         FROM SoapData
         )
    )
);
```

存储过程 echoFloatArray 将以下 XML 发送到互操作性服务器。

```
<inputStructArray xmlns:q2="http://soapinterop.org/xsd"
  SOAP-ENC:arrayType="q2:SOAPStruct[2]">
  <q2:SOAPStruct>
    <varString>Ninety-Nine</varString>
    <varInt>99</varInt>
    <varFloat>99.9990005493164</varFloat>
  </q2:SOAPStruct>
  <q2:SOAPStruct>
    <varString>Hundred and Ninety-Nine</varString>
    <varInt>199</varInt>
    <varFloat>199.998992919922</varFloat>
  </q2:SOAPStruct>
</inputStructArray>
```

来自互操作性服务器的响应如下所示。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
    <SOAPSDK4:echoStructArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result xmlns:SOAPSDK5="http://soapinterop.org/xsd"
        SOAPSDK3:arrayType="SOAPSDK5:SOAPStruct[2]"
        SOAPSDK3:offset="[0]" SOAPSDK2:type="SOAPSDK3:Array">
        <SOAPSDK5:SOAPStruct href="#id1"/>
        <SOAPSDK5:SOAPStruct href="#id2"/>
      </Result>
    </SOAPSDK4:echoStructArrayResponse>
    <SOAPSDK6:SOAPStruct
      xmlns:SOAPSDK6="http://soapinterop.org/xsd"
      id="id1"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK6:SOAPStruct">
      <varString>Ninety-Nine</varString>
      <varInt>99</varInt>
      <varFloat>99.9990005493164</varFloat>
    </SOAPSDK6:SOAPStruct>
    <SOAPSDK7:SOAPStruct
      xmlns:SOAPSDK7="http://soapinterop.org/xsd"
      id="id2"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK7:SOAPStruct">
      <varString>Hundred and Ninety-Nine</varString>
      <varInt>199</varInt>
      <varFloat>199.998992919922</varFloat>
    </SOAPSDK7:SOAPStruct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

如果该响应存储在变量中，则可使用 OPENXML 进行分析。

```
SELECT * FROM OPENXML( resp,'//*:Body/*:SOAPStruct' )
WITH (
varString LONG VARCHAR 'varString',
varInt INT 'varInt',
varFloat FLOAT 'varFloat' );
```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493
Hundred and Ninety-Nine	199	199.9989929199

用于子句值的替代参数

每当运行存储过程或函数时，存储过程或函数的已声明参数将自动替代子句定义中的占位符。替代参数允许创建可在运行时动态配置子句的通用 Web 服务过程。任何包含后面跟有一个已声明参数名称的感叹号 "!" 的子字符串都会被替换为该参数的值。这样，运行时可以替代一个或多个参数值来派生出一个或多个子句值。

参数替代要求遵守以下规则：

- 用于替代的所有参数必须是字母或数字。不允许有下划线。
- 替代参数后面必须紧跟非字母数字字符或终止字符。例如，`!sizeXL` 不能被名为 size 的参数的值替代，因为 X 属于字母数字字符。
- 与参数名称不匹配的替代参数将被忽略。
- 感叹号 (!) 可以用另一个感叹号进行转义。

例如，以下过程说明了参数替代的用法。URL 和 HTTP 标头的定义必须作为参数进行传递。

```
CREATE PROCEDURE test(uid CHAR(128), pwd CHAR(128), headers LONG
VARCHAR)
    URL 'http://!uid:!pwd@localhost/myservice'
    HEADER '!headers';
```

然后可以使用以下语句来调用 `test` 过程并发起一个 HTTP 请求：

```
CALL test('dba', 'sql', 'NewHeader1:value1\nNewHeader2:value2');
```

每次调用此过程时可使用不同的值。

加密证书示例

可使用参数替代传递来自某文件的加密证书并将它们传递到存储过程或存储函数。

下面的示例说明了如何将证书作为替代字符串传递：

```
CREATE PROCEDURE secure(cert LONG VARCHAR)
URL 'https://localhost/secure'
TYPE 'HTTP:GET'
CERTIFICATE 'cert=!cert;company=test;unit=test;name=RSA Root';
```

在以下调用中，从文件中读取证书并将其传递至 `secure`。

```
CALL secure( xp_read_file('%ALLUSERSPROFILE%/SybaseIQ/demo\
\Certificates\\rsaroot.crt) );
```

本示例仅作演示之用。使用 CERTIFICATE 子句的 `file=` 关键字可以直接从文件中读取证书。

没有匹配参数名的示例

没有匹配参数名的占位符将被自动删除。

例如，将不会用参数 size 替代以下过程中的占位符：

```
CREATE PROCEDURE orderitem (size CHAR(18))
    URL 'HTTP://localhost/salesserver/order?size=!sizeXL'
    TYPE 'SOAP:RPC';
```

在本例中，!sizeXL 会始终被删除，因为它是一个没有匹配参数的占位符。

调用存储函数或存储过程时，可使用参数替换该函数或过程主体中的占位符。如果不存在特定变量的占位符，参数及其值将作为请求的一部分传递。以这种方式用于替代的参数和值不会作为请求的一部分传递。

HTTP 和 SOAP 请求结构

除非在参数替代期间使用，否则函数或过程的所有参数都将作为 Web 服务请求的一部分传递。传递的格式取决于 Web 服务请求的类型。

不属于字符或二进制数据类型的参数值将在添加到请求之前先转换为字符串表示形式。此过程相当于将值设置为字符类型。该转换是在调用函数或过程时根据数据类型格式设置选项设置进行的。特别是，转换可能会受到 precision、scale 和 timestamp_format 之类选项的影响。

HTTP 请求结构

HTTP:GET 类型的参数进行了 URL 编码，位于 URL 内部。参数名称将被逐字用作 HTTP 变量的名称。例如，以下过程将声明两个参数：

```
CREATE PROCEDURE test(a INTEGER, b CHAR(128))
    URL 'HTTP://localhost/myservice'
    TYPE 'HTTP:GET';
```

如果此过程使用 123 和 'xyz' 这两个值调用，则用于该请求的 URL 如下所示：

```
HTTP://localhost/myservice?a=123&b=xyz
```

如果类型为 HTTP:POST，则参数及其相应值将进行 URL 编码并放置在请求主体中。针对这两个参数及其相应值，将在 HTTP 请求主体中的标头后显示以下文本：

```
a=123&b=xyz
```

SOAP 请求结构

按照 SOAP 规范的要求，传递给 SOAP 请求的参数将捆绑为请求主体的一部分：

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:test>
      <m:a>123</m:a>
      <m:b>abc</m:b>
    </m:test>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

如何记录 Web 客户端请求

可将包括 HTTP 请求和传输数据的 Web 服务客户端信息记录到 Web 服务客户端日志文件中。可使用 -zoc 服务器选项或使用 sa_server_option 系统过程指定 Web 服务客户端日志文件：

```
CALL sa_server_option( 'WebClientLogFile', 'clientinfo.txt' );
```

在指定 -zoc 服务器选项时自动启用记录。可以使用 sa_server_option 系统过程来允许和禁止记录到此文件：

```
CALL sa_server_option( 'WebClientLogging', 'ON' );
```

Web 服务参考

本节提供关于 Web 服务参考的信息。

Web 服务错误代码参考

当请求失败时，HTTP 服务器会生成标准 Web 服务错误。这些错误被指派了符合协议标准的编号。

下面列出了一些可能遇到的典型错误：

编号	名称	SOAP 故障	说明
301	永久移走	Server	请求的页面已永久移走。服务器会自动将请求重定向到新的位置。
304	未修改	Server	根据请求中的信息，服务器已确定自上次请求后没有修改过请求的数据，因此无需将其再次发送。
307	临时重定向	Server	请求的页面已移动，但此更改不一定是永久性的。服务器会自动将请求重定向到新的位置。
400	错误请求	Client.BadRequest	HTTP 请求不完整或者格式有误。
401	需要授权	Client.Authorization	使用该服务需要授权，但未提供有效的用户名和口令。
403	禁止	Client.Forbidden	您无权访问该数据库。
404	未找到	Client.NotFound	服务器上没有运行指定的数据库，或者指定的 Web 服务不存在。
408	请求超时	Server.RequestTime-out	接收请求时超出了连接空闲时间上限。

编号	名称	SOAP 故障	说明
411	必需的 HTTP 长度	Client.LengthRequired	服务器要求客户端在请求中指定 Content-Length。通常在向服务器上载数据时发生这种情况。
413	实体太大	Server	请求超出允许的大小上限。
414	URI 太大	Server	URI 的长度超出了允许的长度上限。
500	内部服务器错误	Server	出现内部错误。无法处理该请求。
501	未实现	Server	HTTP 请求方法不是 GET、HEAD 或 POST。
502	错误的网关	Server	所请求的文档位于第三方服务器上，并且服务器收到来自第三方服务器的错误。
503	服务不可用	Server	连接数超出允许上限。

如果 SOAP 服务失败，则这些故障消息以 SOAP 1.1 版标准中定义的 SOAP 故障的形式返回到客户端：

- 如果处理请求的应用程序中的错误生成 SQLCODE，则将返回 faultcode 为 Client（还可能带有子类别，如 Procedure）的 SOAP 故障。SOAP 故障中的 faultstring 元素将设置为详细的错误说明，并且 detail 元素中会包含 SQLCODE 数字值。
- 如果发生传输协议错误，faultcode 将设置为 Client 或 Server（具体取决于错误），faultstring 将设置为 HTTP 传输消息（如 `404 Not Found`），并且 detail 元素中包含数字型的 HTTP 错误值。
- 由于返回 SQLCODE 值的应用程序错误而生成的 SOAP 故障消息返回时带有 HTTP 状态 `200 OK`。

如果客户端无法被识别为 SOAP 客户端，相应的 HTTP 错误将返回到生成的 HTML 文档中。

HTTP Web 服务示例

关于 Web 服务的几个示例实现位于 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP` 目录下。有关示例的详细信息，请参见 `%ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\HTTP\readme.txt`。

教程：创建一个 Web 服务器并从 Web 客户端访问此 Web 服务器

本教程演示如何使用 SQL Anywhere 数据库服务器来创建 Web 服务器，以及如何从 Web 客户端数据库服务器向此 Web 服务器发送请求。

必需的软件

- SAP Sybase IQ

能力和经验

- 熟悉 XML
- 熟悉 MIME（多用途 Internet 邮件扩展）类型
- 了解 SAP Sybase IQ Web 服务的基本知识

目标

- 创建并启动新的 SAP Sybase IQ Web 服务器数据库。
- 创建 Web 服务。
- 设置返回 HTTP 请求所包含信息的过程。
- 创建并启动新的 SAP Sybase IQ Web 客户端数据库。
- 从 Web 客户端向数据库服务器发送 HTTP:POST 请求。
- 从 Web 服务器向 Web 客户端发送 HTTP 响应。

特权
需要具备以下特权才能执行本教程中的课程。

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

第 1 课：设置用于接收请求和发送响应的 Web 服务器

本课的目标是设置运行 Web 服务的 SAP Sybase IQ Web 服务器。

前提条件

本课假定您拥有在“教程：创建一个 Web 服务器并从 Web 客户端访问此 Web 服务器”教程开头的“特权”部分中列出的角色和特权。

过程

1. 创建将用于包含 Web 服务定义的 SAP Sybase IQ 数据库。
   ```
   iqinit -dba <user_id>,<password> echo
   ```
2. 使用此数据库启动网络数据库服务器。此服务器将充当 Web 服务器。
   ```
   iqsrv16 -xs http(port=8082) -n echo echo.db
   ```

将 HTTP Web 服务器设置为监听端口 8082 上的请求。如果网络禁用了 8082 端口，则使用其它端口号。

3. 使用 Interactive SQL 连接到数据库服务器。

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=echo"
```

4. 创建新的 Web 服务来接受进来的请求。

```
CREATE SERVICE EchoService
TYPE 'RAW'
USER DBA
AUTHORIZATION OFF
SECURE OFF
AS CALL Echo();
```

上述语句创建一个名为 `EchoService` 的新服务，该服务将在 Web 客户端向其发送请求时调用一个名为 `Echo` 的存储过程。它会为 Web 客户端生成没有任何格式设置 (RAW) 的 HTTP 响应正文。

5. 创建 `Echo` 过程以处理进来的请求。

```
CREATE OR REPLACE PROCEDURE Echo()
BEGIN
    DECLARE request_body LONG VARCHAR;
    DECLARE request_mimetype LONG VARCHAR;

    SET request_mimetype = http_header( 'Content-Type' );
    SET request_body = isnull( http_variable('text'),
http_variable('body') );
    IF request_body IS NULL THEN
        CALL sa_set_http_header('Content-Type', 'text/plain' );
        SELECT 'failed'
    ELSE
        CALL sa_set_http_header('Content-Type',
request_mimetype );
        SELECT request_body;
    END IF;
END
```

此过程将为发送到 Web 客户端的响应的 `Content-Type` 标头和主体设置格式。

Web 服务器已设置为接收请求和发送响应。

下一步

继续进行第 2 课：从 Web 客户端发送请求并接收响应。

第 2 课：从 Web 客户端发送请求并接收响应

在本课中，会将数据库客户端设置为使用 POST 方法向 Web 服务器发送请求并接收 Web 服务器的响应。

前提条件

本课假设您已经按照第 1 课中说明的方式设置了 Web 服务器。

本课假定您拥有在“教程：创建一个 Web 服务器并从 Web 客户端访问此 Web 服务器”教程开头的“特权”部分中列出的角色和特权。

过程

本课中包含对 `localhost` 的引用。如果 Web 客户端与 Web 服务器运行在不同的计算机上，则使用第 1 课中 Web 服务器的主机名或 IP 地址而不是 `localhost`。

1. 创建将用于包含 Web 客户端过程的 SAP Sybase IQ 数据库。

   ```
   iqinit -dba <user_id>,<password> echo_client
   ```

2. 使用此数据库启动网络数据库服务器。此服务器将充当 Web 客户端。

   ```
   iqsrv16 echo_client.db
   ```

3. 使用 Interactive SQL 连接到数据库服务器。

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=echo_client"
   ```

4. 创建新的存储过程以向 Web 服务发送请求。

   ```
   CREATE OR REPLACE PROCEDURE SendWithMimeType(
       value LONG VARCHAR,
       mimeType LONG VARCHAR,
       urlSpec LONG VARCHAR
   )
   URL '!urlSpec'
   TYPE 'HTTP:POST:!mimeType';
   ```

   `SendWithMimeType` 过程具有三个参数。`value` 参数表示应发送到 Web 服务的请求正文。`urlSpec` 参数表示用于连接到 Web 服务的 URL。`mimeType` 表示 HTTP:POST 所用的 MIME 类型。

5. 将请求发送到 Web 服务器并获得响应。

   ```
   CALL SendWithMimeType('<hello>this is xml</hello>',
       'text/xml',
       'http://localhost:8082/EchoService'
   );
   ```

   `http://localhost:8082/EchoService` 字符串表示 Web 服务器运行在 `localhost` 上并监听 8082 端口。所需的 Web 服务名为 `EchoService`。

6. 尝试其它 MIME 类型并观察响应。

   ```
   CALL SendWithMimeType('{"menu": {  "id": "file", "value": "File",
   "popup": {
       "menuitem": [{"value": "New", "onclick": "CreateNew()"},
                    {"value": "Open", "onclick": "Open()"},
                  {"value": "Close", "onclick": "Close()"} ] } } }',
       'application/json',
       'http://localhost:8082/EchoService'
   );
   ```

Web 客户端已设置为使用 POST 方法向 Web 服务器发送 HTTP 请求并接收 Web 服务器的响应。

教程：使用 SAP Sybase IQ 访问 SOAP/DISH 服务

本教程演示了如何创建用于将 Web 客户端提供的华氏温度值转换为摄氏温度的 SOAP 服务器。

必需的软件

- SAP Sybase IQ

能力和经验

- 熟悉 SOAP
- 了解 SAP Sybase IQ Web 服务的基本知识

目标

- 创建并启动新的 SAP Sybase IQ Web 服务器数据库。
- 创建 SOAP Web 服务。
- 设置将客户端提供的华氏温度值转换为摄氏温度值的过程。
- 创建并启动新的 SAP Sybase IQ Web 客户端数据库。
- 从 Web 客户端向数据库服务器发送 SOAP 请求。
- 从数据库服务器向 Web 客户端发送 SOAP 响应。

特权
需要具备以下特权才能执行本教程中的课程。

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

第 1 课：设置用于接收 SOAP 请求和发送 SOAP 响应的 Web 服务器

在本课中，您将设置新的数据库服务器并创建 SOAP 服务来处理进来的 SOAP 请求。服务器提出 SOAP 请求，该请求提供了将转换为等效摄氏温度的华氏温度值。

前提条件

本课假定您拥有在“教程：使用 SAP Sybase IQ 访问 SOAP/DISH 服务”教程开头的“特权”部分中列出的角色和特权。

过程

1. 创建将用于包含 Web 服务定义的 SAP Sybase IQ 数据库。

   ```
   iqinit -dba <user_id>,<password> ftc
   ```

2. 使用此数据库启动数据库服务器。此服务器将充当 Web 服务器。

   ```
   iqsrv16 -xs http(port=8082) -n ftc ftc.db
   ```

将 HTTP Web 服务器设置为监听端口 8082 上的请求。如果网络禁用了 8082 端口，则使用其它端口号。

3. 使用 Interactive SQL 连接到数据库服务器。

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=ftc"
```

4. 创建新的 DISH 服务来接受进来的请求。

```
CREATE SERVICE soap_endpoint
    TYPE 'DISH'
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA;
```

此语句创建名为 `soap_endpoint` 的新 DISH 服务，此服务处理进来的 SOAP 服务请求。

5. 创建新的 SOAP 服务来处理华氏温度到摄氏温度的转换。

```
CREATE SERVICE FtoCService
    TYPE 'SOAP'
    FORMAT 'XML'
    AUTHORIZATION OFF
    USER DBA
    AS CALL FToCConverter( :fahrenheit );
```

此语句创建名为 `FtoCService` 的新 SOAP 服务，此服务生成 XML 格式的字符串作为输出。当 Web 客户端向服务发送 SOAP 请求时，它调用名为 `FToCConverter` 的存储过程。

6. 创建 `FToCConverter` 过程以处理进来的 SOAP 请求。此过程执行必要的运算，将客户端提供的华氏温度值转换为等效的摄氏温度值。

```
CREATE OR REPLACE PROCEDURE FToCConverter( temperature FLOAT )
BEGIN
    DECLARE hd_key LONG VARCHAR;
    DECLARE hd_entry LONG VARCHAR;
    DECLARE alias LONG VARCHAR;
    DECLARE first_name LONG VARCHAR;
    DECLARE last_name LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE authinfo LONG VARCHAR;
    DECLARE namespace LONG VARCHAR;
    DECLARE mustUnderstand LONG VARCHAR;
header_loop:
    LOOP
        SET hd_key = NEXT_SOAP_HEADER( hd_key );
        IF hd_key IS NULL THEN
            -- no more header entries
            LEAVE header_loop;
        END IF;
        IF hd_key = 'Authentication' THEN
            SET hd_entry = SOAP_HEADER( hd_key );
            SET xpath = '/*:' || hd_key || '/*:userName';
           SET namespace = SOAP_HEADER( hd_key, 1, '@namespace' );
            SET mustUnderstand = SOAP_HEADER( hd_key, 1,
'mustUnderstand' );
```

```
            BEGIN
                -- parse the XML returned in the SOAP header
                DECLARE crsr CURSOR FOR
                    SELECT * FROM OPENXML( hd_entry, xpath )
                        WITH ( alias LONG VARCHAR '@*:alias',
                         first_name LONG VARCHAR '*:first/text()',
                         last_name LONG VARCHAR '*:last/text()' );
                OPEN crsr;
                FETCH crsr INTO alias, first_name, last_name;
                CLOSE crsr;
            END;

            -- build a response header
            -- based on the pieces from the request header
            SET authinfo =
                XMLELEMENT( 'Authentication',
                    XMLATTRIBUTES(
                        namespace as xmlns,
                        alias,
                        mustUnderstand ),
                        XMLELEMENT( 'first', first_name ),
                        XMLELEMENT( 'last', last_name ) );
            CALL SA_SET_SOAP_HEADER( 'authinfo', authinfo );
        END IF;
    END LOOP header_loop;
    SELECT ROUND((temperature - 32.0) * 5.0 / 9.0, 5) AS answer;
END;
```

NEXT_SOAP_HEADER 函数用于 LOOP 结构中遍历 SOAP 请求中的所有标头名，并在 NEXT_SOAP_HEADER 函数返回 NULL 时退出循环。

注意： 此函数不一定按照标头在 SOAP 请求中出现的顺序遍历它们。

SOAP_HEADER 函数在标头名不存在时返回标头值或 NULL。`FToCConverter` 过程搜索名为 `Authentication` 的标头，并抽取标头结构，包括 `@namespace` 和 `mustUnderstand` 属性。`@namespace` 标头属性为特殊 SAP Sybase IQ 属性，用于访问给定标头条目的命名空间 (`xmlns`)。

以下为可能的 `Authentication` 标头结构的 XML 字符串表示，其中 `@namespace` 属性值为 `"SecretAgent"`，且 `mustUnderstand` 值为 1：

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
    <userName alias="99">
        <first>Susan</first>
        <last>Hilton</last>
    </userName>
</Authentication>
```

SELECT 语句中的 OPENXML 系统过程解析使用 XPath 字符串 `"/*:Authentication/*:userName"` 的 XML 标头，从而抽取 `alias` 属性值以及 `first` 和 `last` 标记的内容。使用游标处理结果集以获取三个列值。

此时，您获取了传递给 Web 服务的所有相关信息。您获取了华氏温度和在 SOAP 标头中传递给 Web 服务的一些其它属性。你可以查找提供的名称和别名，查看此人是否有权使用 Web 服务。但是，示例中没有此练习。

SET 语句用于构建 XML 格式的 SOAP 响应以发送给客户端。以下是可能的 SOAP 响应的 XML 字符串表示。它基于上述 `Authentication` 标头结构示例。

```
<Authentication xmlns="SecretAgent" alias="99"
mustUnderstand="1">
  <first>Susan</first>
  <last>Hilton</last>
</Authentication>
```

SA_SET_SOAP_HEADER 系统过程用于设置将发送到客户端的 SOAP 响应标头。

最后的 SELECT 语句用来将提供的华氏温度值转换为摄氏温度值。信息将被中继回客户端。

此时，正在运行的 SQL Anywhere Web 服务器提供了从华氏温度转换为摄氏温度的服务。此服务处理来自客户端的 SOAP 标头并将 SOAP 响应发送回客户端。

下一步

在下一课中，将开发一个能向 Web 服务器发送 SOAP 请求并从 Web 服务器接收 SOAP 响应的示例客户端。

第 2 课：设置用于发送 SOAP 请求和接收 SOAP 响应的 Web 客户端

在本课中，您将设置用于发送 SOAP 请求和接收 SOAP 响应的 Web 客户端。

前提条件

本课假设您已经按照上一课设置了 Web 服务器。

本课假定您拥有在“教程：使用 SAP Sybase IQ 访问 SOAP/DISH 服务”教程开头的“特权”部分中列出的角色和特权。

过程

本课中包含对 `localhost` 的引用。如果 Web 客户端与 Web 服务器运行在不同的计算机上，则使用第 1 课中 Web 服务器的主机名或 IP 地址而不是 `localhost`。

1. 运行以下命令创建 SAP Sybase IQ 数据库：

   ```
   iqinit -dba <user_id>,<password> ftc_client
   ```
2. 使用以下命令启动数据库客户端：

   ```
   iqsrv16 ftc_client.db
   ```
3. 使用以下命令在 Interactive SQL 中连接数据库：

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=ftc_client"
   ```
4. 创建新的存储过程以向 DISH 服务发送 SOAP 请求。

在 Interactive SQL 中执行以下 SQL 语句：

```
CREATE OR REPLACE PROCEDURE FtoC( fahrenheit FLOAT,
    INOUT inoutheader LONG VARCHAR,
    IN inheader LONG VARCHAR )
  URL 'http://localhost:8082/soap_endpoint'
  SET 'SOAP(OP=FtoCService)'
  TYPE 'SOAP:DOC'
  SOAPHEADER '!inoutheader!inheader';
```

URL 子句中的 `http://localhost:8082/soap_endpoint` 字符串表示 Web 服务器在 `localhost` 上运行且监听 8082 端口。所需的 DISH Web 服务名为 `soap_endpoint`，它作为 SOAP 端点。

SET 子句指定要调用的 SOAP 操作名称或 `FtoCService` 服务。

创建 Web 服务请求时使用的缺省格式是 'SOAP:RPC'。本例中选择了 'SOAP:DOC' 格式，此格式与 'SOAP:RPC' 相似但允许更多数据类型。SOAP 请求始终以 XML 文档发送。SOAP 请求的发送机制是 'HTTP:POST'。

诸如 FtoC 的 Web 服务客户端过程中的替换变量（`inoutheader`、`inheader`）必须是字母数字字符。如果将 Web 服务客户端声明为函数，所有参数仅处于“IN”模式（无法由调用函数赋值）。因此，OPENXML 或其它字符串函数将必须用来提取 SOAP 响应标头信息。

5. 创建用于构建两个特殊 SOAP 请求标头条目的包装过程，将它们传递给 `FtoC` 过程，并处理服务器响应。

   在 Interactive SQL 中执行以下 SQL 语句：

```
CREATE OR REPLACE PROCEDURE FahrenheitToCelsius( Fahrenheit
FLOAT )
BEGIN
  DECLARE io_header LONG VARCHAR;
  DECLARE in_header LONG VARCHAR;
  DECLARE result LONG VARCHAR;
  DECLARE err INTEGER;
  DECLARE crsr CURSOR FOR
    CALL FtoC( Fahrenheit io_header, in_header );
  SET io_header =
    '<Authentication xmlns="SecretAgent" ' ||
    'mustUnderstand="1">' ||
    '<userName alias="99">' ||
    '<first>Susan</first><last>Hilton</last>' ||
    '</userName>' ||
    '</Authentication>';
  SET in_header =
    '<Session xmlns="SomeSession">' ||
    '123456789' ||
    '</Session>';

  MESSAGE 'send, soapheader=' || io_header || in_header;
  OPEN crsr;
  FETCH crsr INTO result, err;
  CLOSE crsr;
```

```
  MESSAGE 'receive, soapheader=' || io_header;
  SELECT Fahrenheit, Celsius
      FROM OPENXML(result, '//tns:answer', 1, result)
      WITH ("Celsius" FLOAT 'text()');
END;
```

第一个 SET 语句创建 SOAP 标头条目的 XML 表示，以向 Web 服务器通知用户证书：

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
  <userName alias="99">
    <first>Susan</first>
    <last>Hilton</last>
  </userName>
</Authentication>
```

第二个 SET 语句创建 SOAP 标头条目的 XML 表示，以跟踪客户端会话 ID：

```
<Session xmlns="SomeSession">123456789</Session>
```

6. OPEN 语句导致调用 FtoC 过程，此过程向 WEB 服务器发送 SOAP 请求并处理来自 Web 服务器的响应。响应包括在 `inoutheader` 中返回的标头。

此时，您已拥有一个可以向 Web 服务器发送 SOAP 请求并从 Web 服务器接收 SOAP 响应的客户端。

第 3 课：发送 SOAP 请求和接收 SOAP 响应

在本课中，将调用在上一课中创建的包装过程，它会向第一课中创建的 Web 服务器发送 SOAP 请求。

前提条件

本课假设您已经按照第 1 课中说明的方式设置了 Web 服务器。

本课假设您已经按照第 2 课中说明的方式设置了 Web 客户端。

本课假定您拥有在“教程：使用 SAP Sybase IQ 访问 SOAP/DISH 服务”教程开头的“特权”部分中列出的角色和特权。

过程

1. 如果在第二课中没有打开客户端数据库，则在 Interactive SQL 中连接它。

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=ftc_client"
   ```

2. 启用 SOAP 请求和响应的记录。

   在 Interactive SQL 中执行以下 SQL 语句：

   ```
   CALL sa_server_option('WebClientLogFile', 'soap.txt');
   CALL sa_server_option('WebClientLogging', 'ON');
   ```

   这些调用用于检查 SOAP 请求和响应的内容。请求和响应均记录在名为 `soap.txt` 的文件中。

3. 调用该包装过程以发送 SOAP 请求并接收 SOAP 响应。

   在 Interactive SQL 中执行以下 SQL 语句：

   ```
   CALL FahrenheitToCelsius(212);
   ```

   本次调用将华氏温度值 212 传递给 FahrenheitToCelsius 过程，而此过程将值和两个自定义的 SOAP 标头传递给 FToC 过程。两个客户端过程都在上一课中创建。

FToC Web 服务过程向 Web 服务器发送华氏温度值和 SOAP 标头。SOAP 请求包含如下内容。

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Header>
    <Authentication xmlns="SecretAgent" mustUnderstand="1">
      <userName alias="99">
        <first>Susan</first>
        <last>Hilton</last>
      </userName>
    </Authentication>
    <Session xmlns="SomeSession">123456789</Session>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:FtoCService>
      <m:fahrenheit>212</m:fahrenheit>
    </m:FtoCService>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

然后，FtoC 过程接收 Web 服务器的响应，此响应包括基于华氏温度值的结果集。SOAP 响应包含如下内容。

```
<SOAP-ENV:Envelope
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:tns='http://localhost:8082'>
  <SOAP-ENV:Header>
    <Authentication xmlns="SecretAgent" alias="99"
mustUnderstand="1">
      <first>Susan</first>
      <last>Hilton</last>
    </Authentication>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <tns:FtoCServiceResponse>
      <tns:FtoCServiceResult xsi:type='xsd:string'>
        &lt;tns:rowset xmlns:tns=&quot;http://localhost:8082/
ftc&quot;&gt;&#x0A;
        &lt;tns:row&gt;&#x0A;
        &lt;tns:answer&gt;100
        &lt;/tns:answer&gt;&#x0A;
```

```
        &lt;/tns:row&gt;&#x0A;
        &lt;/tns:rowset&gt;&#x0A;
      </tns:FtoCServiceResult>
      <tns:sqlcode>0</tns:sqlcode>
    </tns:FtoCServiceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

<SOAP-ENV:Header> 的内容在 inoutheader 中返回。

如果检查 SOAP 响应，则可以看出由 FToCService Web 服务在响应中对结果集编码。将结果集解码并返回给 FahrenheitToCelsius 过程。将华氏温度值 212 传递给 Web 服务器时，结果集类似于如下内容：

```
<tns:rowset xmlns:tns="http://localhost:8082/ftc">
  <tns:row>
    <tns:answer>100
    </tns:answer>
  </tns:row>
</tns:rowset>
```

FahrenheitToCelsius 过程中的 SELECT 语句使用 OPENXML 函数来解析 SOAP 响应，从中提取由 tns:answer 结构定义的摄氏温度值。

在 Interactive SQL 中生成以下结果集：

```
Fahrenheit   Celsius
       212       100
```

教程：使用 Visual C# 访问 SOAP/DISH Web 服务

本教程演示如何创建 Visual C# 客户端应用程序，访问 SAP Sybase IQ Web 服务器上的 SOAP/DISH 服务。

必需的软件

- SAP Sybase IQ
- Visual Studio

能力和经验

- 熟悉 SOAP
- 熟悉 .NET framework
- 了解 SQL Anywhere Web 服务的基本知识

目标

- 创建并启动新的 SAP Sybase IQ Web 服务器数据库。
- 创建 SOAP Web 服务。
- 设置返回 SOAP 请求所包含信息的过程。
- 创建提供 WSDL 文档并用作代理的 DISH Web 服务。

- 在客户端计算机上设置 Visual C# 并从 Web 服务器导入 WSDL 文档。
- 创建 Java 客户端应用程序，以使用 WSDL 文档中的信息从 SOAP 服务中检索信息。

特权
需要具备以下特权才能执行本教程中的课程。

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

第 1 课：设置用于接收 SOAP 请求和发送 SOAP 响应的 Web 服务器

在本课中，您将设置运行 SOAP 和 DISH Web 服务的 SAP Sybase IQ Web 服务器，处理 Visual C# 客户端应用程序的请求。

前提条件

需要最新版本的 Visual Studio。

本课假定您拥有在“教程：使用 Visual C# 访问 SOAP/DISH Web 服务”教程开头的“特权”部分中列出的角色和特权。

过程

1. 使用以下命令启动 SAP Sybase IQ 演示数据库：

   ```
   iqsrv16 -xs http(port=8082) iqdemo.db
   ```

   此命令表示 HTTP Web 服务器应当监听 8082 端口上的请求。如果网络禁用了 8082 端口，则使用其它端口号。
2. 使用以下命令来连接 Interactive SQL 中的数据库服务器：

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=demo"
   ```
3. 创建新的 SOAP 服务来接受进来的请求。

   在 Interactive SQL 中执行以下 SQL 语句：

   ```
   CREATE SERVICE "SASoapTest/EmployeeList"
       TYPE 'SOAP'
       DATATYPE ON
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA
       AS SELECT * FROM Employees;
   ```

   此语句创建名为 `SASoapTest/EmployeeList` 的新 SOAP Web 服务，此服务生成 SOAP 类型作为输出。它将从 `Employees` 表上选定所有列，并向客户端返回结果集。用引号将服务名引起来，因为服务名中存在斜线字符 (/)。

   DATATYPE ON 表示显式数据类型信息在 XML 结果集响应和输入参数中生成。此选项不影响生成的 WSDL 文档。

由于未指定 FORMAT 子句，所以 SOAP 服务格式将由下一个步骤中声明的相关 DISH 服务格式来决定。

4. 创建一个新 DISH 服务以充当 SOAP 服务的代理并生成 WSDL 文档。

   在 Interactive SQL 中执行以下 SQL 语句：

   ```
   CREATE SERVICE SASoapTest_DNET
       TYPE 'DISH'
       GROUP SASoapTest
       FORMAT 'DNET'
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA;
   ```

   从 .NET 访问的 DISH Web 服务应该用 FORMAT 'DNET' 子句来声明。GROUP 子句标识 DISH 服务应当处理的 SOAP 服务。前面步骤中创建的 `EmployeeList` 服务是 GROUP SASoapTest 的一部分，因为它被声明为 `SASoapTest/EmployeeList`。

5. 通过利用 Web 浏览器访问相关 WSDL 文档来验证 DISH Web 服务是否工作正常。

   打开 Web 浏览器，访问 http://localhost:8082/demo/SASoapTest_DNET。

   DISH 服务会自动生成一个 WSDL 文档，该文档将出现在浏览器窗口中。

您已设置运行 SOAP 和 DISH Web 服务的 SAP Sybase IQ Web 服务器，从而能够处理 Visual C# 客户端应用程序的请求。

下一步

在下一课中，将创建与 Web 服务器通信的 Visual C# 应用程序。

第 2 课：创建与 Web 服务器通信的 Visual C# 应用程序

在本课中，您将创建与 Web 服务器通信的 Visual C# 应用程序。

前提条件

本课假设您已经按照第 1 课中说明的方式设置了 Web 服务器。

完成本课需要最新版本的 Visual Studio。

本课假定您拥有在“教程：使用 Visual C# 访问 SOAP/DISH Web 服务”教程开头的“特权”部分中列出的角色和特权。

过程

本课中包含对 `localhost` 的引用。如果 Web 客户端与 Web 服务器运行在不同的计算机上，则使用第 1 课中 Web 服务器的主机名或 IP 地址而不是 `localhost`。

本示例使用 .NET Framework 2.0 中的函数。

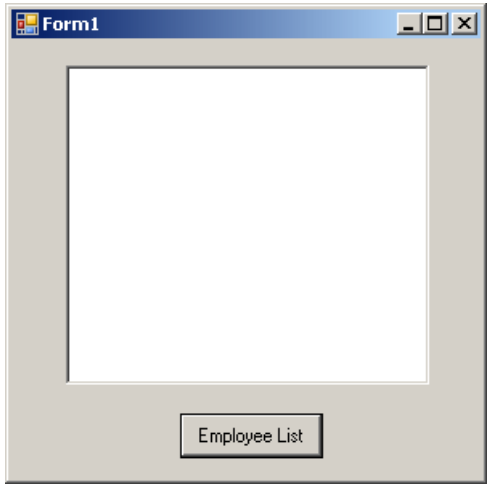
1. 启动 Visual Studio。
2. 新建一个 Visual C# **“Windows 窗体应用程序”**项目。

   将出现一个空的窗体。
3. 向项目添加 Web 引用。
   a. 单击**“Project”**»**“Add Service Reference”**。
   b. 在“Add Service Reference”窗口中，单击**“Advanced”**。
   c. 在“Service Reference Settings”窗口中，单击**“Add Web Reference”**。
   d. 在“Add Web Reference”窗口中的 **“URL”** 字段中，键入 `http://localhost:8082/demo/SASoapTest_DNET`。
   e. 单击**“Go”**（或绿色箭头）。

      Visual Studio 列出了 SASoapTest_DNET 服务里可用的 EmployeeList 方法。
   f. 单击**“Add Reference”**。

      Visual Studio 将 `localhost` 添加到**“Solution Explorer”**窗格的项目**“Web References”**中。
4. 用 Web 客户端应用程序所需的对象填充空窗体。

   将 ListBox 和 Button 对象从**“Toolbox”**窗格中拖到窗体上，并更新文本属性，使窗体与下图类似：



5. 编写访问 Web 参考并使用可用方法的过程。

   双击 `Employee List` 按钮，然后为按钮的 click 事件添加以下代码：

```
int sqlCode;

listBox1.Items.Clear();

localhost.SASoapTest_DNET proxy = new
localhost.SASoapTest_DNET();
```

```
DataSet results = proxy.EmployeeList(out sqlCode);
DataTableReader dr = results.CreateDataReader();
while (dr.Read())
{
  for (int i = 0; i < dr.FieldCount; i++)
  {
    string columnName = "(" + dr.GetDataTypeName(i)
                      + ")"
                      + dr.GetName(i);
    if (dr.IsDBNull(i))
    {
      listBox1.Items.Add(columnName + "=(null)");
    }
    else {
      System.TypeCode typeCode =
          System.Type.GetTypeCode(dr.GetFieldType(i));
      switch (typeCode)
      {
        case System.TypeCode.Int32:
          Int32 intValue = dr.GetInt32(i);
          listBox1.Items.Add(columnName + "="
              + intValue);
          break;
        case System.TypeCode.Decimal:
          Decimal decValue = dr.GetDecimal(i);
          listBox1.Items.Add(columnName + "="
              + decValue.ToString("c"));
          break;
        case System.TypeCode.String:
          string stringValue = dr.GetString(i);
          listBox1.Items.Add(columnName + "="
              + stringValue);
          break;
        case System.TypeCode.DateTime:
          DateTime dateValue = dr.GetDateTime(i);
          listBox1.Items.Add(columnName + "="
              + dateValue);
          break;
        case System.TypeCode.Boolean:
          Boolean boolValue = dr.GetBoolean(i);
          listBox1.Items.Add(columnName + "="
              + boolValue);
          break;
        case System.TypeCode.DBNull:
          listBox1.Items.Add(columnName
              + "=(null)");
          break;
        default:
          listBox1.Items.Add(columnName
              + "=(unsupported)");
          break;
      }
    }
  }
  listBox1.Items.Add("");
```

```
}
dr.Close();
```

6. 运行应用程序。

   单击“**Debug**”»“**Start Debugging**”。

7. 与 Web 数据库服务器通信。

   单击 `Employee List`。

   `ListBox` 对象以 (type)name=value 对形式显示 `EmployeeList` 结果集。以下输出说明了条目如何在 `ListBox` 对象上显示：

```
(Int32)EmployeeID=102
(Int32)ManagerID=501
(String)Surname=Whitney
(String)GivenName=Fran
(Int32)DepartmentID=100
(String)Street=9 East Washington Street
(String)City=Cornwall
(String)State=New York
(String)Country=USA
(String)PostalCode=02192
(String)Phone=6175553985
(String)Status=A
(String)SocialSecurityNumber=017349033
(String)Salary=$45,700.00
(DateTime)StartDate=28/08/1984 0:00:00 AM
(DateTime)TerminationDate=(null)
(DateTime)BirthDate=05/06/1958 0:00:00 AM
(Boolean)BenefitHealthInsurance=True
(Boolean)BenefitLifeInsurance=True
(Boolean)BenefitDayCare=False
(String)Sex=F
```

   Salary 金额已被转换为客户端的货币形式。

   将包含空值的值返回为 DBNull。对于只包含日期而不包含时间的值，将指派 00:00:00 或午夜作为时间。

来自 Web 服务器的 XML 响应包括已设置格式的结果集。所有列数据都转换为数据的字符串表示形式。以下结果集说明了结果集在发送给客户端时的格式：

```
<row>
 <EmployeeID>102</EmployeeID>
 <ManagerID>501</ManagerID>
 <Surname>Whitney</Surname>
 <GivenName>Fran</GivenName>
 <DepartmentID>100</DepartmentID>
 <Street>9 East Washington Street</Street>
 <City>Cornwall</City>
 <State>NY</State>
 <Country>USA</Country>
 <PostalCode>02192</PostalCode>
 <Phone>6175553985</Phone>
 <Status>A</Status>
 <SocialSecurityNumber>017349033</SocialSecurityNumber>
```

```
<Salary>45700.000</Salary>
<StartDate>1984-08-28-05:00</StartDate>
<TerminationDate xsi:nil="true" />
<BirthDate>1958-06-05-05:00</BirthDate>
<BenefitHealthInsurance>1</BenefitHealthInsurance>
<BenefitLifeInsurance>1</BenefitLifeInsurance>
<BenefitDayCare>0</BenefitDayCare>
<Sex>F</Sex>
</row>
```

包含日期或时间信息的列包括有 Web 服务器时间相对 UTC 的偏移。在上面的结果集中，偏移为 -05:00，即比 UTC（北美东部标准时间）时间相差 5 小时。

只含有日期的列的格式为 `yyyy-mm-dd-HH:MM` 或 `yyyy-mm-dd+HH:MM`。将向字符串后添加时区偏移（-HH:MM 或 +HH:MM）后缀。

只含有时间的列的格式为 `hh:mm:ss.nnn-HH:MM` 或 `hh:mm:ss.nnn+HH:MM`。将向字符串后添加时区偏移（-HH:MM 或 +HH:MM）后缀。

同时含有日期和时间的列的格式为 `yyyy-mm-ddThh:mm:ss.nnn-HH:MM` 或 `yyyy-mm-ddThh:mm:ss.nnn+HH:MM`。使用字母 'T' 分隔日期和时间。将向字符串后添加时区偏移（-HH:MM 或 +HH:MM）后缀。

上一课中指定了 DATATYPE ON 子句以生成 XML 结果集响应中的数据类型信息。来自 Web 服务器的响应片段如下所示。该类型信息与数据库列的数据类型相匹配。

```
<xsd:element name='EmployeeID' minOccurs='0' type='xsd:int' />
<xsd:element name='ManagerID' minOccurs='0' type='xsd:int' />
<xsd:element name='Surname' minOccurs='0' type='xsd:string' />
<xsd:element name='GivenName' minOccurs='0' type='xsd:string' />
<xsd:element name='DepartmentID' minOccurs='0' type='xsd:int' />
<xsd:element name='Street' minOccurs='0' type='xsd:string' />
<xsd:element name='City' minOccurs='0' type='xsd:string' />
<xsd:element name='State' minOccurs='0' type='xsd:string' />
<xsd:element name='Country' minOccurs='0' type='xsd:string' />
<xsd:element name='PostalCode' minOccurs='0' type='xsd:string' />
<xsd:element name='Phone' minOccurs='0' type='xsd:string' />
<xsd:element name='Status' minOccurs='0' type='xsd:string' />
<xsd:element name='SocialSecurityNumber' minOccurs='0'
type='xsd:string' />
<xsd:element name='Salary' minOccurs='0' type='xsd:decimal' />
<xsd:element name='StartDate' minOccurs='0' type='xsd:date' />
<xsd:element name='TerminationDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BirthDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BenefitHealthInsurance' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='BenefitLifeInsurance' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='BenefitDayCare' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='Sex' minOccurs='0' type='xsd:string' />
```

教程：使用 JAX-WS 访问 SOAP/DISH Web 服务

本教程演示如何创建 Java API for XML Web Services (JAX-WS) 客户端应用程序，访问 SQL Anywhere Web 服务器上的 SOAP/DISH 服务。

必需的软件

- SAP Sybase IQ
- JDK 1.7.0
- JAX-WS 2.2.7 或更高版本

能力和经验

- 熟悉 SOAP
- 熟悉 Java 和 JAX-WS
- 了解 SAP Sybase IQ Web 服务的基本知识

目标

- 创建并启动新的 SAP Sybase IQ Web 服务器数据库。
- 创建 SOAP Web 服务。
- 设置返回 SOAP 请求所包含信息的过程。
- 创建提供 WSDL 文档并用作代理的 DISH Web 服务。
- 在客户端计算机上使用 JAX-WS 处理 Web 服务器的 WSDL 文档。
- 创建 Java 客户端应用程序，以使用 WSDL 文档中的信息从 SOAP 服务中检索信息。

特权
需要具备以下特权才能执行本教程中的课程。

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

第 1 课：设置用于接收 SOAP 请求和发送 SOAP 响应的 Web 服务器

在本课中，您将设置运行 SOAP 和 DISH Web 服务的 SAP Sybase IQ Web 服务器，处理 JAX-WS 客户端应用程序的请求。

前提条件

本课假定您拥有在“教程：使用 JAX-WS 访问 SOAP/DISH Web 服务”教程开头的“特权”部分中列出的角色和特权。

过程

本课将设置 Web 服务器，并设置一个下一课将用到的简单 Web 服务。它对于使用代理软件观察 XML 消息流量具有指导意义。代理软件置身于客户端应用程序和 Web 服务器之间。

1. 使用以下命令启动 SAP Sybase IQ 演示数据库：

   ```
   iqsrv16 -xs http(port=8082) iqdemo.db
   ```

   此命令表示 HTTP Web 服务器应当监听 8082 端口上的请求。如果网络禁用了 8082 端口，则使用其它端口号。

2. 使用以下命令在 Interactive SQL 中连接数据库服务器：

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=demo"
   ```

3. 创建列出 `Employees` 表列的存储过程。

   在 Interactive SQL 中执行以下 SQL 语句：

   ```
   CREATE OR REPLACE PROCEDURE ListEmployees()
   RESULT (
       EmployeeID            INTEGER,
       Surname               CHAR(20),
       GivenName             CHAR(20),
       StartDate             DATE,
       TerminationDate       DATE )
   BEGIN
       SELECT EmployeeID, Surname, GivenName, StartDate,
   TerminationDate
       FROM Employees;
   END;
   ```

   这些语句创建名为 `ListEmployees` 的新过程，它定义结果集输出的结构，并从雇员表中选出某些列。

4. 创建新的 SOAP 服务来接受进来的请求。

   在 Interactive SQL 中执行以下 SQL 语句：

   ```
   CREATE SERVICE "WS/EmployeeList"
       TYPE 'SOAP'
       FORMAT 'CONCRETE' EXPLICIT ON
       DATATYPE ON
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA
       AS CALL ListEmployees();
   ```

   此语句创建名为 `WS/EmployeeList` 的新 SOAP Web 服务，此服务生成 SOAP 类型作为输出。当 Web 客户端向服务发送请求时，它调用 `ListEmployees` 过程。用引号将服务名引起来，因为服务名中存在斜线字符 (/)。

   从 JAX-WS 访问的 SOAP Web 服务应该用 FORMAT 'CONCRETE' 子句来声明。EXPLICIT ON 子句表示相应的 DISH 服务应当生成 XML 模式，该模式基于 `ListEmployees` 过程的结果集描述一个显式 dataset 对象。EXPLICIT 子句只对生成的 WSDL 文档有影响。

DATATYPE ON 表示显式数据类型信息在 XML 结果集响应和输入参数中生成。此选项不影响生成的 WSDL 文档。

5. 创建一个新 DISH 服务以充当 SOAP 服务的代理并生成 WSDL 文档。

   在 Interactive SQL 中执行以下 SQL 语句：

```
CREATE SERVICE WSDish
    TYPE 'DISH'
    FORMAT 'CONCRETE'
    GROUP WS
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA;
```

   从 JAX-WS 访问的 DISH Web 服务应该用 FORMAT 'CONCRETE' 子句来声明。GROUP 子句标识 DISH 服务应当处理的 SOAP 服务。前面步骤中创建的 EmployeeList 服务是 GROUP `WS` 的一部分，因为它被声明为 `WS/EmployeeList`。

6. 通过利用 Web 浏览器访问相关 WSDL 文档来验证 DISH Web 服务是否工作正常。

   打开 Web 浏览器，访问 http://localhost:8082/demo/WSDish。

   DISH 服务会自动生成一个 WSDL 文档，该文档将出现在浏览器窗口中。检查 `EmployeeListDataset` 对象，它类似以下输出内容：

```
<s:complexType name="EmployeeListDataset">
<s:sequence>
<s:element name="rowset">
  <s:complexType>
  <s:sequence>
  <s:element name="row" minOccurs="0" maxOccurs="unbounded">
    <s:complexType>
    <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="EmployeeID"
nillable="true" type="s:int" />
    <s:element minOccurs="0" maxOccurs="1" name="Surname"
nillable="true" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="GivenName"
nillable="true" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="StartDate"
nillable="true" type="s:date" />
   <s:element minOccurs="0" maxOccurs="1" name="TerminationDate"
nillable="true" type="s:date" />
    </s:sequence>
    </s:complexType>
  </s:element>
  </s:sequence>
  </s:complexType>
</s:element>
</s:sequence>
</s:complexType>
```

EmployeeListDataset 是 EmployeeList SOAP 服务中的 FORMAT 'CONCRETE' 和 EXPLICIT ON 子句生成的显式对象。在随后的课程中，wsimport 应用程序将使用此信息为该服务生成一个 SOAP 1.1 客户端接口。

您已设置运行 SOAP 和 DISH Web 服务的 SAP Sybase IQ Web 服务器，从而能够处理 JAX-WS 客户端应用程序的请求。

下一步

在下一课中，将创建与 Web 服务器通信的 Java 应用程序。

第 2 课：创建与 Web 服务器通信的 Java 应用程序

在本课中，您将处理从 DISH 服务生成的 WSDL 文档，并根据 WSDL 文档中定义的模式来创建 Java 应用程序，访问表数据。

前提条件

本课取决于第 1 课中执行的步骤。

本课假定您拥有在“教程：使用 JAX-WS 访问 SOAP/DISH Web 服务”教程开头的“特权”部分中列出的角色和特权。

过程

编写本文档时，JDK 1.7.0 中包含 JAX-WS，且 JAX-WS 的最新版本为 2.2.7。后面的步骤均以此版本为基础。要确定您的 JDK 中是否存在 JAX-WS，请检查 JDK bin 目录中是否存在 wsimport 应用程序。如果不存在，请转到 http://jax-ws.java.net/ 下载并安装最新版本的 JAX-WS。

本课中包含对 localhost 的引用。如果 Web 客户端与 Web 服务器运行在不同的计算机上，则使用第 1 课中 Web 服务器的主机名或 IP 地址而不是 localhost。

1. 在命令提示符下，为 Java 代码和生成的文件创建新的工作目录。转到该新目录下。
2. 使用以下命令生成调用 DISH Web 服务和导入 WSDL 文档的接口：

   ```
   wsimport -keep "http://localhost:8082/demo/WSDish"
   ```

   wsimport 应用程序从给定的 URL 读取 WSDL 文档。它生成 .java 文件为其创建接口，然后将它们编译到 .class 文件中。

   keep 选项表示生成类文件后不应删除 .java 文件。生成的 Java 源代码可让您理解所生成的类文件。

   wsimport 应用程序会在当前工作目录中创建新的子目录结构，名为 localhost_8082\demo\ws。下面列出了目录 ws 的内容：

   - EmployeeList.class

- EmployeeList.java
- EmployeeListDataset$Rowset$Row.class
- EmployeeListDataset$Rowset.class
- EmployeeListDataset.class
- EmployeeListDataset.java
- EmployeeListResponse.class
- EmployeeListResponse.java
- FaultMessage.class
- FaultMessage.java
- ObjectFactory.class
- ObjectFactory.java
- package-info.class
- package-info.java
- WSDish.class
- WSDish.java
- WSDishSoapPort.class
- WSDishSoapPort.java

3. 根据所生成源代码中定义的 dataset 对象模式，编写用于访问数据库服务器上表数据的 Java 应用程序。

以下是可进行此操作的 Java 应用程序示例。在当前工作目录中将源代码保存为 SASoapDemo.java。您的当前工作目录中必须包含 localhost 子目录。

```
// SASoapDemo.java illustrates a web service client that
// calls the WSDish service and prints out the data.

import java.util.*;
import javax.xml.ws.*;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import javax.xml.datatype.*;
import localhost._8082.demo.ws.*;

public class SASoapDemo
{
  public static void main( String[] args )
  {
    try {
      WSDish service = new WSDish();

      Holder<EmployeeListDataset> response =
          new Holder<EmployeeListDataset>();
      Holder<Integer> sqlcode = new Holder<Integer>();

      WSDishSoapPort port = service.getWSDishSoap();

      // This is the SOAP service call to EmployeeList
      port.employeeList( response, sqlcode );
```

```
      EmployeeListDataset result = response.value;
      EmployeeListDataset.Rowset rowset = result.getRowset();

     List<EmployeeListDataset.Rowset.Row> rows = rowset.getRow();

      String fieldType;
      String fieldName;
      String fieldValue;
      Integer fieldInt;
      XMLGregorianCalendar fieldDate;

      for ( int i = 0; i < rows.size(); i++ ) {
        EmployeeListDataset.Rowset.Row row = rows.get( i );

        fieldType =
row.getEmployeeID().getDeclaredType().getSimpleName();
        fieldName = row.getEmployeeID().getName().getLocalPart();
        fieldInt = row.getEmployeeID().getValue();
        System.out.println( "(" + fieldType + ")" + fieldName +
                            "=" + fieldInt );

        fieldType =
row.getSurname().getDeclaredType().getSimpleName();
        fieldName = row.getSurname().getName().getLocalPart();
        fieldValue = row.getSurname().getValue();
        System.out.println( "(" + fieldType + ")" + fieldName +
                            "=" + fieldValue );

        fieldType =
row.getGivenName().getDeclaredType().getSimpleName();
        fieldName = row.getGivenName().getName().getLocalPart();
        fieldValue = row.getGivenName().getValue();
        System.out.println( "(" + fieldType + ")" + fieldName +
                             "=" + fieldValue );

        fieldType =
row.getStartDate().getDeclaredType().getSimpleName();
        fieldName = row.getStartDate().getName().getLocalPart();
        fieldDate = row.getStartDate().getValue();
        System.out.println( "(" + fieldType + ")" + fieldName +
                            "=" + fieldDate );

        if ( row.getTerminationDate() == null ) {
          fieldType = "unknown";
          fieldName = "TerminationDate";
          fieldDate = null;
        } else {
          fieldType =

row.getTerminationDate().getDeclaredType().getSimpleName();
          fieldName =
row.getTerminationDate().getName().getLocalPart();
          fieldDate = row.getTerminationDate().getValue();
        }
        System.out.println( "(" + fieldType + ")" + fieldName +
                            "=" + fieldDate );
```

```
        System.out.println();
      }
    }
    catch (Exception x) {
      x.printStackTrace();
    }
  }
}
```

该应用程序将服务器提供的所有列数据打印为标准的系统输出。

注意： 该应用程序假定您的 SAP Sybase IQ Web 服务器按第 1 课的指示监听 8082 端口。将 `import localhost._8082.demo.ws.*` 代码行中的 8082 部分替换成您在启动 SAP Sybase IQ Web 服务器时指定的端口号。

有关此应用程序中使用的 Java 方法的详细信息，请参见 javax.xml.bind.JAXBElement 类的 API 文档，网址为 http://docs.oracle.com/javase/。

4. 使用以下命令编译 Java 应用程序：

```
javac SASoapDemo.java
```

5. 使用以下命令执行应用程序：

```
java SASoapDemo
```

6. 应用程序向 Web 服务器发送请求。它会收到一个包含 `EmployeeListResult`（带有含有几个行条目的行集）的 XML 结果集响应。

以下示例显示了运行中的 SASoapDemo 的输出：

```
(Integer)EmployeeID=102
(String)Surname=Whitney
(String)GivenName=Fran
(XMLGregorianCalendar)StartDate=1984-08-28
(unknown)TerminationDate=null

(Integer)EmployeeID=105
(String)Surname=Cobb
(String)GivenName=Matthew
(XMLGregorianCalendar)StartDate=1985-01-01
(unknown)TerminationDate=null
.
.
.
(Integer)EmployeeID=1740
(String)Surname=Nielsen
(String)GivenName=Robert
(XMLGregorianCalendar)StartDate=1994-06-24
(unknown)TerminationDate=null

(Integer)EmployeeID=1751
(String)Surname=Ahmed
(String)GivenName=Alex
(XMLGregorianCalendar)StartDate=1994-07-12
(XMLGregorianCalendar)TerminationDate=2008-04-18
```

仅当 TerminationDate 列值为非 NULL 时，才发送它。Java 应用程序用于检测 TerminationDate 列何时不出现。对于本示例，Employees 表的最后一行被改动，因此将设置一个非 NULL 终止日期。

以下示例显示了来自 Web 服务器的 SOAP 响应。通过执行查询得到的 SQLCODE 结果包含在响应中。

```
<tns:EmployeeListResponse>
 <tns:EmployeeListResult xsi:type='tns:EmployeeListDataset'>
  <tns:rowset>
    <tns:row> ... </tns:row>
    .
    .
    .
    <tns:row>
      <tns:EmployeeID xsi:type="xsd:int">1751</tns:EmployeeID>
      <tns:Surname xsi:type="xsd:string">Ahmed</tns:Surname>
      <tns:GivenName xsi:type="xsd:string">Alex</tns:GivenName>
      <tns:StartDate xsi:type="xsd:dateTime">1994-07-12</
tns:StartDate>
      <tns:TerminationDate xsi:type="xsd:dateTime">2010-03-22</
tns:TerminationDate>
    </tns:row>
  </tns:rowset>
 </tns:EmployeeListResult>
 <tns:sqlcode>0</tns:sqlcode>
</tns:EmployeeListResponse>
```

列名称和数据类型都包含在每个行集中。

三层计算和分布式事务

可以将 SAP Sybase IQ 用作数据库服务器或*资源管理器*，以参与由事务服务器协调的分布式事务。

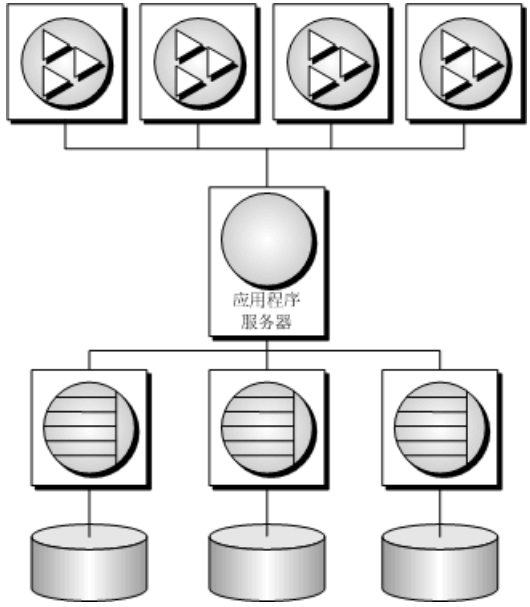
三层环境是一种常见的分布式事务环境，在此环境中应用程序服务器位于客户端应用程序和一组资源管理器之间。Sybase EAServer 及其它一些应用程序服务器也都是事务服务器。

Sybase EAServer 和 Microsoft Transaction Server 都使用 Microsoft 分布式事务处理协调器（Distributed Transaction Coordinator，简称 DTC）来协调事务。SAP Sybase IQ 提供了对 DTC 服务所控制的分布式事务的支持，因此可将 SAP Sybase IQ 用于上述两种应用程序服务器中的任意一种，也可用于任何其它基于 DTC 模型的产品。

将 SAP Sybase IQ 集成到三层环境中时，大多数工作都需要在应用程序服务器上完成。本节将介绍三层计算的概念和体系结构，并将概述 SAP Sybase IQ 的相关功能。但本节不介绍如何将应用程序服务器配置为与 SAP Sybase IQ 一起使用。有关详细信息，请参见应用程序服务器的文档。

三层计算体系结构

在三层计算中，应用程序逻辑保存在应用程序服务器（如 Sybase EAServer）上，该服务器位于资源管理器和客户端应用程序之间。在许多情况下，一个应用程序服务器可以访问多个资源管理器。在 Internet 环境中，客户端应用程序是基于浏览器的，而应用程序服务器通常是 Web 服务器扩展。



Sybase EAServer 以组件形式存储应用程序逻辑，并使这些组件可以供客户端应用程序使用。这些组件可以是 PowerBuilder® 组件、JavaBean，也可以是 COM 组件。

有关详细信息，请参见您的 Sybase EAServer 文档。

三层计算中的分布式事务

客户端应用程序或应用程序服务器使用一个事务处理数据库（例如 SAP Sybase IQ）时，数据库外部并不需要事务逻辑；但是，如果使用多个资源管理器，事务控制就必须包括事务所涉及的各种资源。应用程序服务器会向它们的客户端应用程序提供事务逻辑，以保证操作集能够以原子方式执行。

许多事务服务器（包括 Sybase EAServer）都使用 Microsoft 分布式事务处理协调器（Distributed Transaction Coordinator，简称 DTC），向它们的客户端应用程序提供事务服务。DTC 使用 *OLE 事务*，而该事务又使用*两阶段提交*协议协调涉及多个资源管理器的事务。您必须安装 DTC，然后才能使用本节介绍的功能。

分布式事务中的 SAP Sybase IQ

SAP Sybase IQ 可以参与由 DTC 协调的事务，这意味着您可在使用事务服务器（例如 Sybase EAServer 或 Microsoft Transaction Server）的分布式事务中使用 SAP Sybase IQ 数据库。您还可直接在应用程序中使用 DTC 来协调多个资源管理器中的事务。

分布式事务词汇

本节假设您对分布式事务有一定的了解。有关信息，请参见您的事务服务器文档。本节介绍一些常用的术语。

- *资源管理器* 是那些对事务中涉及的数据进行管理的服务。

  在通过 ADO.NET、OLE DB 或 ODBC 访问 SAP Sybase IQ 数据库服务器时，该数据库服务器可以用作分布式事务中的资源管理器。SAP Sybase IQ .NET 数据提供程序、OLE DB 提供程序和 ODBC 驱动程序用作客户端计算机上的资源管理器代理。SAP Sybase IQ .NET 数据提供程序使用 DbProviderFactory 和 TransactionScope 来支持分布式事务。
- 应用程序组件并不直接与资源管理器通信，但可以与*资源分发器* 通信，而资源分发器又管理与这些资源管理器的连接或连接池。

  SAP Sybase IQ 支持两种资源分发器：ODBC 驱动程序管理器和 OLE DB。
- 在事务组件请求数据库连接时（使用资源管理器），应用程序服务器会*征用* 参与该事务的每个数据库连接。DTC 和资源分发器执行征用过程。

两阶段提交

分布式事务通过两阶段提交进行管理。当事务的工作完成时，事务管理器 (DTC) 会询问事务中征用的所有资源管理器是否准备提交该事务。此阶段称为*准备* 提交。

如果所有资源管理器都作出准备提交的响应，则 DTC 会向每个资源管理器发送一个提交请求，并对其客户端作出事务已完成的响应。如果有一个或多个资源管理器不响应或者作出无法提交事务的响应，则事务的所有工作都将通过所有资源管理器进行回退。

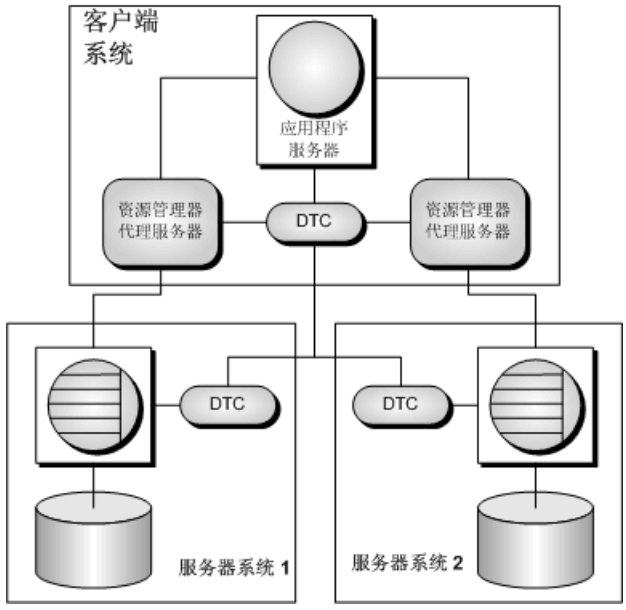
应用程序服务器如何使用 DTC

Sybase EAServer 和 Microsoft Transaction Server 都是组件服务器。应用程序逻辑以组件的形式保存，并且对客户端应用程序可用。

每个组件都有一个事务属性来指示组件参与事务的方式。创建组件时，必须将事务的工作编为组件，即资源管理器连接，而资源管理器连接是每个资源管理器负责的对数据的操作。但是，不需要将事务管理逻辑添加到组件中。在设置了事务属性来指示组件需要事务管理之后，EAServer 即会使用 DTC 来征用事务并管理两阶段提交过程。

分布式事务体系结构

下图显示了分布式事务的体系结构。在此例中，资源管理器代理可以是 ADO.NET、OLE DB 或 ODBC。



此例中使用了一个资源分发器。应用程序服务器请求 DTC 准备事务。DTC 和资源分发器征用事务中的每个连接。每个资源管理器必须与 DTC 和数据库都保持联系，这样才能进行工作并在必要时向 DTC 通报其事务状态。

每台计算机上都必须运行分布式事务协调器 (DTC) 服务，才能操作分布式事务。可以通过 Microsoft Windows 的 **“Services”** 窗口启动或停止 DTC；其中 DTC 服务任务名为 `MSDTC`。

有关详细信息，请参见您的 DTC 或 EAServer 文档。

分布式事务

当 SAP Sybase IQ 在分布式事务中被征用时，它会将事务控制权交给事务服务器，而 SAP Sybase IQ 将确保它不会执行任何隐式事务管理。SAP Sybase IQ 参与分布式事务时会自动规定以下条件：

- 如果自动提交处于使用状态，则它自动关闭。
- 在分布式事务过程中不允许数据定义语句（其提交属于意外情况）。
- 由应用程序直接（而不是通过事务协调器）向 SAP Sybase IQ 发出的显式 COMMIT 或 ROLLBACK 会生成错误。但是该事务并不中止。

- 一个连接一次只能参与一个分布式事务。
- 当连接被征用到分布式事务中时一定不能有未提交的操作。

DTC 隔离级别

DTC 具有一组由应用程序服务器指定的隔离级别。这些隔离级别将按照以下方式映射到 SAP Sybase IQ 隔离级别：

DTC 隔离级别	SAP Sybase IQ 隔离级别
ISOLATIONLEVEL_UNSPECIFIED	0
ISOLATIONLEVEL_CHAOS	0
ISOLATIONLEVEL_READUNCOMMITTED	0
ISOLATIONLEVEL_BROWSE	0
ISOLATIONLEVEL_CURSORSTABILITY	1
ISOLATIONLEVEL_READCOMMITTED	1
ISOLATIONLEVEL_REPEATABLEREAD	2
ISOLATIONLEVEL_SERIALIZABLE	3
ISOLATIONLEVEL_ISOLATED	3

从分布式事务恢复

如果数据库服务器在未提交操作处于待执行状态时出现了故障，则它必须在启动时回退或者提交那些操作，以保持事务的原子特征。

如果在恢复过程中发现来自分布式事务的未提交操作，则数据库服务器将尝试连接到 DTC，并请求被重新征用到待执行或不确定的事务中。重新征用完成后，DTC 即会指示数据库服务器回退或提交未完成的操作。

如果重新征用过程失败，则 SAP Sybase IQ 将无法知道是应该提交还是应该回退不确定的操作，因而恢复将失败。如果要恢复此类状态下的数据库而不考虑数据的不确定状态，则可使用以下数据库服务器选项来强制恢复：

- **-tmf** – 如果找不到 DTC，则未完成的操作将回退，并继续进行恢复。
- **-tmt** – 如果重新征用未能在指定的时间之前实现，则未完成的操作将回退，并继续进行恢复。

数据库工具接口 (DBTools)

SAP Sybase IQ 包括用于管理数据库的 Sybase Control Center 和一组实用程序。这些数据库管理实用程序执行各种任务，如备份数据库、创建数据库、将事务日志转换为 SQL 等。

支持的平台
所有数据库管理实用程序都使用一个名为*数据库工具库*的共享库。它是为 Windows 操作系统、Linux 和 Unix 提供的。对于 Windows，此库的名称为 `dbtool16.dll`。对于 Linux 和 Unix，此库的名称为 `libdbtool16_r.so`。

您可以开发自己的数据库管理实用程序，也可以通过调用数据库工具库将数据库管理功能合并到您的应用程序中。本节介绍数据库工具库的接口。本节将假定您已熟悉如何从所使用的开发环境中调用库例程。

数据库工具库为每个数据库管理实用程序都提供了函数或入口点。此外，函数的调用必须是在使用其它数据库工具函数之前和完成其它数据库工具函数的使用之后。

dbtools.h 头文件
dbtools 头文件列出了 DBTools 库的入口点以及用于将信息传入和传出该库的结构。`dbtools.h` 文件安装在 SAP Sybase IQ 安装目录下的 `SDK\Include` 子目录中。您应当查阅 `dbtools.h` 文件以获得有关入口点和结构成员的最新信息。

`dbtools.h` 头文件包含其它文件，如：

- **`sqlca.h`** – 这是为解析各种宏（而非 SQLCA 本身）而提供的。
- **`dllapi.h`** – 为与操作系统和语言相关的宏定义预处理器宏。
- **`dbtlvers.h`** – 定义 DB_TOOLS_VERSION_NUMBER 预处理器宏和其它版本特定的宏。

sqldef.h 头文件
`sqldef.h` 头文件包括错误返回值。

dbrmt.h 头文件
随 SAP Sybase IQ 提供的 `dbrmt.h` 头文件介绍了 DBTools 库中的 DBRemoteSQL 入口点以及用于将信息传入和传出 DBRemoteSQL 入口点的结构。`dbrmt.h` 文件安装在 SAP Sybase IQ 安装目录下的 `SDK\Include` 子目录中。您应当查阅 `dbrmt.h` 文件以获得有关 DBRemoteSQL 入口点和结构成员的最新信息。

DBTools 导入库

要使用 DBTools 函数，您必须将应用程序链接到包含所需函数定义的 DBTools *导入库*。

对于 Unix 系统，不需要任何导入库。请直接链接到 `libdbtool16.so`（非线程）或 `libdbtool16_r.so`（线程）。

导入库
SAP Sybase IQ 为 Windows 提供了 DBTools 接口的导入库。对于 Windows 系统，可以在 SAP Sybase IQ 安装目录下的 `SDK\Lib\x86` 和 `SDK\Lib\x64` 子目录中找到导入库。所提供的 DBTools 导入库如下所示：

编译器	库
Microsoft Windows	`dbtlstm.lib`

DBTools 库初始化和终止化

在使用任何其它 DBTools 函数之前，您必须先调用 DBToolsInit。使用完 DBTools 库后，您必须调用 DBToolsFini。

DBToolsInit 和 DBToolsFini 函数的主要用途是允许 DBTools 库装载和卸载 SAP Sybase IQ 消息库。消息库包含 DBTools 内部使用的所有本地化版本的错误消息和提示。如果未调用 DBToolsFini，消息库的引用计数不会减少，而且不会被卸载，因此请注意：一定要有一对匹配的 DBToolsInit/DBToolsFini 调用。

下列代码段阐释了如何初始化和终止化 DBTools：

```
// Declarations
a_dbtools_info  info;
short           ret;

//Initialize the a_dbtools_info structure
memset( &info, 0, sizeof( a_dbtools_info) );
info.errorrtn = (MSG_CALLBACK)MyErrorRtn;

// initialize the DBTools library
ret = DBToolsInit( &info );
if( ret != EXIT_OKAY ) {
    // library initialization failed
    ...
}
// call some DBTools routines ...
...
// finalize the DBTools library
DBToolsFini( &info );
```

DBTools 函数调用

所有工具都是通过先填写结构然后调用 DBTools 库中的函数（或*入口点*）来运行的。每个入口点都将指向单个结构的指针视为参数。

下面的示例展示了如何在 Windows 操作系统上使用 DBBackup 函数。

```
// Initialize the structure
a_backup_db backup_info;
memset( &backup_info, 0, sizeof( backup_info ) );

// Fill out the structure
backup_info.version = DB_TOOLS_VERSION_NUMBER;
backup_info.output_dir = "c:\\backup";
backup_info.connectparms
="UID=<user_id>;PWD=<password>;DBF=iqdemo.db";

backup_info.confirmrtn = (MSG_CALLBACK) ConfirmRtn ;
backup_info.errorrtn = (MSG_CALLBACK) ErrorRtn ;
backup_info.msgrtn = (MSG_CALLBACK) MessageRtn ;
backup_info.statusrtn = (MSG_CALLBACK) StatusRtn ;
backup_info.backup_database = TRUE;

// start the backup
DBBackup( &backup_info );
```

回调函数

DBTools 结构中有几个类型为 MSG_CALLBACK 的元素。这些是指向回调函数的指针。

使用回调函数

回调函数允许 DBTools 函数将对操作的控制返回到用户的调用应用程序中。DBTools 库使用回调函数来处理因以下四种用途而由 DBTools 函数发送到用户的消息：

- **确认** – 当需要由用户确认操作时调用。例如，如果备份目录不存在，则工具库会询问是否需要创建它。
- **错误消息** – 在发生错误（如在执行某个操作时磁盘空间不足）时被调用以处理消息。
- **信息消息** – 在工具要向用户显示某些消息（如正在卸载的当前表的名称）时调用。
- **状态信息** – 针对工具调用以显示操作的状态（如卸载表时的完成百分比）。

将回调函数指派给结构

您可以将回调例程直接指派给结构。以下语句是使用备份结构的一个示例：

```
backup_info.errorrtn = (MSG_CALLBACK) MyFunction
```

MSG_CALLBACK 在随 SAP Sybase IQ 提供的 dllapi.h 头文件中定义。工具例程可以使用应当出现在相应用户界面中的消息回调调用应用程序，而无论该用户界面是窗口环境、基于字符的系统上的标准输出还是其它用户界面。

确认回调函数的示例
下面这个示例确认例程要求用户对提示回答“是”或“否”并返回用户的选择：

```
extern short _callback ConfirmRtn(
        char * question )
{
    int ret = IDNO;
    if( question != NULL ) {
        ret = MessageBox( HwndParent, question,
        "Confirm", MB_ICONEXCLAMATION|MB_YESNO );
    }
    return( ret == IDYES );
}
```

错误回调函数的示例
下面是一个错误消息处理例程的示例，它在一个窗口中显示错误消息。

```
extern short _callback ErrorRtn(
        char * errorstr )
{
    if( errorstr != NULL ) {
        MessageBox( HwndParent, errorstr, "Backup Error",
MB_ICONSTOP|MB_OK );
    }
    return( 0 );
}
```

消息回调函数的示例
消息回调函数的公用实现会将消息输出到屏幕上：

```
extern short _callback MessageRtn(
        char * messagestr )
{
    if( messagestr != NULL ) {
                    OutputMessageToWindow( messagestr );
    }
    return( 0 );
}
```

状态回调函数的示例
状态回调例程在工具需要显示操作的状态（如卸载表时的完成百分比）时被调用。公用实现只将消息输出到屏幕上：

```
extern short _callback StatusRtn(
        char * statusstr )
{
    if( statusstr != NULL ) {
        OutputMessageToWindow( statusstr );
    return( 0 );
}
```

版本号和兼容性

每个结构都有一个指示版本号的成员。您应在调用任意 DBTools 函数之前将版本字段设置为应用程序开发时所针对的 DBTools 库的版本号。当您将 dbtools.h 头文件包括进来时便定义了 DBTools 库的当前版本。

下面的示例中将当前版本指派给了 a_backup_db 结构的一个实例：

```
backup_info.version = DB_TOOLS_VERSION_NUMBER;
```

版本号允许应用程序继续处理较新版本的 DBTools 库。即使新成员已添加到 DBTools 结构中，DBTools 函数也使用由应用程序提供的版本号来允许应用程序工作。

如果更新了任何 DBTools 结构或是发布了更新版本的软件，版本号就会增大。如果您使用了 DB_TOOLS_VERSION_NUMBER 并用新版本的 DBTools 头文件重建您的应用程序，则必须部署新版本的 DBTools 库。

位字段

许多 DBTools 结构都使用位字段来以紧凑方式保存布尔信息。例如，备份结构包括以下位字段：

```
a_bit_field     backup_database : 1;
a_bit_field     backup_logfile  : 1;
a_bit_field     no_confirm  : 1;
a_bit_field     quiet       : 1;
a_bit_field     rename_log  : 1;
a_bit_field     truncate_log : 1;
a_bit_field     rename_local_log: 1;
a_bit_field     server_backup   : 1;
```

每个位字段的长度都是一位，这由结构声明中冒号右侧的 1 来指示。所使用的特定数据类型取决于指派给 a_bit_field 的值，此参数在 dbtools.h 的顶部设置且与操作系统相关。

将值 0 或 1 指派给位字段，以传递此结构中的布尔信息。

DBTools 的示例

可以在 %ALLUSERSPROFILE%\SybaseIQ\samples\SQLAnywhere\DBTools 目录中找到本示例以及对其进行编译的说明。示例程序本身在 main.cpp 中。该示例说明了如何使用 DBTools 库执行数据库备份。

```
#define WIN32

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

```
#include "windows.h"
#include "sqldef.h"
#include "dbtools.h"
extern short _callback ConfirmCallBack( char * str )
{
    if( MessageBox( NULL, str, "Backup",
         MB_YESNO|MB_ICONQUESTION ) == IDYES )
    {
        return 1;
    }
    return 0;
}
extern short _callback MessageCallBack( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
extern short _callback StatusCallBack( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
extern short _callback ErrorCallBack( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s\n", str );
    }
    return 0;
}
typedef void (CALLBACK *DBTOOLSPROC)( void * );
typedef short (CALLBACK *DBTOOLSFUNC)( void * );

// Main entry point into the program.
int main( int argc, char * argv[] )
{
    a_dbtools_info  dbt_info;
    a_backup_db     backup_info;
    char            dir_name[ _MAX_PATH + 1];
    char            connect[ 256 ];
    HINSTANCE       hinst;
    DBTOOLSFUNC     dbbackup;
    DBTOOLSFUNC     dbtoolsinit;
    DBTOOLSPROC     dbtoolsfini;
    short           ret_code;

    // Always initialize to 0 so new versions
    // of the structure will be compatible.
    memset( &dbt_info, 0, sizeof( a_dbtools_info ) );
    dbt_info.errorrtn = (MSG_CALLBACK)MessageCallBack;;
```

```
    memset( &backup_info, 0, sizeof( a_backup_db ) );
    backup_info.version = DB_TOOLS_VERSION_NUMBER;
    backup_info.quiet = 0;
    backup_info.no_confirm = 0;
    backup_info.confirmrtn = (MSG_CALLBACK)ConfirmCallBack;
    backup_info.errorrtn = (MSG_CALLBACK)ErrorCallBack;
    backup_info.msgrtn = (MSG_CALLBACK)MessageCallBack;
    backup_info.statusrtn = (MSG_CALLBACK)StatusCallBack;
    if( argc > 1 )
    {
        strncpy( dir_name, argv[1], _MAX_PATH );
    }
    else
    {
        // DBTools does not expect (or like) a trailing slash
        strcpy( dir_name, "c:\\temp" );
    }
    backup_info.output_dir = dir_name;
    if( argc > 2 )
    {
        strncpy( connect, argv[2], 255 );
    }
    else
    {
        strcpy( connect, "DSN=Sybase IQ Demo" );
    }
    backup_info.connectparms = connect;
    backup_info.quiet = 0;
    backup_info.no_confirm = 0;
    backup_info.backup_database = 1;
    backup_info.backup_logfile = 1;
    backup_info.rename_log = 0;
    backup_info.truncate_log = 0;
    hinst = LoadLibrary( "dbtool16.dll" );
    if( hinst == NULL )
    {
        // Failed
        return EXIT_FAIL;
    }
    dbbackup = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
        "_DBBackup@4" );
    dbtoolsinit = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
        "_DBToolsInit@4" );
    dbtoolsfini = (DBTOOLSPROC) GetProcAddress( (HMODULE)hinst,
        "_DBToolsFini@4" );
    ret_code = (*dbtoolsinit)( &dbt_info );
    if( ret_code != EXIT_OKAY ) {
        return ret_code;
    }
    ret_code = (*dbbackup)( &backup_info );
    (*dbtoolsfini)( &dbt_info );
    FreeLibrary( hinst );
    return ret_code;
}
```

软件组件的退出代码

所有数据库工具库入口点使用如下退出代码。SAP Sybase IQ 实用程序也使用这些退出代码。

代码	状态	解释
0	EXIT_OKAY	成功
1	EXIT_FAIL	一般故障
2	EXIT_BAD_DATA	文件格式无效
3	EXIT_FILE_ERROR	文件未找到，无法打开
4	EXIT_OUT_OF_MEMORY	内存不足
5	EXIT_BREAK	被用户终止
6	EXIT_COMMUNICATIONS_FAIL	通信失败
7	EXIT_MISSING_DATABASE	缺少一个必需的数据库名
8	EXIT_PROTOCOL_MISMATCH	客户端/服务器协议不匹配
9	EXIT_UNABLE_TO_CONNECT	无法连接到数据库服务器
10	EXIT_ENGINE_NOT_RUNNING	数据库服务器未运行
11	EXIT_SERVER_NOT_FOUND	未找到数据库服务器
12	EXIT_BAD_ENCRYPT_KEY	加密密钥缺失或错误
13	EXIT_DB_VER_NEWER	必须升级服务器才能运行数据库
14	EXIT_FILE_INVALID_DB	文件不是数据库
15	EXIT_LOG_FILE_ERROR	日志文件缺失或其它错误
16	EXIT_FILE_IN_USE	文件正在使用
17	EXIT_FATAL_ERROR	发生致命错误
18	EXIT_MISSING_LICENSE_FILE	缺少服务器许可文件
19	EXIT_BACKGROUND_SYNC_ABORTED	中止后台同步以允许进行优先级较高的操作
20	EXIT_FILE_ACCESS_DENIED	访问被拒绝，数据库无法启动
255	EXIT_USAGE	命令行上的参数无效

代码	状态	解释
21	EXIT_SERVER_NAME_IN_USE	另一个同名服务器当前正在运行。

这些退出代码在 `%IQDIR16%\sdk\include\sqldef.h` 文件中定义。

数据库工具 C API 参考

头文件为 `dbtools.h` 和 `dbrmt.h`。

附录：使用 OLAP

OLAP（联机分析处理）是对存储在关系数据库中的信息执行数据分析的有效方法。

使用 OLAP，可以基于不同的维度对数据进行分析、获取含小计行的结果集，以及将数据组织成多维立方体，所有这些操作均可通过一个 SQL 查询完成。您也可以使用过滤器深入了解数据，从而快速返回结果集。本章介绍 SAP Sybase IQ 支持的 SQL/OLAP 功能。

注意： OLAP 示例中显示的表可从 `iqdemo` 数据库中获得。

关于 OLAP

这些分析函数（使用它们，可以通过一个 SQL 语句执行复杂数据分析）通过一种名为“联机分析处理”(OLAP) 的软件技术来执行。以下列表中显示了此类函数：

- **GROUP BY** 子句扩展 - **CUBE** 和 **ROLLUP**
- 分析函数：
  - 简单集合 - **AVG**、**COUNT**、**MAX**、**MIN**、**SUM**、**STDDEV** 和 **VARIANCE**

    注意： 您可以将除 **Grouping()** 以外的简单集合函数与 OLAP 窗口函数结合使用。
  - 窗口函数：
    - 窗口化集合 - **AVG**、**COUNT**、**MAX**、**MIN** 和 **SUM**
    - 排名函数 - **RANK**、**DENSE_RANK**、**PERCENT_RANK** 和 **NTILE**
    - 统计函数 – **STDDEV**、**STDDEV_SAMP**、**STDDEV_POP**、**VARIANCE**、**VAR_POP**、**VAR_SAMP**、**REGR_AVGX**、**REGR_AVGY**、**REGR_COUNT**、**REGR_INTERCEPT**、**REGR_R2**、**REGR_SLOPE**、**REGR_SXX**、**REGR_SXY**、**REGR_SYY**、**CORR**、**COVAR_POP**、**COVAR_SAMP**、**CUME_DIST**、**EXP_WEIGHTED_AVG** 和 **WEIGHTED_AVG**。
    - 分布函数 - **PERCENTILE_CONT** 和 **PERCENTILE_DISC**
  - 数值函数 - **WIDTH_BUCKET**、**CEIL**、**LN**、**EXP**、**POWER**、**SQRT** 和 **FLOOR**

作为对 1999 SQL 标准的修改，引入了对 ANSI SQL 标准的扩展以包含复杂数据分析。SAP Sybase IQ SQL 增强支持这些扩展。

有些数据库产品会提供一个单独的 OLAP 模块，您在分析数据之前需要先将数据从数据库移至该 OLAP 模块。与此不同的是，SAP Sybase IQ 将 OLAP 功能内置在数据库本身中，使您可以轻松且顺利地进行部署以及与其它数据库功能（如存储过程）集成。

OLAP 优点

OLAP 函数与 **GROUPING**、**CUBE** 和 **ROLLUP** 扩展组合使用时，具有以下两大优点。

首先，利用它们，可以执行多维数据分析、数据采集、时序分析、趋势分析、成本分配、目标寻求、即席多维结构更改、非过程建模，以及异常警告，执行这些操作时通常是利用一个 SQL 语句。 其次，窗口集合函数和报告聚合函数使用一个名为 *window* 的关系运算符，与使用自连接或相关子查询的同语义查询相比，该运算符可以更有效地执行。使用 OLAP 获得的结果集可以包含小计行，并且可以组织成多维 CUBE。

可计算各种区间的移动平均值和移动总和，在所选列值改变时重置集合和秩，以及使用简单的术语来表示复杂的比值。在一个查询表达式的范围内，您可以定义多个不同的 OLAP 函数，每个函数都有它自己的分区规则。

OLAP 计算

OLAP 计算可概念化为多个查询执行阶段，这些阶段共同配合来生成最终结果。

您可以通过查询中的相关子句来标识执行的 OLAP 阶段。例如，如果 SQL 查询规范包含窗口函数，则会首先处理 **WHERE**、**JOIN**、**GROUP BY** 和 **HAVING** 子句。分区是在 **GROUP BY** 子句中定义组之后、对查询的 **ORDER BY** 子句中的最终 **SELECT** 列表进行计算之前创建的。

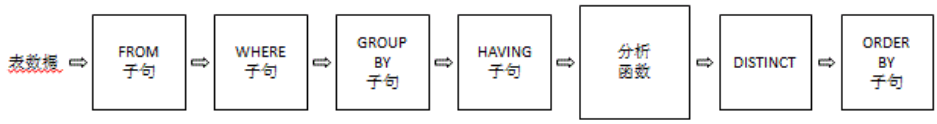
为了进行分组，将所有 NULL 视为属于同一组，即使 NULL 彼此不相等也是如此。

HAVING 子句充当过滤器 （与 **WHERE** 子句非常相似），用于过滤 **GROUP BY** 子句的结果。

请考虑涉及 ANSI SQL 标准中的 SQL 语句和子句（**SELECT**、**FROM**、**WHERE**、**GROUP BY** 和 **HAVING**）的简单查询规范的语义：

1. 查询生成一组满足 **FROM** 子句中存在的表表达式的行。
2. **WHERE** 子句中的谓词应用到表中的行。不满足 **WHERE** 子句条件的行（不等于 True）将被拒绝。
3. 除集合函数外，系统还会为其余每一行计算 **SELECT** 列表和 **GROUP BY** 子句 中的表达式。
4. 结果行根据 **GROUP BY** 子句中表达式的不同值组合在一起，并将 NULL 视为每个域中的特殊值。如果存在 **PARTITION BY** 子句，则 **GROUP BY** 子句中的表达式充当分区键。
5. 对于每个分区，系统将计算 **SELECT** 列表或 **HAVING** 子句中存在的集合函数。单个表行一旦集合在一起，便无法再显示在中间结果集中。新结果集包含 **GROUP BY** 表达式以及为每个分区计算的集合函数的值。
6. **HAVING** 子句中的条件应用到结果组。不满足 **HAVING** 子句的组将被删除。
7. 结果根据 **PARTITION BY** 子句中定义的边界进行分区。系统将为结果窗口计算 OLAP 窗口函数（秩和集合）。

图 1： 对 OLAP 的 SQL 处理



GROUP BY 子句扩展

使用 **GROUP BY** 子句的扩展，应用程序开发人员可以编写用于执行以下操作的复杂 SQL 语句：

- 将输入行以多维形式分区，并将结果组的多个子集合并。
- 创建“数据 CUBE”，从而提供稀疏的多维结果集以进行数据采集分析。
- 创建包含原始组（并且也可包含小计和总计行）的结果集。

OLAP Grouping() 操作（如 **ROLLUP** 和 **CUBE**）可概念化为前缀和小计行。

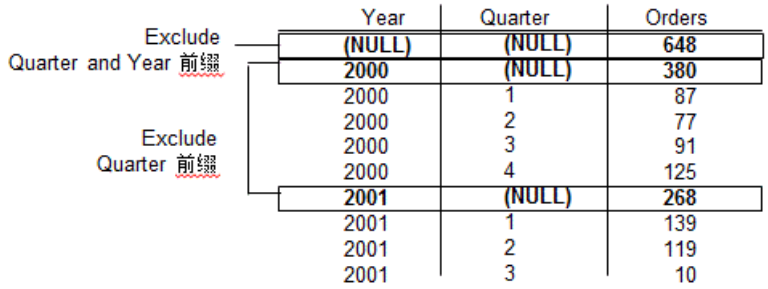
前缀
前缀 列表针对所有包含 **GROUP BY** 子句的查询而构造。前缀是 **GROUP BY** 子句中项的子集，并且通过从查询的 **GROUP BY** 子句中的项中排除最右侧的一个或多个项构造而成。剩余的列称为*前缀列*。

ROLLUP 示例 1 - 在以下 **ROLLUP** 示例查询中，**GROUP BY** 列表包括 *Year* 和 *Quarter* 这两个变量：

```
SELECT year (OrderDate) AS Year, quarter(OrderDate)
  AS Quarter, COUNT(*) Orders
FROM SalesOrders
GROUP BY ROLLUP(Year, Quarter)
ORDER BY Year, Quarter
```

查询的两个前缀是：

- Exclude Quarter - 前缀列的集包含一个列 `Year`。
- Exclude both Quarter and Year – 不存在前缀列。



Year	Quarter	Orders
(NULL)	**(NULL)**	**648**
2000	**(NULL)**	**380**
2000	1	87
2000	2	77
2000	3	91
2000	4	125
2001	**(NULL)**	**268**
2001	1	139
2001	2	119
2001	3	10

注意： **GROUP BY** 列表包含与项数量相同的前缀。

Group by ROLLUP 和 Group by CUBE

ROLLUP 和 **CUBE** 是指定常见分组前缀的语法快捷方式。

Group by ROLLUP

ROLLUP 运算符要求以参数的方式提供分组表达式的有序列表。

ROLLUP 语法。

```
SELECT ⋯ [ GROUPING (column-name) ⋯ ] ⋯
GROUP BY [ expression [, ⋯]
| ROLLUP ( expression [, ⋯] ) ]
```

GROUPING 采用列名作为参数，并返回下表中所列的布尔值：

表 1. 使用 ROLLUP 运算符时 GROUPING 返回的值

如果结果值是	GROUPING 将返回
由 **ROLLUP** 运算创建的 NULL	1 (TRUE)
指示该行是小计所在行的 NULL	1 (TRUE)
并非由 **ROLLUP** 运算创建	0 (FALSE)
存储的 NULL	0 (FALSE)

ROLLUP 首先计算 **GROUP BY** 子句中指定的标准集合值。然后，**ROLLUP** 在整个分组列的列表中从右侧移到左侧，并以累积方式创建更高级别的小计。在结尾处创建总计。如果 *n* 代表分组列数，则 **ROLLUP** 会创建 *n*+1 个级别的小计。

此 SQL 语法...	定义以下集...
`GROUP BY ROLLUP (A, B, C);`	(A, B, C) (A, B) (A) ( )

ROLLUP 和小计行

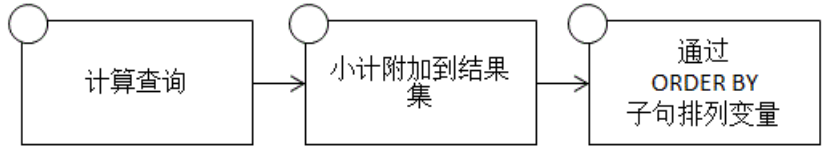
ROLLUP 等同于一组 **GROUP BY** 查询的 **UNION**。以下查询的结果集是相同的。**GROUP BY** (A, B) 的结果集包括所有那些 A 和 B 保持不变的行的小计。要实现联合，可为列 C 分配 NULL。

此 ROLLUP 查询...	等同于不含 ROLLUP 的以下查询...
`select year(order-date) as year, quar-ter(orderdate) as Quarter, count(*) Or-dersfrom SalesOr-dersgroup by Rollup (year, quarter)order by year, quarter`	`Select null,null, count(*) Orders from SalesOrdersunion allSELECT year(orderdate) AS YEAR, NULL, count(*) Orders from SalesOrdersGROUP BY year(orderdate) union allSELECT year(orderdate) as YEAR, quarter(or-derdate) as QUATER, count(*) Orders from SalesOrdersGROUP BY year(order-date), quarter(orderdate)`

小计行可以帮助您分析数据，尤其是在以下情况下：有大量数据、数据有不同的维数、数据包含在不同的表甚至完全不同的数据库中。例如，一名销售经理可能会发现按销售代表、区域和季度进行分类的有关销售数据的报告对于了解销售模式十分有用。数据小计可让销售经理从不同的角度了解总体销售情况。如果根据销售经理要比较的条件提供了摘要信息，分析此数据就会更加容易。

使用 OLAP，用户看不到分析和计算行和列小计的过程。

图 2：小计



1. 此步骤生成尚未考虑 **ROLLUP** 的中间结果集。

2. 计算小计并将其附加到结果集中。
3. 行根据查询中的 **ORDER BY** 子句排列。

NULL 和小计行

当 **GROUP BY** 运算的输入行包含 NULL 时，可能会将由 **ROLLUP** 或 **CUBE** 运算添加的小计行与包含的 NULL 属于原始输入数据的行混淆。

Grouping() 函数通过以下方法将小计行与其它行区分开：采用 **GROUP BY** 列表中的列作为它的参数，并在该列为 NULL 时（因为行是小计行）返回 1，在该列不为 NULL 时返回 0。

以下示例的结果集中包含 Grouping() 列。包含 NULL 的行会突出显示，这是由输入数据导致的，并不因为它们是小计行。Grouping() 列会突出显示。该查询是 `Employees` 表和 `SalesOrders` 表之间的外连接。该查询选择居住在德克萨斯、纽约或加利福尼亚的女雇员。NULL 显示在不是销售代表（因而没有销售额）的那些女雇员所对应的列中。

注意： 例如，使用 SAP Sybase IQ demo 数据库 `iqdemo.db`。

```
SELECT Employees.EmployeeID as EMP, year(OrderDate) as
  YEAR, count(*) as ORDERS, grouping(EMP) as
  GE, grouping(YEAR) as GY
  FROM Employees LEFT OUTER JOIN SalesOrders on
  Employees.EmployeeID = SalesOrders.SalesRepresentative
  WHERE Employees.Sex IN ('F') AND Employees.State
  IN ('TX', 'CA', 'NY')
GROUP BY ROLLUP (YEAR, EMP)
ORDER BY YEAR, EMP
```

前面的查询返回：

```
EMP          YEAR       ORDERS     GE     GY
------       ----       ------     --     --
NULL         NULL            5      1      0
NULL         NULL          169      1      1
 102         NULL            1      0      0
 309         NULL            1      0      0
1062         NULL            1      0      0
1090         NULL            1      0      0
1507         NULL            1      0      0
NULL         2000           98      1      0
 667         2000           34      0      0
 949         2000           31      0      0
1142         2000           33      0      0
NULL         2001           66      1      0
 667         2001           20      0      0
949          2001           22      0      0
1142         2001           24      0      0
```

对于每个前缀，*小计行*是对应于所有那些前缀列具有相同值的行而构建的。

为了说明 **ROLLUP** 结果，请再看一下示例查询：

```
SELECT year (OrderDate) AS Year, quarter
  (OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
  GROUP BY ROLLUP (Year, Quarter)
  ORDER BY Year, Quarter
```

在此查询中，包含 Year 列的前缀生成了年份为 2000 的汇总行以及年份为 2001 的汇总行。前缀的单一汇总行不包括列，该行是中间结果集中所有行的小计。

小计行中每一列的值如下：

- 前缀中包括的列 - 列的值。例如，在上一查询中，年份为 2000 的行的小计对应的 Year 列的值为 2000。
- 从前缀中排除的列 – NULL。例如，对于由包含年份列的前缀生成的小计行，Quarter 列的值为 NULL。
- 集合函数 – 所排除的列的值的集合。
  小计值是针对基础数据中的行而不是集合行而计算的。在许多情况下，例如，使用 **SUM** 或 **COUNT** 时，结果都是相同的，但是，在使用 **AVG**、**STDDEV** 和 **VARIANCE** 这样的统计函数时，结果是不同的，因而这时差别就很重要。

ROLLUP 运算符的限制如下：

- **ROLLUP** 运算符支持除 **COUNT DISTINCT** 和 **SUM DISTINCT** 以外所有可用于 **GROUP BY** 子句的集合函数。
- **ROLLUP** 只能用在 **SELECT** 语句中；子查询中无法使用 **ROLLUP**。
- 当前不支持将多个 **ROLLUP**、**CUBE** 和 **GROUP BY** 列组合在同一个 **GROUP BY** 子句中的分组规范。
- 不支持以常量表达式作为 **GROUP BY** 键。

ROLLUP 示例 2 - 以下示例说明了 **ROLLUP** 和 **GROUPING** 的用法并显示了由 **GROUPING** 创建的一组掩码列。显示在列 S、N 和 C 中的数字 0 和 1 是由 **GROUPING** 返回的值，表示 **ROLLUP** 结果的值。程序可通过使用掩码“011”标识小计行，使用“111”标识总计行来分析此查询的结果。

```
SELECT size, name, color, SUM(quantity),
  GROUPING(size) AS S,
  GROUPING(name) AS N,
  GROUPING(color) AS C
FROM Products
GROUP BY ROLLUP(size, name, color) HAVING (S=1 or N=1 or C=1)
ORDER BY size, name, color;
```

前面的查询返回：

```
size                name            color      SUM  S  N  C
----                -----           ------      ---  -  -  -
(NULL)              (NULL)           (NULL)    496  1  1  1
Large               (NULL)           (NULL)     71  0  1  1
Large               Sweatshirt       (NULL)     71  0  0  1
Medium              (NULL)           (NULL)    134  0  1  1
Medium              Shorts           (NULL)     80  0  0  1
Medium              Tee Shirt        (NULL)     54  0  0  1
```

```
One size fits all  (NULL)          (NULL)    263  0  1  1
One size fits all  Baseball Cap    (NULL)    124  0  0  1
One size fits all  Tee Shirt       (NULL)     75  0  0  1
One size fits all  Visor           (NULL)     64  0  0  1
Small              (NULL)          (NULL)     28  0  1  1
Small              Tee Shirt       (NULL)     28  0  0  1
```

注意： 在 Rollup 示例 2 的结果中，SUM 列显示为 `SUM(products.quantity)`。

ROLLUP 示例 3 - 以下示例说明如何使用 **GROUPING** 来区分存储的 NULL 值和 **ROLLUP** 运算创建的 "NULL" 值。存储的 NULL 值随后在列 `prod_id` 中显示为 [NULL]，而 **ROLLUP** 创建的 "NULL" 值在 `PROD_IDS` 列中由 ALL 取代，如查询中所指定的那样。

```
SELECT year(ShipDate) AS Year,
                ProductID, SUM(quantity)AS OSum,
CASE
        WHEN GROUPING(Year) = 1
        THEN 'ALL'
        ELSE
        CAST(Year AS char(8))
END,
CASE
        WHEN GROUPING(ProductID) = 1
        THEN 'ALL'
        ELSE
        CAST(ProductID as char(8))
END
FROM SalesOrderItems
GROUP BY ROLLUP(Year, ProductID) HAVING OSum > 36
ORDER BY Year, ProductID;
```

前面的查询返回：

```
Year  ProductID  OSum  ...(Year)...  ...(ProductID)...
---------     -------   ---   ----------   --------
NULL           NULL   28359   ALL          ALL
2000           NULL   17642   2000         ALL
2000           300     1476   2000         300
2000           301     1440   2000         301
2000           302     1152   2000         302
2000           400     1946   2000         400
2000           401     1596   2000         401
2000           500     1704   2000         500
2000           501     1572   2000         501
2000           600     2124   2000         600
2000           601     1932   2000         601
2000           700     2700   2000         700
2001           NULL   10717   2001         ALL
2001           300      888   2001         300
2001           301      948   2001         301
2001           302      996   2001         302
2001           400     1332   2001         400
2001           401     1105   2001         401
2001           500      948   2001         500
2001           501      936   2001         501
```

```
2001            600       936   2001          600
2001            601       792   2001          601
2001            700      1836   2001          700
```

ROLLUP 示例 4 - 下面的示例查询返回按年和季度汇总销售订单数的数据。

```
SELECT year (OrderDate) AS Year,
quarter(OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
GROUP BY ROLLUP (Year, Quarter)
ORDER BY Year, Quarter
```

下图说明了查询结果，并突出显示了结果集中的小计行。每个小计行在计算小计时所针对的列中的值为 NULL。

	Year	Quarter	Orders
1	**(NULL)**	**(NULL)**	**648**
2	**2000**	**(NULL)**	**380**
3	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
2	**2001**	**(NULL)**	**268**
3	2001	1	139
	2001	2	119
	2001	3	10

行 [1] 表示 2000 年和 2001 年两年以及所有季度的订单总数。该行的 `Year` 列和 `Quarter` 列均包含 NULL，并且是从前缀中排除的所有列对应的行。

注意： 每个 **ROLLUP** 运算返回的结果集均含有这样一行：除集合列外，每个列中均显示 NULL。该行表示针对其执行集合函数的每列的汇总。例如，如果 SUM 是正在讨论的集合函数，则该行将表示所有值的总和。

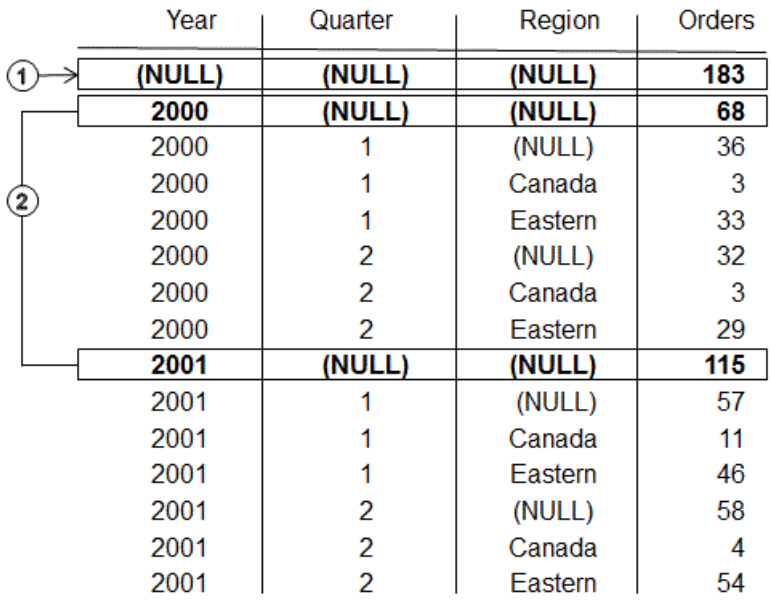
行 [2] 表示 2000 年和 2001 年各自的订单总数。这两行的 `Quarter` 列中均包含 NULL，因为该列中的值会累计，从而得出 `Year` 的小计。结果集中类似的行数取决于 **ROLLUP** 查询中出现的变量数。

标有 [3] 的其余行通过提供两年内每个季度的订单总数而提供摘要信息。

ROLLUP 示例 5 - 此 **ROLLUP** 运算示例返回一个略加复杂的结果集，该结果集按年份、季度和区域汇总销售订单数。在此示例中，只检查第一季度和第二季度以及两个选定区域（加拿大和东部地区）。

```
SELECT year(OrderDate) AS Year, quarter(OrderDate)AS Quarter,
region, COUNT(*) AS Orders
FROM SalesOrders WHERE region IN ('Canada','Eastern') AND quarter IN
(1, 2)
GROUP BY ROLLUP (Year, Quarter, Region)ORDER BY Year, Quarter, Region
```

下图说明了从上述查询得到的结果集。每个小计行在计算小计时所针对的列中的值为 NULL。



	Year	Quarter	Region	Orders
1	**(NULL)**	**(NULL)**	**(NULL)**	**183**
	2000	**(NULL)**	**(NULL)**	**68**
	2000	1	(NULL)	36
	2000	1	Canada	3
2	2000	1	Eastern	33
	2000	2	(NULL)	32
	2000	2	Canada	3
	2000	2	Eastern	29
	2001	**(NULL)**	**(NULL)**	**115**
	2001	1	(NULL)	57
	2001	1	Canada	11
	2001	1	Eastern	46
	2001	2	(NULL)	58
	2001	2	Canada	4
	2001	2	Eastern	54

行 [1] 是所有行的集合，其对应的 `Year`、`Quarter` 和 `Region` 列中包含 NULL。此行的 `Orders` 列中的值表示加拿大和东部地区在 2000 年和 2001 年第 1 季度和第 2 季度的订单总数。

标有 [2] 的行表示加拿大和东部地区每年（2000 年和 2001 年）第 1 季度和第 2 季度的销售订单总数。这些行 [2] 的值等于行 [1] 中呈现的总数。

标有 [3] 的行按区域提供有关给定年份和季度的订单总数的数据。

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	**1**	**(NULL)**	**36**
2000	1	Canada	3
2000	1	Eastern	33
2000	**2**	**(NULL)**	**32**
2000	2	Canada	3
2000	2	Eastern	29
2001	(NULL)	(NULL)	**115**
2001	**1**	**(NULL)**	57
2001	1	Canada	11
2001	1	Eastern	46
2001	**2**	**(NULL)**	**58**
2001	2	Canada	4
2001	2	Eastern	54

③

标有 [4] 的行在结果集中提供了有关每年、每季度和每个区域的订单总数的数据。

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	1	(NULL)	36
2000	**1**	**Canada**	**3**
2000	**1**	**Eastern**	**33**
2000	2	(NULL)	32
2000	**2**	**Canada**	**3**
2000	**2**	**Eastern**	**29**
2001	(NULL)	(NULL)	115
2001	1	(NULL)	57
2001	**1**	**Canada**	**11**
2001	**1**	**Eastern**	**46**
2001	2	(NULL)	58
2001	**2**	**Canada**	**4**
2001	**2**	**Eastern**	**54**

④

Group by CUBE

GROUP BY 子句中的 **CUBE** 运算符通过以多维形式将数据分组（分组表达式）来分析数据。

CUBE 需要一个有序维度列表作为参数，它让 **SELECT** 语句计算您在查询中指定的维度组的所有可能组合的小计，并生成一个结果集来显示选定列中值的所有组合的集合。

CUBE 语法：

```
SELECT … [ GROUPING (column-name) … ] … GROUP BY
[ expression [,…] | CUBE ( expression [,…] ) ]
```

GROUPING 采用列名作为参数，并返回下表中所列的布尔值：

表 2. 使用 CUBE 运算符时 GROUPING 返回的值

如果结果值是	GROUPING 将返回
由 **CUBE** 运算创建的 NULL	1 (TRUE)
指示该行是小计所在行的 NULL	1 (TRUE)
并非由 **CUBE** 运算创建	0 (FALSE)
存储的 NULL	0 (FALSE)

当维度不是同一层次的一部分时，**CUBE** 特别有用。

此 SQL 语法...	定义以下集...
`GROUP BY CUBE (A, B, C);`	(A, B, C) (A, B) (A, C) (A) (B, C) (B) (C) ( )

对 **CUBE** 运算符的限制如下：

- **CUBE** 运算符支持所有可用于 **GROUP BY** 子句的集合函数，但 **COUNT DISTINCT** 或 **SUM DISTINCT** 当前不支持 **CUBE**。
- 逆分布分析函数 **PERCENTILE_CONT** 和 **PERCENTILE_DISC** 当前不支持 **CUBE**。

- **CUBE** 只能用在 **SELECT** 语句中；**SELECT** 子查询中无法使用 **CUBE**。
- 当前不支持将 **ROLLUP**、**CUBE** 和 **GROUP BY** 列组合在同一个 GROUP BY 子句中的 **GROUPING** 规范。
- 不支持以常量表达式作为 **GROUP BY** 键。

注意： 如果 CUBE 的大小超过临时高速缓存的大小，**CUBE** 的性能将会下降。

GROUPING 可与 **CUBE** 运算符配合使用来区分存储 NULL 和 **CUBE** 创建的查询结果中的 NULL。

请参见 **ROLLUP** 运算符说明中的示例，以了解如何使用 **GROUPING** 函数解释结果。

所有 **CUBE** 运算返回的结果集均至少有一行在除集合列以外的每个列中显示 NULL。该行表示针对其执行集合函数的每列的汇总。

CUBE 示例 1 - 下列查询使用人口统计中的数据，包括州/省（地理位置）、性别、受教育水平，以及人员收入。第一个查询包含一个 **GROUP BY** 子句，该子句根据表 census 中列 state、gender 和 education 的值将查询结果组织成行组，并计算每一组的平均收入和总计。此查询只使用 **GROUP BY** 子句（不带 **CUBE** 运算符）来将行分组。

```
SELECT State, Sex as gender, DepartmentID,
COUNT(*),CAST(ROUND(AVG(Salary),2) AS NUMERIC(18,2))AS AVERAGEFROM
employees WHERE state IN ('MA' , 'CA')GROUP BY State, Sex,
DepartmentIDORDER BY 1,2;
```

以上查询的结果：

```
state  gender  DepartmentID  COUNT()     AVERAGE
-----  ------  -------   --------    -------- CA      F      200
 2         58650.00 CA       M       200           1         39300.00
```

如果要根据州/省、性别和受教育情况计算整个人口统计中的平均收入，并计算列 state、gender 和 education 的所有可能组合的平均收入，同时只遍历一次人口统计数据，请使用 **GROUP BY** 子句的 **CUBE** 扩展。例如，如果要计算所有州/省的所有女性的平均收入，或者要根据所受教育和地理位置计算人口统计中所有人的平均收入，请使用 **CUBE** 运算符。

当 **CUBE** 计算组时，系统会为计算其组的列生成 NULL。必须使用 **GROUPING** 函数确定某个 NULL 是存储在数据库中的 NULL，还是从 **CUBE** 生成的 NULL。如果已将指定列合并到更高级别的组中，则 **GROUPING** 函数返回 1。

CUBE 示例 2 - 以下查询说明了如何将 **GROUPING** 函数与 **GROUP BY CUBE** 结合使用。

```
SELECT case grouping(State) WHEN 1 THEN 'ALL' ELSE StateEND AS
c_state, case grouping(sex) WHEN 1 THEN 'ALL'ELSE Sex end AS
c_gender, case grouping(DepartmentID)WHEN 1 THEN 'ALL' ELSE
cast(DepartmentID as char(4)) endAS c_dept, COUNT(*),
CAST(ROUND(AVG(salary),2) ASNUMERIC(18,2))AS AVERAGEFROM employees
WHERE state IN ('MA' , 'CA')GROUP BY CUBE(state, sex,
DepartmentID)ORDER BY 1,2,3;
```

下面显示了此查询的结果。系统将按照查询中的指定，在小计行中将 **CUBE** 生成的指示小计行的 NULL 替换为 ALL。

```
c_state   c_gender  c_dept    COUNT()   AVERAGE
-------   --------  -------    -----     --------
ALL       ALL        200        3       52200.00
ALL       ALL        ALL        3       52200.00
ALL       F          200        2       58650.00
ALL       F          ALL        2       58650.00
ALL       M         200         1       39300.00
ALL       M         ALL         1       39300.00
CA        ALL       200         3       52200.00
CA        ALL       ALL         3       52200.00
CA        F         200         2       58650.00
CA        F         ALL         2       58650.00
CA        M         200         1       39300.00
CA        M         ALL         1       39300.00
```

CUBE 示例 3 - 在此示例中，查询返回一个汇总订单总数的结果集，然后按年和季度计算订单数小计。

注意： 随着待比较变量数量的增加，CUBE 的计算成本呈指数增长。

```
SELECT year (OrderDate) AS Year, quarter(OrderDate) AS Quarter, COUNT
(*) OrdersFROM SalesOrdersGROUP BY CUBE (Year, Quarter)ORDER BY Year,
Quarter
```

下图显示的是从查询得到的结果集。在结果集中突出显示小计行。每个小计行在计算小计所针对的列中的值为 NULL。

	Year	Quarter	Orders
1	**(NULL)**	**(NULL)**	**648**
2	**(NULL)**	**1**	**226**
	(NULL)	**2**	**196**
	(NULL)	**3**	**101**
	(NULL)	**4**	**125**
3	**2000**	**(NULL)**	**380**
	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
3	**2001**	**(NULL)**	**268**
	2001	1	139
	2001	2	119
	2001	3	10

第一个突出显示的行 [1] 表示两年和所有季度内的订单总数。`Orders` 列中的值是每个标有 [3] 的行中值的总和。它也是标有 [2] 的行中的四个值的总和。

下一组突出显示的行 [2] 按季度显示两年内的订单总数。标有 [3] 的两个行分别显示 2000 年和 2001 年的所有季度的订单总数。

分析函数

SAP Sybase IQ 既提供简单集合函数，又提供窗口集合函数，利用这些函数，可以通过一个 SQL 语句执行复杂的数据分析。

可以使用这些函数计算查询（如“What is the quarterly moving average of the Dow Jones Industrial average”或“List all employees and their cumulative salaries for each department”）的结果。可计算各个区间的移动平均值和累计总和，并且可对集合和秩进行分区以便集合计算在分区值改变时重置。在单个查询表达式的范围内，您可以定义多个不同的 OLAP 函数，每个函数都有它自己的任意分区规则。分析函数可分为两种类别：

- 简单集合函数（如 **AVG**、**COUNT**、**MAX**、**MIN** 和 **SUM**）可对数据库的一组行中的数据进行汇总。这些组是使用 **SELECT** 语句的 **GROUP BY** 子句构成的。
- 采用一个参数的一元统计集合函数包括 **STDDEV**、**STDDEV_SAMP**、**STDDEV_POP**、**VARIANCE**、**VAR_SAMP** 和 **VAR_POP**。

简单集合类别和一元集合类别汇总数据库中某一组行的数据，并且可在处理结果集时与窗口规范配合使用来计算结果集的移动窗口。

注意： 集合函数 **AVG**、**SUM**、**STDDEV**、**STDDEV_POP**、**STDDEV_SAMP**、**VAR_POP**、**VAR_SAMP** 和 **VARIANCE** 不支持二进制数据类型 `BINARY` 和 `VARBINARY`。

简单集合函数

简单集合函数（如 **AVG**、**COUNT**、**MAX**、**MIN** 和 **SUM**）可对数据库的一组行中的数据进行汇总。

这些组是使用 **SELECT** 语句的 **GROUP BY** 子句构成的。只能在选择列表中以及 **SELECT** 语句的 **HAVING** 和 **ORDER BY** 子句中使用这些集合。

注意： 除 Grouping() 函数以外，简单集合和一元集合都可以在将 <window clause> 纳入 SQL 查询规范（一个*窗口*，可在处理结果集时在概念上为该结果集创建移动窗口）的窗口函数中使用。

窗口化

OLAP 的 ANSI SQL 扩展的一个主要特征是：它具有一个名为*窗口*的结构。利用此窗口化扩展，用户可以将查询（或查询的逻辑分区）的结果集分成名为“分区”的行组，并确定要与当前行集合的行的子集。

您可以对窗口使用三类窗口函数：排名函数、行计算函数和窗口集合函数。

```
<WINDOWED TABLE FUNCTION TYPE> ::=
  <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
```

```
  | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
  | <WINDOW AGGREGATE FUNCTION>
```

窗口化扩展通过窗口名称或规范指定窗口函数的类型，并应用于单个查询表达式范围内的分区结果集。窗口分区是由查询返回的行的子集，由一个特殊的 **OVER** 子句中的一个或多个列定义：

```
olap_function() OVER (PARTITION BY col1, col2...)
```

使用窗口化操作，可以建立信息，诸如在行的分区中排列每行、在某个分区内的行中分布值，以及类似操作。利用窗口化操作，还可对数据计算移动平均值和总数，从而增强对数据及其对操作的影响进行评估的能力。

OLAP 窗口的三个基本部分

OLAP 窗口包含三个基本方面：窗口分区、窗口排序和窗口构架。在任何时刻，每一方面都会对窗口中显示的特定数据行产生重大影响。同时，OLAP **OVER** 子句可利用三项独特的功能将 OLAP 函数与其它分析函数或报告函数区分开：

- 定义窗口分区（**PARTITION BY** 子句）。
- 对分区中的行进行排序（**ORDER BY** 子句）。
- 定义窗口构架（ROWS/RANGE 规范）。

若要指定多个窗口函数，并避免出现冗余窗口定义，可以指定 OLAP 窗口规范的名称。在这种用法中，关键字 **WINDOW** 后面至少跟有一个窗口定义，窗口定义之间用逗号分隔开。窗口定义包含窗口在查询中使用的名称以及窗口规范中的详细信息，利用它，可以定义窗口分区、窗口排序和窗口构架：

```
<WINDOW CLAUSE> ::= <WINDOW DEFINITION LIST>
```

```
<WINDOW DEFINITION LIST> ::=
  <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
  } . . . ]
```

```
<WINDOW DEFINITION> ::=
  <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>
```

```
<WINDOW SPECIFICATION DETAILS> ::=
  [ <EXISTING WINDOW NAME> ]
  [ <WINDOW PARTITION CLAUSE> ]
  [ <WINDOW ORDER CLAUSE> ]
  [ <WINDOW FRAME CLAUSE> ]
```

对于窗口分区中的每一行，用户可以定义窗口构架，这可能会改变用于对分区的当前行执行任何计算的特定行范围。当前行提供了可用于确定窗口构架的起点和终点的参照点。

窗口规范可以基于一定数量的物理行（使用定义 ROWS 的窗口构架单元的窗口规范）或数值的逻辑区间（使用定义 RANGE 的窗口构架单元的窗口规范）。

在 OLAP 窗口化操作中，可以使用以下函数类别：

- 排名函数

- 窗口化集合函数
- 统计集合函数
- 分布函数

窗口分区

窗口分区是指使用 **PARTITION BY** 子句拆分用户指定的结果集（输入行）。

分区由一个或多个用逗号分隔的值表达式定义。分区数据也是隐式排序的，缺省排序顺序是升序 (ASC)。

```
<WINDOW PARTITION CLAUSE> ::=
  PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

如果未指定窗口分区子句，则会将输入作为单个分区处理。

注意： 术语*分区*像在分析函数中使用时一样，仅仅是指使用 **PARTITION BY** 子句将结果行集拆分开。

可根据任意表达式定义窗口分区。另外，因为窗口分区在 GROUPING 之后发生（如果指定了 **GROUP BY** 子句），所以，任何集合函数（例如 **SUM**、**AVG** 和 **VARIANCE**）的结果都可以用于分区表达式。因此，除了使用 **GROUP BY** 和 **ORDER BY** 子句外，利用分区也可以执行分组和排序操作；例如，您可以构造通过集合函数来计算集合函数（如特定数量的最大 **SUM**）的查询。

即使没有 **GROUP BY** 子句，也可以指定 **PARTITION BY** 子句。

窗口排序

窗口排序是指使用 window order 子句（包含一个或多个用逗号分隔的值表达式）排列每个窗口分区中的结果（行）。

如果未指定 window order 子句，则可以按任意顺序处理输入行。

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

OLAP window order 子句不同于 **ORDER BY** 子句，后者可以附加到非窗口查询表达式中。

例如，OLAP 函数中的 **ORDER BY** 子句通常定义用于对窗口分区中的行进行排序的表达式；但是，您可以使用不带 **PARTITION BY** 子句的 **ORDER BY** 子句，在这种情况下，排序规范可确保使用 OLAP 函数来对中间结果集进行有意义的（并且是预期的）排序。

排序规范是 OLAP 函数的排列系列的前提条件；用于标识排列值的测量标准的不是函数参数本身，而是 **ORDER BY** 子句。如果是 OLAP 集合，则一般不需要 **ORDER BY** 子句，但该子句是定义窗口构架的前提条件。这是因为，必须先对分区行进行排序，然后才可以为每个构架计算相应的集合值。

ORDER BY 子句包括用于定义升序排序和降序排序的语义，以及 NULL 的处理规则。缺省情况下，OLAP 函数采用升序，而最小测量值的排名为 1。

虽然此行为与 **ORDER BY** 子句（位于 **SELECT** 语句的末尾）的缺省行为一致，但对于大多数有序计算，它都是与直觉相反的。OLAP 计算通常需要降序，而最大测量值的排名为 1；必须使用带 DESC 关键字的 **ORDER BY** 子句显式声明此要求。

注意： 排名函数需要 <window order clause>，因为它们是仅针对排序输入定义的。就像使用 <query specification> 中的 <order by clause> 一样，缺省排序顺序是升序。

使用 RANGE 的 <window frame unit> 时也要求存在 <window order clause>。如果是 RANGE，则 <window order clause> 只能包括一个表达式。

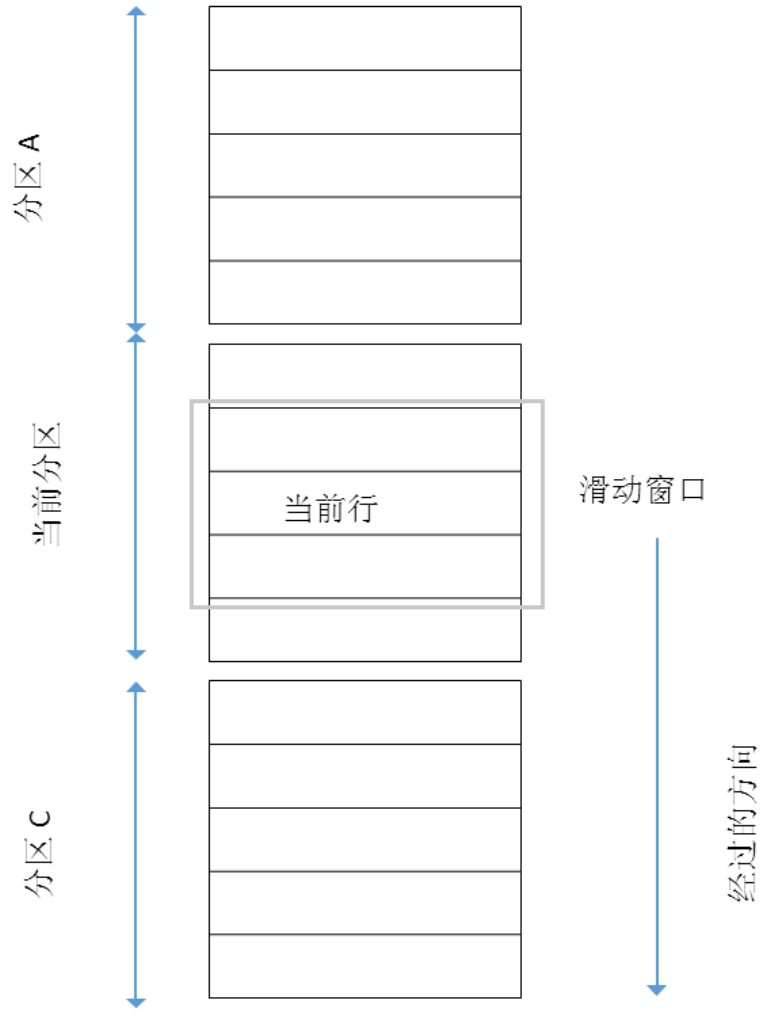
窗口构架

对于非秩集合 OLAP 函数，可以使用 window frame 子句定义一个窗口构架，该子句指定窗口的开始位置和结束位置（相对于当前行）。

```
<WINDOW FRAME CLAUSE> ::=
  <WINDOW FRAME UNIT>
  <WINDOW FRAME EXTENT>
```

计算此 OLAP 函数时针对的是移动构架的内容而不是整个分区的固定内容。根据定义，分区具有开始行和结束行，而窗口构架从分区的起点滑至终点。

图 3： 含分区输入的三行移动窗口



UNBOUNDED PRECEDING 和 FOLLOWING
窗口构架可由未受限制的集合组（扩展回到分区的开始位置 (UNBOUNDED PRECEDING) 或扩展到分区的结束位置 (UNBOUNDED FOLLOWING)）定义。

UNBOUNDED PRECEDING 包括分区中当前行之前的所有行，对于这些行，可使用 ROWS 或 RANGE 指定。UNBOUNDED FOLLOWING 包括分区中当前行之后的所有行，对于这些行，可使用 ROWS 或 RANGE 指定。

值 FOLLOWING 指定当前行后面的行的范围或数量。如果指定了 ROWS，则该值是一个表示行数的正整数。如果指定了 RANGE，则窗口包括所有小于当前行加指定数值的行。如果是 RANGE，则窗口值的数据类型必须与 **ORDER BY** 子句的排序键表达式的类型相当。只能有一个排序键表达式，并且排序键表达式的数据类型必须允许相加。

值 PRECEDING 指定当前行前面的行的范围或数量。如果指定了 ROWS，则该值是一个表示行数的正整数。如果指定了 RANGE，则窗口包括所有小于当前行减去指定数值的行。如果是 RANGE，则窗口值的数据类型必须与 **ORDER BY** 子句的排序键表达式的类型相当。只能有一个排序键表达式，并且排序键表达式的数据类型必须允许相减。如果第一个绑定组为 CURRENT ROW 或值 FOLLOWING，则不能在第二个绑定组中指定此子句。

组合 BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING 提供整个分区的集合，且不需要构造到分组查询的连接。整个分区的集合也称为报告聚合。

CURRENT ROW 概念
在物理集合组中，根据行相对于当前行的位置来包括或排除行（通过统计相邻行的数目）。当前行仅仅是查询的中间结果中下一行的参照。如果当前行前进，则会根据窗口中的新行集重新计算窗口。对于窗口中包括的当前行，没有任何要求。

如果未指定 window frame 子句，则缺省窗口构架取决于是否指定了 window order 子句：

- 如果窗口规范包含 window order 子句，且窗口的起点是 **UNBOUNDED PRECEDING**，终点是 **CURRENT ROW**，则定义适合用于计算累计值的大小可变的窗口。
- 如果窗口规范不包含 window order 子句，且窗口的起点是 **UNBOUNDED PRECEDING**，终点是 **UNBOUNDED FOLLOWING**，则定义大小固定的窗口（不考虑当前行）。

注意： window frame 子句不能与排名函数一起使用。

您还可以通过指定基于行（行规范）或基于值（范围规范）的窗口构架单元来定义窗口。

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

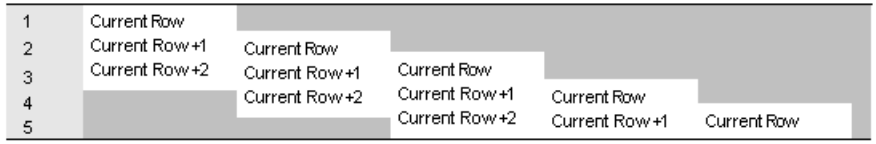
```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW FRAME
BETWEEN>
```

当窗口构架范围指定 **BETWEEN** 时，它会显式提供窗口构架的开始位置和结束位置。

如果窗口构架范围仅指定这两个值中的一个，则另一个值缺省为 **CURRENT ROW**。

基于行的窗口构架 - 在示例中，行 [1] 至 [5] 表示分区；随着 OLAP 窗口构架的向前滑动，每个行都可成为当前行。构架是指“Between Current Row And 2 Following”，因此每个构架都最多包括三行，最少包括一行。当构架到达分区的末尾时，只包括当前行。带阴影的区域表示每一步从构架中排除的行。

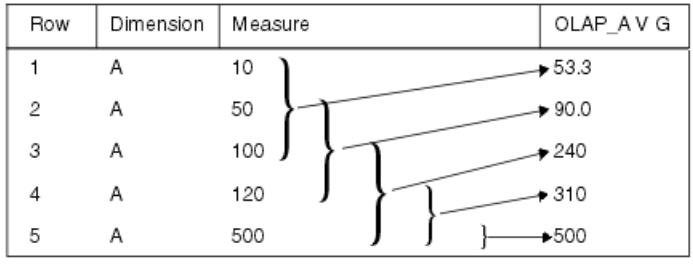
图 4： 基于行的窗口构架



窗口构架实施下列规则：

- 当行 [1] 为当前行时，系统将排除行 [4] 和行 [5]。
- 当行 [2] 为当前行时，将排除行 [5] 和行 [1]。
- 当行 [3] 为当前行时，系统将排除行 [1] 和行 [2]。
- 当行 [4] 为当前行时，将排除行 [1]、[2] 和 [3]。
- 当行 [5] 为当前行时，系统将排除行 [1]、[2]、[3] 和 [4]。

下图将这些规则应用到一组特定值，并显示将为每一行计算的 OLAP **AVG** 函数。滑动计算生成区间为三行（或更少）的移动平均值，具体取决于哪一行是当前行：



下面的示例说明了一个滑动窗口：

```
SELECT dimension, measure,
  AVG(measure) OVER(partition BY dimension
    ORDER BY measure
    ROWS BETWEEN CURRENT ROW and 2 FOLLOWING)
    AS olap_avg
FROM ...
```

平均值按以下方式计算：

- 行 [1] = (10 + 50 + 100)/3

- 行 [2] = (50+ 100 + 120)/3
- 行 [3] = (100 + 120 + 500)/3
- 行 [4] = (120 + 500 + NULL)/3
- 行 [5] =（500 + NULL + NULL）/3

将为结果集中的所有后续分区执行相似计算（如 B、C 等）。

如果当前窗口中没有行，则结果为 NULL，但对于 **COUNT** 则例外。

ROWS

窗口构架单元 ROWS 通过指定当前行前面或后面的行的数量来定义窗口，当前行充当确定窗口开始位置和结束位置的参照点。

每个分析计算都基于分区中的当前行。要为以行表示的窗口生成确定性结果，排序表达式应该是唯一的。

所有窗口构架的参照点都是当前行。使用 SQL/OLAP 语法，可以将基于行的窗口构架定义为当前行前面和/或后面的任意数量的行。

以下列表说明了常见的窗口构架单元示例：

- Rows Between Unbounded Preceding and Current Row - 指定以每个分区的开始位置作为起点并以当前行作为终点的窗口，经常用于构造计算累计结果（如累计总和）的窗口。
- Rows between unbounded preceding and unbounded following - 指定跨整个分区的固定窗口（不考虑当前行）。因此，窗口集合函数的值在分区的每一行中都是相同的。
- Rows between 1 preceding and 1 following - 指定跨三个相邻行（当前行前后各一行）的大小固定的移动窗口。您可以使用此窗口构架单元执行计算，例如，计算 3 天或 3 个月的移动平均值。

  请注意，由于窗口值之间存在间距，因此在使用 ROWS 时可能会生成没有意义的结果。如果值集是不连续的，请考虑使用 RANGE 而不是 ROWS，因为基于 RANGE 的窗口定义会自动处理含重复值的相邻行，并且，如果范围中存在间距，则不会包括任何其它行。

  注意： 如果是移动窗口，则假设在输入中，包含 NULL 的行位于第一行之前、最后一行之后。这意味着，在一个包含 3 行的移动窗口中，输入中最后一行（当前行）的计算将包括紧挨在它前面的一行和 NULL。
- Rows between current row and current row - 将窗口限制为仅含当前行。
- Rows Between 1 Preceding and 1 Preceding - 指定仅含上一行（相对于当前行而言）的单行窗口。此构造与仅基于当前行计算值的另一窗口函数组合使用，使得您可以轻松地计算相邻行间的增量（即值的差额）。

RANGE

基于范围的窗口构架 - SQL/OLAP 语法支持另一种窗口构架，这种构架的限制是根据一组基于值（或基于范围）的行定义的，而不是根据一系列特定行定义的。

基于值的窗口构架定义窗口分区中包含特定范围的数值的行。OLAP 函数的 **ORDER BY** 子句定义应用范围规范的数值列（相对于当前行的该列中的值）。范围规范与行规范使用的语法相同，但是语法的解释方式不同。

窗口构架单元 RANGE 可定义一个窗口构架，通过查找排序列的值在指定值范围内的行（相对于当前行），可以确定该构架的内容。这称为窗口构架的逻辑偏移，您可以使用常量（如“3 preceding”）或任何计算结果为数值常量的表达式指定该偏移。如果所使用的窗口是用 RANGE 定义的，则 **ORDER BY** 子句中只能有一个数值表达式。

注意： **ORDER BY** 键必须是 RANGE 窗口构架中的数值数据。

例如，可将构架定义为一组含 *year* 值（当前行的年份之前或之后的数年）的行：

```
ORDER BY year ASC range BETWEEN 1 PRECEDING AND CURRENT ROW
```

词语 1 PRECEDING 表示当前行的 *year* 值减 1。

这种范围规范是包含性的。如果当前行的 *year* 值为 2000，则窗口分区中所有 year 值为 2000 和 1999 的行都符合成为构架内容的条件，无论这些行在分区中处于哪个物理位置都是如此。用于包括和排除基于值的行的规则与应用于基于行的构架的规则非常不同，后者完全依赖于行的物理顺序。

如果是在 OLAP **AVG()** 计算环境中，则下面的部分结果集可进一步说明基于值的窗口构架的概念。重申一遍，构架包括符合以下条件的行：

- 与当前行具有相同的 year 值
- year 值等于当前行减去 1

Row	Dimension	Year	Measure	Olap_avg
1	A	1999	10000	10000
2	A	2001	5000	3000
3	A	2001	1000	3000
4	A	2002	12000	5250
5	A	2002	3000	5250

以下查询说明了基于范围的窗口定义：

```
SELECT dimension, year, measure,
  AVG(measure) OVER(PARTITION BY dimension
    ORDER BY year ASC
    range BETWEEN CURRENT ROW and 1 PRECEDING)
    as olap_avg
FROM ...
```

平均值按以下方式计算：

- 行 [1] = 1999；排除行 [2] 至 [5]；AVG = 10,000/1

- 行 [2] = 2001；排除行 [1]、[4] 和 [5]；AVG = 6,000/2
- 行 [3] = 2001；排除行 [1]、[4] 和 [5]；AVG = 6,000/2
- 行 [4] = 2002；排除行 [1]；AVG = 21,000/4
- 行 [5] = 2002；排除行 [1]；AVG = 21,000/4

基于值的构架的升序和降序 - 用于基于值的窗口构架的 OLAP 函数的 **ORDER BY** 子句不仅标识范围规范所基于的数值列，而且还声明 **ORDER BY** 值的排序顺序。以下规范受它之前的排序顺序（ASC 或 DESC）约束：

```
RANGE BETWEEN CURRENT ROW AND n FOLLOWING
```

规范 *n* FOLLOWING 表示：

- 如果分区按缺省升序 (ASC) 排序，则加上 *n*
- 如果分区按降序 (DESC) 排序，则减去 *n*

例如，假设 `year` 列包含四个不同的值（从 1999 到 2002）。下表左侧显示的是这此值的缺省升序排列结果，而右侧显示的是这些值的降序排列结果：

ORDER BY year ASC	ORDER BY year DESC
1999	2002
2000	2001
2001	2000
2002	1999

如果当前行是 1999 并且按照如下所示指定构架，则包含值 1999 和 1998 的行（在表中不存在）会包括在构架中：

```
ORDER BY year DESC range BETWEEN CURRENT ROW and 1 FOLLOWING
```

注意： **ORDER BY** 值的排序顺序是测试哪些行可以包括在基于值的构架中的关键部分；仅仅依赖于数值，无法确定是排除还是包括行。

使用未受限制的窗口 - 以下查询会生成一个结果集，结果集中包含所有产品以及所有产品的总量：

```
SELECT id, description, quantity,
  SUM(quantity) OVER () AS total
FROM products;
```

计算相邻行之间的增量 - 如果使用两个窗口（一个在当前行上，另一个在上一行上），则可以直接计算相邻行之间的增量（即变化）。

```
SELECT EmployeeID, Surname, SUM(salary)
OVER(ORDER BY BirthDate rows between current row and current row)
AS curr, SUM(Salary)
OVER(ORDER BY BirthDate rows between 1 preceding and 1 preceding)
AS prev, (curr-prev) as delta
FROM Employees
WHERE State IN ('MA', 'AZ', 'CA', 'CO') AND DepartmentID>10
ORDER BY EmployeeID, Surname;
```

以上查询的结果：

```
EmployeeID  Surname      curr         prev         delta
---------   ---------    --------     ---------    -----
```

```
148         Jordan       51432.000191
209         Bertrand              29800.000
39300.000             -9500.000278
225         Melkisetian     48500.000       42300.000
6200.000299
657         Overbey                   39300.000
41700.750             -2400.750318
902         Crow                      41700.750
45000.000             -3299.250586
949         Coleman                   42300.000
46200.000             -3900.000690
1053        Poitras                   46200.000
29800.000             16400.000703
1090       Martinez               55500.800           51432.000
4068.800949
1154        Savarino                  72300.000
55500.800             16799.2001101
1420        Preston      37803.000             48500.000
-10697.0001142
1507        Clark        45000.000             72300.000
-27300.000
```

虽然使用窗口函数 **SUM()**，但考虑到窗口的指定方式，总和只包含当前行或上一行的薪水值。另外，结果中第一行的 `prev` 值为 NULL，因为第一行没有前一行；因此，`delta` 也为 NULL。

在上面的每个示例中，用于 **OVER()** 子句的函数是 **SUM()** 集合函数。

显式窗口子句和行内窗口子句

SQL OLAP 提供了两种方式来在查询中指定窗口：

- 使用 **HAVING** 子句后面的显式窗口子句，可以定义窗口。您可以在调用 OLAP 函数时通过指定名称来引用这些窗口子句定义的窗口，如：

  ```
  SUM ( ...) OVER w2
  ```

- 使用行内窗口规格，可以通过查询表达式的 **SELECT** 列表定义窗口。使用此功能，可以在跟在 **HAVING** 子句后面的窗口子句中定义窗口，然后利用窗口函数调用按名称引用它们，或将它们与函数调用一起定义。

  注意： 如果使用行内窗口规格，则无法命名窗口。一个 **SELECT** 列表中的两个或更多个使用相同窗口的窗口函数调用都必须引用在窗口子句中定义的命名窗口，或者必须以冗余方式定义它们的行内窗口。

窗口 函数示例 - 下面的示例显示了一个窗口函数。 查询返回一个结果集，该结果集按部门将数据分区， 然后提供雇员薪水的累计汇总 （从在公司工作时间最长的雇员开始）。 结果集仅包括住在马萨诸塞州的雇员。 列 `sum_salary` 提供雇员薪水的累计总计。

```
SELECT DepartmentID, Surname, StartDate, Salary, SUM(Salary) OVER
(PARTITION BY DepartmentID ORDER BY startdate
rows between unbounded preceding and current row)
AS sum_salary FROM Employees
```

```
WHERE State IN ('CA') AND DepartmentID IN (100, 200)
ORDER BY DepartmentID;
```

以下结果集是按部门分区的。

```
DepartmentID  Surname   start_date  salary     sum_salary
------------  --------  ----------  ---------  ----------
200           Overbey   1987-02-19  39300.000   39300.000
200           Savarino  1989-11-07  72300.000  111600.000
200           Clark     1990-07-21  45000.000  156600.000
```

排名函数

使用排名函数，可以按排列顺序编译数据集中值的列表，以及编写执行请求的单语句 SQL 查询（如“Name the top 10 products shipped this year by total sales”或“Give the top 5% of salespersons who sold orders to at least 15 different companies”）。

SQL/OLAP 定义五个属于排名函数类别的函数：

```
<RANK FUNCTION TYPE> ::=
  RANK | DENSE_RANK | PERCENT_RANK | ROW_NUMBER | NTILE
```

使用排名函数，可以根据在查询中指定的顺序为结果集中的每一行计算秩值。例如，一名销售经理可能需要确定公司中的最领先或最落后的销售人员，业绩最好或最差的销售区域，或者销售情况最好或最差的产品。排名函数可以提供此信息。

RANK

RANK 函数返回一个数字，该数字表示当前行在行分区中各行间的秩（由 **ORDER BY** 子句定义）。

分区中第一行的秩为 1，包含 25 个行的分区的最后一个秩为 25。**RANK** 被指定为语法转换，这表示实施可以选择将 **RANK** 实际转换为它的等同项，否则，它仅能返回与转换返回的结果等同的结果。

在下面的示例中，ws1 表示定义名为 w1 的窗口的窗口规范。

```
RANK() OVER ws
```

等效于：

```
( COUNT (*) OVER ( ws RANGE UNBOUNDED PRECEDING )
- COUNT (*) OVER ( ws RANGE CURRENT ROW ) + 1 )
```

RANK 函数的转换使用逻辑集合 (RANGE)。因此，两个或更多个绑定的（即在排序列中具有相等值）的记录将具有相同的秩。分区中下一个具有不同值的组的秩将比绑定行的秩至少大 1。例如，如果有一些行的排序列值为 10、20、20、20、30，则第一行的秩为 1，第二行的秩为 2，第三行和第四行的秩也为 2，但第五行的秩为 5。没有秩为 3 或 4 的行。此算法有时也称为稀疏排名。

RANK 函数 [分析]
排列组中的项目。

语法

```
RANK ( ) OVER ( [ PARTITION BY ] ORDER BY expression [ ASC | DESC ] )
```

参数

参数	描述
expression	排序规范，可以是涉及列引用、集合的任意有效表达式，也可以是调用这些项目的表达式。

返回
INTEGER

注释
RANK 是 rank 分析函数。行 R 的秩是指位于 R 前面但不是 R 的对等行的行数。如果两个或更多个行在 **OVER** 子句指定的组中或者在整个结果集中是不重复的，则连续秩编号中存在一个或多个空位。**RANK** 和 **DENSE_RANK** 之间的区别是：当存在并列排名时，**DENSE_RANK** 不会保留空位，而 **RANK** 会保留空位。

RANK 需要使用 **OVER (ORDER BY)** 子句。**ORDER BY** 子句指定要执行排序的参数以及每个组中行的排列顺序。此 **ORDER BY** 子句只在 **OVER** 子句中使用，而不是 **SELECT** 的 **ORDER BY**。不允许排序查询中的任何集合函数指定 DISTINCT。

OVER (ORDER BY) 子句中的 **PARTITION BY** 窗口分区子句是可选的。

ASC 或 DESC 参数用于指定升序或降序排序序列。升序是缺省值。

OVER 子句表示函数对查询结果集进行操作。结果集是在对 **FROM**、**WHERE**、**GROUP BY** 和 **HAVING** 子句求值完成之后返回的行。**OVER** 子句定义要包括在 rank 分析函数计算中的行数据集。

RANK 只能在 **SELECT** 或 **INSERT** 语句的选择列表中或者 **SELECT** 语句的 **ORDER BY** 子句中使用。**RANK** 可以在视图中或联合中使用。您不能在子查询中、**HAVING** 子句中，或者 **UPDATE** 或 **DELETE** 语句的选择列表中使用 **RANK** 函数。每个查询仅允许使用一个 rank 分析函数。

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - Adaptive Server 或 SQL Anywhere 不支持。

示例
下面的语句说明了 **RANK** 函数的用法：

```
SELECT Surname, Sex, Salary, RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA', 'AZ') AND DepartmentID IN (200, 300)
ORDER BY Sex, Salary DESC;
```

以上查询的结果：

```
Surname           Sex      Salary     RANK
-------           ---      ------     ----
Savarino           F        72300.000  1
Jordan             F        51432.000  2
Clark              F        45000.000  3
Coleman            M        42300.000  1
Overbey            M        39300.000  2
```

DENSE_RANK

DENSE_RANK 会返回无间隔的秩值。

绑定行的值仍旧相等，但行的秩表示在排序列中具有相等值的行的集群位置，而不是各个行的位置。正如在 **RANK** 示例中，行排序列值是 10、20、20、20、30，第一行的秩仍旧是 1，而第二行的秩仍旧是 2，第三行和第四行的秩也是相同的。但是，最后一行的秩是 3 而不是 5。

DENSE_RANK 也是通过语法转换计算的。

```
DENSE_RANK() OVER ws
```

等效于：

```
COUNT ( DISTINCT ROW ( expr_1, . . ., expr_n ) )
 OVER ( ws RANGE UNBOUNDED PRECEDING )
```

在上面的示例中，*expr_1* 至 *expr_n* 代表窗口 w1 的排序规范列表中值表达式的列表。

DENSE_RANK 函数 [分析]

排列组中的项目。

语法

```
DENSE_RANK () OVER ( ORDER BY  expression [ ASC | DESC ] )
```

参数

表 3. 参数

参数	描述
表达式	排序规范，可以是涉及列引用、集合的任意有效表达式，也可以是调用这些项目的表达式。

返回

INTEGER

注释

DENSE_RANK 是 rank 分析函数。行 R 的密集排名是指位于该行前面（包括行 R）并且在 **OVER** 子句指定的组中或者在整个结果集中保持不重复的行数。**DENSE_RANK** 和 **RANK** 之间的区别是：当存在并列排名时，**DENSE_RANK** 不会保留空位，而 **RANK** 会保留空位。

DENSE_RANK 需要使用 **OVER (ORDER BY)** 子句。**ORDER BY** 子句指定要执行排序的参数以及每个组中行的排列顺序。此 **ORDER BY** 子句只在 **OVER** 子句中使用，而不是 **SELECT** 的 **ORDER BY**。不允许排序查询中的任何集合函数指定 **DISTINCT**。

OVER 子句表示函数对查询结果集进行操作。结果集是在对 **FROM**、**WHERE**、**GROUP BY** 和 **HAVING** 子句求值完成之后返回的行。**OVER** 子句定义要包括在 rank 分析函数计算中的行数据集。

ASC 或 DESC 参数用于指定升序或降序排序序列。升序是缺省值。

DENSE_RANK 只能在 **SELECT** 或 **INSERT** 语句的选择列表中或者 **SELECT** 语句的 **ORDER BY** 子句中使用。**DENSE_RANK** 可以在视图中或联合中使用。您不能在子查询中、HAVING 子句中，或者 **UPDATE** 或 **DELETE** 语句的选择列表中使用 **DENSE_RANK** 函数。每个查询仅允许使用一个 rank 分析函数。

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 不受 Adaptive Server 或 SQL Anywhere 支持。

示例

以下语句说明了 **DENSE_RANK** 函数的用法：

```
SELECT s_suppkey, DENSE_RANK()
OVER ( ORDER BY ( SUM(s_acctBal) DESC )
AS rank_dense FROM supplier GROUP BY s_suppkey;

s_suppkey        sum_acctBal       rank_dense
supplier#011     200,000            1
supplier#002     200,000            1
supplier#013     123,000            2
supplier#004     110,000            3
supplier#035     110,000            3
supplier#006     50,000             4
supplier#021     10,000             5
```

PERCENT_RANK

PERCENT_RANK 函数计算秩的百分比，而不是分数，并返回一个介于 0 和 1 之间的小数值。

PERCENT_RANK 返回行的相对秩，该数字表示当前行在其所在窗口分区中的相对位置。例如，如果分区中包含 10 个在排序列中具有不同值的行，则第三行的 **PERCENT_RANK** 值为 0.222 …，原因是涵盖了分区第一行后面的 2/9（22.222…%）

个行。行的 **PERCENT_RANK** 是指行的 **RANK** 减去一，再除以分区中的行数减去一，如以下示例中所示（其中 "ANT" 代表近似数值类型，如 REAL 或 DOUBLE PRECISION）。

```
PERCENT_RANK() OVER ws
```

等效于：

```
CASE
  WHEN COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
  PRECEDING AND UNBOUNDED FOLLOWING ) = 1
  THEN CAST (0 AS ANT)
  ELSE
   ( CAST ( RANK () OVER ( ws ) AS ANT ) -1 /
   ( COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
   PRECEDING AND UNBOUNDED FOLLOWING ) - 1 )
END
```

PERCENT_RANK 函数 [分析]

根据 **ORDER BY** 子句的定义，计算查询返回的一行相对于该查询返回的其它行的（小数）位置。

它返回介于 0 和 1 之间的小数值。

语法

PERCENT_RANK ( ) **OVER** (**ORDER BY** *expression* [**ASC** | **DESC**])

参数

参数	描述
expression	排序规范，可以是涉及列引用、集合的任意有效表达式，也可以是调用这些项目的表达式。

返回

PERCENT_RANK 函数返回介于 0 和 1 之间的 DOUBLE 值。

注释

PERCENT_RANK 是 rank 分析函数。行 R 的百分比秩是指在 **OVER** 子句中指定的组中某个行的秩减去一，再除以在 **OVER** 子句中指定的组中的总行数减去一。**PERCENT_RANK** 返回介于 0 和 1 之间的值。第一行的百分比秩为零。

行的 **PERCENT_RANK** 的计算公式为

```
(Rx - 1) / (NtotalRow - 1)
```

其中 *Rx* 是组中某行的秩位置，*NtotalRow* 是 **OVER** 子句指定的组中的总行数。

PERCENT_RANK 需要使用 **OVER (ORDER BY)** 子句。**ORDER BY** 子句指定要执行排序的参数以及每个组中行的排列顺序。此 **ORDER BY** 子句只在 **OVER** 子句中使用，而不是 **SELECT** 的 **ORDER BY**。不允许排序查询中的任何集合函数指定 **DISTINCT**。

OVER 子句表示函数对查询结果集进行操作。结果集是在对 **FROM**、**WHERE**、**GROUP BY** 和 **HAVING** 子句求值完成之后返回的行。**OVER** 子句定义要包括在 rank 分析函数计算中的行数据集。

ASC 或 DESC 参数用于指定升序或降序排序序列。升序是缺省值。

PERCENT_RANK 只能在 **SELECT** 或 **INSERT** 语句的选择列表中或者 **SELECT** 语句的 **ORDER BY** 子句中使用。**PERCENT_RANK** 可以在视图中或联合中使用。您不能在子查询中、**HAVING** 子句中，或者 **UPDATE** 或 **DELETE** 语句的选择列表中使用 **PERCENT_RANK** 函数。每个查询仅允许使用一个 rank 分析函数。

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 不受 Adaptive Server 或 SQL Anywhere 支持。

示例
以下语句说明了 **PERCENT_RANK** 函数的用法：

```
SELECT s_suppkey, SUM(s_acctBal) AS sum_acctBal,
PERCENT_RANK() OVER ( ORDER BY SUM(s_acctBal) DESC )
AS percent_rank_all FROM supplier GROUP BY s_suppkey;

s_suppkey        sum_acctBal       percent_rank_all
supplier#011     200000            0
supplier#002     200000            0
supplier#013     123000            0.3333
supplier#004     110000            0.5
supplier#035     110000            0.5
supplier#006     50000             0.8333
supplier#021     10000             1
```

ROW_NUMBER
ROW_NUMBER 函数为每一行返回一个唯一行号。

如果您定义窗口分区，**ROW_NUMBER** 就会在每个分区中从 1 开始对行进行编号，每行递增 1。如果您不指定窗口分区，**ROW_NUMBER** 就会对整个结果集进行编号，从 1 到表的总基数。

ROW_NUMBER 函数语法为：

```
ROW_NUMBER() OVER ([PARTITION BY
window partition] ORDER BY
window ordering)
```

ROW_NUMBER 不需要参数，但您必须指定括号。

PARTITION BY 子句是可选的。**OVER** (**ORDER_BY**) 子句不能包含窗口构架 **ROWS**/**RANGE** 规范。

ROW_NUMBER 函数 [分析]
一个排名函数，它为窗口分区中的每一行返回一个唯一行号，以便在每个窗口分区的开始位置对行重新开始编号。

如果窗口分区不存在，则该函数按从 1 到表基数的顺序对结果集中的行进行编号。

语法

```
ROW_NUMBER() OVER ([PARTITION BY window partition] ORDER BY window ordering)
```

参数

参数	描述
window partition	（可选）一个或多个用逗号分隔的值表达式，表示您希望如何分隔一组结果行。
window ordering	如果指定窗口分区，则定义用于对窗口分区中的行进行排序的表达式；如果未指定窗口分区，则定义用于对结果集中的行进行排序的表达式。

注释
ROW_NUMBER 函数需要 **OVER (ORDER_BY)** 窗口规范。**OVER (ORDER_BY)** 子句中的窗口分区子句是可选的。**OVER (ORDER_BY)** 子句不能包含窗口构架 **ROWS**/**RANGE** 规范。

标准和兼容性

- SQL - 符合 ISO/ANSI SQL 标准。SQL/OLAP 功能 T611。

示例
以下示例返回雇员表中的薪水数据，按部门 ID 对结果集进行分区，并根据雇用开始日期对数据进行排序。**ROW_NUMBER** 函数为每一行指派一个行号，并对每个窗口分区中的行重新开始编号：

```
SELECT DepartmentID dID, StartDate, Salary,
ROW_NUMBER()OVER(PARTITION BY dID ORDER BY StartDate) FROM  Employees
ORDER BY 1,2;
```

返回的结果集为：

```
dID         StartDate    Salary      Row_number()
=========  ===========  ==========  =============
100         1986-10-14  42,998.000     1
100         1987-07-23  39,875.500     2
100         1988-03-23  37,400.000     3
100         1989-04-20  42,500.000     4
100         1990-01-15  42,100.000     5
200         1985-02-03  38,500.000     1
200         1987-02-19  39,300.000     2
200         1988-11-22  39,800.000     3
```

```
200          1989-06-01  34,892.000     4
200          1990-05-13  33,890.000     5
200          1990-07-11  37,803.000     6
```

排名示例

下面是一些排名函数示例：

排名示例 1 - 下面的 SQL 查询查找加利福尼亚州的男女雇员并按薪水以降序排列他(她）们。

```
SELECT Surname, Sex, Salary, RANK() OVER (
ORDER BY Salary DESC) as RANK FROM Employees
WHERE State IN ('CA') AND DepartmentID =200
ORDER BY Salary DESC;
```

以上查询的结果：

```
Surname       Sex         Salary     RANK
-------       ---         ------     ----
Savarino       F           72300.000  1
Clark          F           45000.000  2
Overbey        M           39300.000  3
```

排名示例 2 - 使用上一示例中的查询，您可以按性别划分数据，从而更改数据。下面的示例按薪水以降序排列雇员并按性别将雇员分区：

```
SELECT Surname, Sex, Salary, RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA', 'AZ') AND DepartmentID IN (200, 300)
ORDER BY Sex, Salary DESC;
```

以上查询的结果：

```
Surname            Sex       Salary      RANK
-------            ---       ---------   ----
Savarino            F         72300.000  1
Jordan              F         51432.000  2
Clark               F         45000.000  3
Coleman             M         42300.000  1
Overbey             M         39300.000  2
```

排名示例 3 - 此示例查找加利福尼亚州和德克萨斯州的一些女雇员并根据薪水以降序排列她们。**PERCENT_RANK** 函数按降序提供累计总计。

```
SELECT Surname, Salary, Sex, CAST(PERCENT_RANK() OVER
(ORDER BY Salary DESC) AS numeric (4, 2)) AS RANK
FROM Employees WHERE State IN ('CA', 'TX') AND Sex ='F'
ORDER BY Salary DESC;
```

以上查询的结果：

```
Surname          salary    sex            RANK
---------       ---------   ---   ----------
Savarino        72300.000   F            0.00
Smith           51411.000   F            0.33
```

```
Clark          45000.000   F           0.66
Garcia         39800.000   F           1.00
```

排名示例 4 - 您可以使用 **PERCENT_RANK** 函数在数据集中查找最高或最低的百分点。此查询会返回其薪水在数据集的最高的五个百分点之内的男雇员。

```
SELECT * FROM (SELECT Surname, Salary, Sex,
CAST(PERCENT_RANK() OVER (ORDER BY salary DESC) as
numeric (4, 2)) AS percent
FROM Employees WHERE State IN ('CA') AND sex ='F' ) AS
DT where percent > 0.5
ORDER BY Salary DESC;
```

以上查询的结果：

```
Surname          salary    sex     percent
---------      ----------    ---   ---------
Clark                          45000.000     F      1.00
```

排名示例 5 - 此示例使用 **ROW_NUMBER** 函数为所有窗口分区中的每一行返回行号。查询按照部门 ID 对 Employees 表进行分区，并按照开始日期对每个分区中的行进行排序。

```
SELECT DepartmentID dID, StartDate, Salary ,
ROW_NUMBER()OVER(PARTITION BY dID ORDER BY StartDate)
FROM  Employees ORDER BY 1,2;
```

以上查询的结果为：

```
dID         StartDate     Salary       Row_number()
========   ===========   ==========  =============
  100       1984-08-28    47500.000        1
  100       1985-01-01    62000.500        2
  100       1985-06-17    57490.000        3
  100       1986-06-07    72995.000        4
  100       1986-07-01    48023.690        5
  ...       ...           ...              ...
  200       1985-02-03    38500.000        1
  200       1985-12-06    54800.000        2
  200       1987-02-19    39300.000        3
  200       1987-07-10    49500.000        4
  ...       ...           ...              ...
  500       1994-02-27     24903.000       9
```

窗口化集合函数

使用窗口化集合函数，可以在同一查询中处理多个集合级别。

例如，可以列出花销小于平均值的所有季度。您可以使用集合函数（包括简单集合函数 **AVG**、**COUNT**、**MAX**、**MIN** 和 **SUM**）将可能在语句中的不同级别计算出的结果放置在同一行。通过这样放置，可以将集合值与组中的明细行进行比较，且不需要连接或相关子查询。

使用这些函数，还可以将非集合值与集合值进行比较。例如，有些客户所订购产品的数量超过了某产品在指定年份内的平均数量，销售人员可能需要编辑所有这些客户的列表，或者，经理可能需要将雇员的薪水与部门的平均薪水进行比较。

如果查询在 **SELECT** 语句中指定 **DISTINCT**，则在窗口运算符之后应用 **DISTINCT** 运算。窗口运算符的计算是在处理 **GROUP BY** 子句之后、计算 **SELECT** 列表项和查询的 **ORDER BY** 子句之前进行的。

窗口化集合示例 1 - 此查询返回一个结果集（按年分区），该结果集显示了销售量高于平均销售量的产品的列表。

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID AS
   Dept, CAST(Salary AS numeric(10,2) ) AS Sal,
   CAST(AVG(Sal) OVER(PARTITION BY DepartmentID) AS
   numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
   OVER(PARTITION BY DepartmentID) AS numeric(10,2)) AS
   STD_DEV
FROM Employees
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
   Sal> (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

以上查询的结果：

```
E_name    Dept  Sal       Average   STD_DEV
--------  ----  --------  -------   --------
Lull      100         87900.00        58736.28        16829.59
Sheffield 100         87900.00        58736.28        16829.59
Scott             100         96300.00        58736.28        16829.59
Sterling     200         64900.00        48390.94        13869.59
Savarino     200         72300.00        48390.94     13869.59
Kelly              200         87500.00        48390.94     13869.59
Shea              300        138948.00        59500.00        30752.39
Blaikie          400         54900.00        43640.67     11194.02
Morris             400         61300.00        43640.67     11194.02
Evans              400         68940.00        43640.67     11194.02
Martinez     500         55500.80        33752.20         9084.49
```

对于 2000 年，平均订单数是 1,787。四种产品（700、601、600 和 400）的销售数量高于该数量。在 2001 年，平均订单数是 1,048，有三种产品超过该数量。

窗口化集合示例 2 - 此查询返回一个结果集，该结果集显示了其薪水比其部门的平均薪水高一个标准偏差的雇员。标准偏差是数据与平均值的差异的测量单位。

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID AS
  Dept, CAST(Salary AS numeric(10,2) ) AS Sal,
  CAST(AVG(Sal) OVER(PARTITION BY dept) AS
  numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
  OVER(PARTITION BY dept) AS numeric(10,2)) AS
  STD_DEV
FROM Employees
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
  Sal> (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

每个部门至少有一名雇员的薪水远远超过平均值，如下列结果所示：

```
E_name        Dept    Sal         Average       STD_DEV
--------      ----    --------    --------      --------
Lull           100    87900.00    58736.28      16829.59
Sheffield      100    87900.00    58736.28      16829.59
Scott          100    96300.00    58736.28      16829.59
Sterling       200    64900.00    48390.94      13869.59
Savarino       200    72300.00    48390.94      13869.59
Kelly          200    87500.00    48390.94      13869.59
Shea           300   138948.00    59500.00      30752.39
Blaikie        400    54900.00    43640.67      11194.02
Morris         400    61300.00    43640.67      11194.02
Evans          400    68940.00    43640.67      11194.02
Martinez       500    55500.80    33752.20       9084.49
```

雇员 Scott 的薪水为 $96,300.00，而部门 100 的平均薪水为 $58,736.28。该部门的标准偏差是 16,829.00，这表示低于 $75,565.88 (58736.28 + 16829.60 = 75565.88) 的薪水会在平均值的一个标准偏差范围内。

统计集合函数

ANSI SQL/OLAP 扩展提供了许多其它集合函数，这些函数允许对数值数据进行统计分析。此支持包括用于计算方差、标准偏差、相关和线性回归的函数。

标准偏差和方差

采用一个参数的 SQL/OLAP 常规集函数包括以下语法语句中以粗体显示的那些函数：

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
  <BASIC AGGREGATE FUNCTION TYPE>
  | STDDEV | STDDEV_POP | STDDEV_SAMP
  | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

- **STDDEV_POP** - 计算所提供的值表达式（为组或分区的每个行计算，如果指定了 **DISTINCT**，则为删除重复值后仍旧保留的每个行计算）的总体标准偏差，总体标准偏差是指总体方差的平方根。
- **STDDEV_SAMP** - 计算所提供的值表达式（为组或分区的每个行计算，如果指定了 **DISTINCT**，则为删除重复值后仍旧保留的每个行计算）的总体标准偏差，总体标准偏差是指样本方差的平方根。
- **VAR_POP** - 计算值表达式（为组或分区的每个行计算，如果指定了 **DISTINCT**，则为删除重复值后仍旧保留的每个行计算）的总体方差，总体方差是指值表达式与值表达式平均值的差值的平方和除以组或分区中的保留行数。
- **VAR_SAMP** - 计算值表达式（为组或分区的每个行计算，如果指定了 **DISTINCT**，则为删除重复值后仍旧保留的每个行计算）的样本方差，样本方差是指值表达式与值表达式平均值的差值的平方和除以组或分区中的保留行数减一。

这些函数（包括 **STDDEV** 和 **VARIANCE**）是真正的集合函数，原因在于，它们可以计算由查询的 **ORDER BY** 子句确定的行分区的值。就像使用其它基本集合函数（如 **MAX** 或 **MIN**）一样，它们的计算忽略输入中的 NULL。此外，无论所分析的表达式的域是什么，所有方差和标准偏差计算都使用 IEEE 双精度浮点值。如果对任何方差或标准偏

差函数的输入为空集，则每个函数都将返回 NULL 作为其结果。如果为单个行计算 **VAR_SAMP**，则它返回 NULL，而 **VAR_POP** 返回值 0。

相关
用于计算相关系数的 SQL/OLAP 函数是：

- **CORR** - 返回一组数字对的相关系数。

您可以使用 CORR 函数作为窗口化集合函数（在该函数中指定窗口名称或规范使用的窗口函数类型）或作为不带 OVER 子句的简单集合函数。

协方差
用于计算协方差的 SQL/OLAP 函数包括：

- **COVAR_POP** - 返回一组数字对的总体协方差。
- **COVAR_SAMP** - 返回一组数字对的样本协方差。

协方差函数删除了 expression1 或 expression2 的值为 NULL 的所有对。

您可以使用协方差函数作为窗口化集合函数（在该函数中指定窗口名称或规范使用的窗口函数类型）或作为不带 OVER 子句的简单集合函数。

累积分布
用于计算一个值在行组中的相对位置的 SQL/OLAP 函数是 **CUME_DIST**。

窗口规范必须包含 ORDER_BY 子句。

不得在 CUME_DIST 函数中使用组合排序键。

回归分析
回归分析函数使用线性回归公式计算独立变量和相关变量之间的关系。SQL/OLAP 线性回归函数包括：

- **REGR_AVGX** - 计算回归线的独立变量的平均值。
- **REGR_AVGY** - 计算回归线的相关变量的平均值。
- **REGR_COUNT** - 返回一个整数，该整数表示用于拟合回归线的非空数字对的数量。
- **REGR_INTERCEPT** - 计算可以最好地拟合相关和独立变量的回归线的 y 截距。
- **REGR_R2** - 计算回归线的决定系数（拟合优度统计）。
- **REGR_SLOPE** - 计算与非空对拟合的线性回归线的斜率。
- **REGR_SXX** - 返回线性回归模型中使用的独立表达式的平方和。使用此函数可以计算回归模型的统计有效性。
- **REGR_SXY** - 返回相关和独立变量的乘积之和。使用此函数可以计算回归模型的统计有效性。
- **REGR_SYY** - 返回可以计算回归模型的统计有效性的值。

您可以使用回归分析函数作为窗口化集合函数（在该函数中指定窗口名称或规范使用的窗口函数类型）或作为不带 OVER 子句的简单集合函数。

加权 OLAP 集合
加权 OLAP 集合函数计算加权移动平均值：

- **EXP_WEIGHTED_AVG** - 计算指数加权移动平均值。加权确定构成平均值的每个数量的相对重要性。**EXP_WEIGHTED_AVG** 中的权重呈指数级减小。指数加权会为最新的值应用较多权重，而减小较旧值的权重，同时仍旧为较旧值应用一些权重。
- **WEIGHTED_AVG** - 计算线性加权移动平均值，其中权重随着时间的推移按算术级数减小。对于最新的数据点，权重从最高值减小，而对于最旧的数据点，权重减小为零。

窗口规范必须包含 **ORDER_BY** 子句。

非标准数据库行业扩展
在数据库行业使用的非 ANSI SQL/OLAP 集合函数扩展包括 **FIRST_VALUE**、**MEDIAN** 和 **LAST_VALUE**。

- **FIRST_VALUE** - 返回一组值中的第一个值。
- **MEDIAN** - 返回表达式中的中位数。
- **LAST_VALUE** - 返回一组值中的最后一个值。

FIRST_VALUE 和 **LAST_VALUE** 函数需要窗口规范。您可以使用 **MEDIAN** 函数作为窗口化集合函数（在该函数中指定窗口名称或规范使用的窗口函数类型）或作为不带 OVER 子句的简单集合函数。

行间函数

通过行间函数 **LAG** 和 **LEAD** 可以访问数据系列中的先前值或后续值，或表中的多个行。

行间函数同时还会进行分区，且无需自连接。通过 LAG 可以对位于表或分区中 **CURRENT ROW** 前面且与之相距给定物理偏移量的行进行访问。通过 LEAD 可以对位于表或分区中 **CURRENT ROW** 后面且与之相距给定物理偏移量的行进行访问。

LAG 和 **LEAD** 使用的语法相同。这两个函数都需要 **OVER** (**ORDER_BY**) 窗口规范。例如：

```
LAG (value_expr) [, offset [, default]]) OVER ([PARTITION BY
  window partition] ORDER BY
  window ordering)
```

和：

```
LEAD (value_expr) [, offset [, default]]) OVER ([PARTITION
BY
  window partition] ORDER BY
  window ordering)
```

OVER (**ORDER_BY**) 子句中的 **PARTITION BY** 子句是可选的。**OVER** (**ORDER_BY**) 子句不能包含窗口构架 **ROWS**/**RANGE** 规范。

value_expr 是定义要从表返回的偏移数据的表列或表达式。可以在 *value_expr* 中定义其它函数（分析函数除外）。

对于这两个函数，可输入物理偏移量来指定目标行。*offset* 值是指当前行上面或下面的行数。请输入非负数值数据类型（输入负值会生成错误）。如果输入 0，SAP Sybase IQ 会返回当前行。

可选的 *default* 值定义 *offset* 值超出表范围时要返回的值。*default* 的缺省值为 **NULL**。*default* 的数据类型必须可以隐式转换为 *value_expr* 值的数据类型，否则 SAP Sybase IQ 将生成转换错误。

LAG 示例 1 - 行间函数适用于对数据流进行计算的金融服务应用程序（如股票交易）。此示例使用 **LAG** 函数来计算特定股票的交易价格的百分比变化。假设有以下来自 `stock_trades` 虚构表的贸易数据：

```
traded at             symbol   price
-------------------   ------   ------
2009-07-13 06:07:12   SQL      15.84
2009-07-13 06:07:13   TST       5.75
2009-07-13 06:07:14   TST       5.80
2009-07-13 06:07:15   SQL      15.86
2009-07-13 06:07:16   TST       5.90
2009-07-13 06:07:17   SQL      15.86
```

注意： 虚构表 `stock_trades` 在 `iqdemo` 数据库中不可用。

查询按照股票符号对交易进行分区，按照交易时间对它们进行排序，并使用 **LAG** 函数来计算当前交易和以前交易之间的交易价格增加或降低百分比：

```
select  stock_symbol as 'Stock',
    traded_at    as 'Date/Time of Trade',
    trade_price  as 'Price/Share',
    cast ( ( ( (trade_price
        - (lag(trade_price, 1)
        over (partition by stock_symbol
            order by traded_at)))
        / trade_price)
    * 100.0) as numeric(5, 2) )
        as '% Price Change vs Previous Price'
from stock_trades
order by 1, 2
```

查询返回以下结果：

```
Stock   Date/Time of Trade   Price/  % Price Change_vs
symbol                       Share   Previous Price
------  -------------------  -----   -----------------
SQL     2009-07-13 06:07:12  15.84   NULL
SQL     2009-07-13 06:07:15  15.86   0.13
SQL     2009-07-13 06:07:17  15.86   0.00
TST     2009-07-13 06:07:13   5.75   NULL
```

```
TST     2009-07-13 06:07:14   5.80   0.87
TST     2009-07-13 06:07:16   5.90   1.72
```

第一个和第四个输出行中的 NULL 结果表明 **LAG** 函数同时超出了两个分区中第一行的范围。由于没有先前行可供比较，SAP Sybase IQ 返回 *default* 变量指定的 NULL。

分布函数

SQL/OLAP 定义了多个用于处理有序集合的函数。

这两个逆分布函数是 **PERCENTILE_CONT** 和 **PERCENTILE_DISC**。这些分析函数将一个百分点值作为函数参数使用，并对 **WITHIN GROUP** 子句中指定的一组数据执行操作或对整个数据集执行操作。

这些函数为每个组返回一个值。对于 **PERCENTILE_DISC**（离散），结果的数据类型与在 **WITHIN GROUP** 子句中指定的它的 **ORDER BY** 项的数据类型相同。对于 **PERCENTILE_CONT**（连续），结果的数据类型是数字，但前提是 **WITHIN GROUP** 子句中的 **ORDER BY** 项是数字或双精度型，或者 **ORDER BY** 项是整数或浮点型。

逆分布分析函数需要 **WITHIN GROUP (ORDER BY)** 子句。例如：

```
PERCENTILE_CONT ( expression1 )
WITHIN GROUP ( ORDER BY expression2 [ ASC | DESC ] )
```

expression1 的值必须是数值数据类型的常量，范围从 0 到 1（包含这两个数）。如果参数为 NULL，将返回“wrong argument for percentile”（百分点的参数错误）错误。如果参数值小于 0 或大于 1，将返回“data value out of range”（数据值超出范围）错误。

ORDER BY 子句（必须存在）指定要执行百分点函数的表达式以及每个组中行的排列顺序。此 **ORDER BY** 子句只在 **WITHIN GROUP** 子句中使用，而不是 **SELECT** 语句的 **ORDER BY**。

WITHIN GROUP 子句将查询结果分布到排序数据集中，函数通过此数据集计算结果。

值 *expression2* 是一种排序规范，必须是涉及列引用的单个表达式。不允许多个表达式，并且在此排序表达式中不允许使用 rank 分析函数、集合函数或子查询。

ASC 或 DESC 参数指定排序顺序，如升序或降序。升序是缺省值。

子查询、**HAVING** 子句、视图或联合中允许使用逆分布分析函数。可以在使用简单非分析集合函数的任意位置使用逆分布函数。逆分布函数忽略数据集中的 NULL。

PERCENTILE_CONT 示例 - 此示例根据以下数据集使用 **PERCENTILE_CONT** 函数确定某个区域中要进入前 10 个百分点而应该达到的汽车销售量：

```
sales       region        dealer_name
-----       ---------     -----------
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
```

```
500          Northeast     New Haven
450          Northeast     Hartford
800          Northwest     SF
600          Northwest     Seattle
500          Northwest     Portland
400          Northwest     Dublin
500          South         Houston
400          South         Austin
300          South         Dallas
200          South         Dover
```

在以下示例查询中，**SELECT** 语句包含 **PERCENTILE_CONT** 函数：

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY ProductID DESC )
FROM ViewSalesOrdersSales GROUP BY region;
```

SELECT 语句的结果列出了某区域中要进入前 10 个百分点应该达到的汽车销售量：

```
region           percentile_cont
---------        ---------------
Canada           601.0
Central          700.0
Eastern          700.0

South            700.0
Western             700.0
```

PERCENTILE_DISC 示例 - 此示例根据以下数据集使用 **PERCENTILE_DISC** 函数确定某个区域中要进入前 10 个百分点而应该达到的汽车销售量：

```
sales       region        dealer_name
-----       ---------     -----------
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
500         Northeast     New Haven
450         Northeast     Hartford
800         Northwest     SF
600         Northwest     Seattle
500         Northwest     Portland
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

在以下查询中，**SELECT** 语句包含 **PERCENTILE_DISC** 函数：

```
SELECT region, PERCENTILE_DISC(0.1) WITHIN GROUP    (ORDER BY sales
DESC ) FROM carSales GROUP BY region;
```

SELECT 语句的结果列出了每一个区域中要进入前 10 个百分点而应该达到的汽车销售量：

```
region          percentile_cont ---------        ---------------
Northeast       900 Northwest        800 South            500
```

PERCENTILE_CONT 函数 [分析]
给出一个百分点，返回与该百分点对应的值。假定是连续分布数据模型。

注意： 如果仅需要计算百分点，请改用 **NTILE** 函数，将值设为 100。

语法

```
PERCENTILE_CONT ( expression1 )
WITHIN GROUP ( ORDER BY expression2 [ ASC | DESC ] )
```

参数

参数	描述
expression1	一个 numeric 数据类型的常量，范围从 0 到 1（包含这两个数）。如果参数为 NULL，则返回“wrong argument for percentile”（百分点的参数错误）错误。如果参数值小于 0 或大于 1，则返回“data value out of range”（数据值超出范围）错误。
expression2	排序规范，必须为涉及列引用的单个表达式。不允许多个表达式，并且在此排序表达式中不允许使用 rank 分析函数、集合函数或子查询。

注释
逆分布分析函数返回第 k 个百分点值，该值可用于帮助为一组数据建立可接受阈值。函数 **PERCENTILE_CONT** 采用一个百分点值作为函数参数，并对在 **WITHIN GROUP** 子句中指定的一组数据或整个数据集执行操作。该函数为每组返回一个值。如果查询中的 **GROUP BY** 列不存在，则结果为单个行。结果的数据类型与在 **WITHIN GROUP** 子句中指定的它的 **ORDER BY** 项的数据类型相同。**PERCENTILE_CONT** 的 **ORDER BY** 表达式的数据类型必须是数值。

PERCENTILE_CONT 需要使用 **WITHIN GROUP (ORDER BY)** 子句。

ORDER BY 子句（必须存在）指定要执行百分点函数的表达式以及每个组中行的排列顺序。对于 **PERCENTILE_CONT** 函数，此表达式的数据类型必须是数值。此 **ORDER BY** 子句只在 **WITHIN GROUP** 子句中使用，而不是 **SELECT** 的 **ORDER BY**。

WITHIN GROUP 子句将查询结果分布到排序数据集中，函数通过此数据集计算结果。**WITHIN GROUP** 子句必须包含单个排序项。如果 **WITHIN GROUP** 子句包含多个或者 0 个排序项，系统将报告错误。

ASC 或 DESC 参数用于指定升序或降序排序序列。升序是缺省值。

子查询、**HAVING** 子句、视图或联合中允许使用 **PERCENTILE_CONT** 函数。
PERCENTILE_CONT 可在使用简单非分析集合函数的任意位置使用。
PERCENTILE_CONT 函数忽略数据集中的空值。

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 不受 Adaptive Server 或 SQL Anywhere 支持。

示例

以下示例使用 **PERCENTILE_CONT** 函数来确定某区域前 10 个百分点值所对应的汽车销售量。

示例中使用了以下数据集：

```
sales       region        dealer_name
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
500         Northeast     New Haven
450         Northeast     Hartford
800         Northwest     SF
600         Northwest     Seattle
500         Northwest     Portland
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

以下 **SELECT** 语句包含 **PERCENTILE_CONT** 函数：

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

SELECT 语句的结果列出了某区域前 10 个百分点值所对应的汽车销售量：

```
region           percentile_cont
Northeast        840
Northwest        740
South            470
```

PERCENTILE_DISC 函数 [分析]

给出一个百分点，返回与该百分点对应的值。假定是离散分布数据模型。

注意： 如果仅需要计算百分点，请改用 **NTILE** 函数，将值设为 100。

语法

```
PERCENTILE_DISC ( expression1 )
WITHIN GROUP ( ORDER BY expression2 [ ASC | DESC ] )
```

参数

参数	描述
expression1	一个 numeric 数据类型的常量，范围从 0 到 1（包含这两个数）。如果参数为 NULL，将返回“wrong argument for percentile”（百分点的参数错误）错误。如果参数值小于 0 或大于 1，将返回“data value out of range”（数据值超出范围）错误。
expression2	排序规范，必须为涉及列引用的单个表达式。不允许多个表达式，并且在此排序表达式中不允许使用 rank 分析函数、集合函数或子查询。

注释

逆分布分析函数返回第 k 个百分点值，该值可用于帮助为一组数据建立可接受阈值。函数 **PERCENTILE_DISC** 采用一个百分点值作为函数参数，并对在 **WITHIN GROUP** 子句中指定的一组数据或整个数据集执行运算。该函数为每个组返回一个值。如果查询中的 **GROUP BY** 列不存在，则结果为单个行。结果的数据类型与在 **WITHIN GROUP** 子句中指定的 **ORDER BY** 项的数据类型相同。**PERCENTILE_DISC** 支持可在 SAP Sybase IQ 中排序的所有数据类型。

PERCENTILE_DISC 需要使用 **WITHIN GROUP (ORDER BY)** 子句。

ORDER BY 子句（必须存在）指定要执行百分点函数的表达式以及每个组中行的排列顺序。此 **ORDER BY** 子句只在 **WITHIN GROUP** 子句中使用，而不是 **SELECT** 的 **ORDER BY**。

WITHIN GROUP 子句将查询结果分布到排序数据集中，函数通过此数据集计算结果。**WITHIN GROUP** 子句必须包含单个排序项。如果 **WITHIN GROUP** 子句包含多个或者 0 个排序项，系统将报告错误。

ASC 或 DESC 参数用于指定升序或降序排序序列。升序是缺省值。

子查询、**HAVING** 子句、视图或联合中允许使用 **PERCENTILE_DISC** 函数。**PERCENTILE_DISC** 可在使用简单非分析集合函数的任意位置使用。**PERCENTILE_DISC** 函数忽略数据集中的空值。

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 不受 Adaptive Server 或 SQL Anywhere 支持。

示例

以下示例使用 **PERCENTILE_DISC** 函数来确定某区域前 10 个百分点值所对应的汽车销售量。

示例中使用了以下数据集：

```
sales       region        dealer_name
900         Northeast     Boston
```

```
800          Northeast      Worcester
800          Northeast      Providence
700          Northeast      Lowell
540          Northeast      Natick
500          Northeast      New Haven
450          Northeast      Hartford
800          Northwest      SF
600          Northwest      Seattle
500          Northwest      Portland
400          Northwest      Dublin
500          South          Houston
400          South          Austin
300          South          Dallas
200          South          Dover
```

以下 **SELECT** 语句包含 **PERCENTILE_DISC** 函数：

```
SELECT region, PERCENTILE_DISC(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

SELECT 语句的结果列出了某区域前 10 个百分点值所对应的汽车销售量：

```
region           percentile_cont
Northeast        900
Northwest        800
South            500
```

数值函数

SAP Sybase IQ 支持的 OLAP 数字函数包括 **CEILING**（**CEIL** 为别名）、**EXP**（**EXPONENTIAL** 为别名）、**FLOOR**、**LN**（**LOG** 为别名）、**SQRT** 和 **WIDTH_BUCKET**。

```
<numeric value function> :: =
 <natural logarithm>
| <exponential function>
| <power function>
| <square root>
| <floor function>
| <ceiling function>
| <width bucket function>
```

表 4. 数值函数和语法

数值函数	语法
自然对数	**LN** (*numeric-expression*)
指数函数	**EXP** (*numeric-expression*)
Power 函数	**POWER** (*numeric-expression1*, *numeric-expression2*)
平方根	**SQRT** (*numeric-expression*)
Floor 函数	**FLOOR** (*numeric-expression*)
Ceiling 函数	**CEILING** (*numeric-expression*)

数值函数	语法
Width bucket 函数	**WIDTH_BUCKET** (*expression*, *min_value*, *max_value*, *num_buckets*)

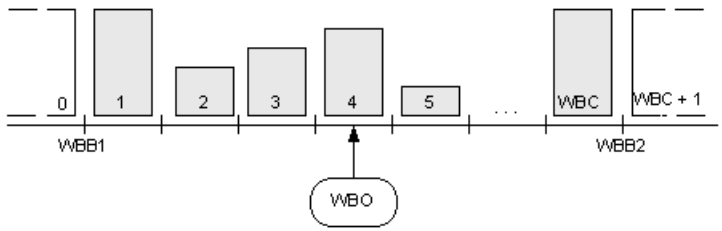
数值函数的语义是：

- **LN** - 返回参数值的自然对数。如果参数值为零或为负数，则会引发错误。LN 是 **LOG** 的同义词。
- **EXP** - 返回通过将 *e*（自然对数的底数）的值增加至参数值指定的幂而计算出的值。
- **POWER** - 返回通过将第一个参数的值增加至第二个参数的值指定的幂而计算出的值。如果第一个参数为 0，第二个参数也为 0，则返回 1。如果第一个参数为 0，而第二个参数为正数，则返回 0。如果第一个参数为 0，而第二个参数为负数，则引发异常。如果第一个参数为负数，第二个参数不是整数，则引发异常。
- **SQRT** - 返回参数值的平方根，该平方根由“**POWER** (*expression*, *0.5*)”的语法转换定义。
- **FLOOR** - 返回最接近于正无限大且不大于参数值的整数值。
- **CEILING** - 返回最接近于负无限小且不小于参数值的整数值。CEIL 是 CEILING 的同义词。

WIDTH_BUCKET 函数

WIDTH_BUCKET 函数比其它数值函数略微复杂一些。它接受四个参数：“实时值”、两个范围边界以及大小相同（或尽可能接近）的分区（边界指示的范围要拆分为这些分区）的数目。WIDTH_BUCKET 返回一个数字，该数字指示应将实时值放置在的分区（基于它的值，它的值是范围上边界和下边界之间差值的百分比）。第一个分区的编号为 1。

为了避免在实时值超出边界范围时出错，小于范围下边界的值应该放置在第一个附加桶（即桶 0）中，大于范围上边界的值应该放在最后一个附加桶（即桶 N+1）中。



例如，**WIDTH_BUCKET** (*14, 5, 30, 5*) 返回 2，原因是：

- (30-5)/5 = 5，因此范围拆分为 5 个分区，每个分区的宽度为 5 个单位。
- 第一个桶表示从 0.00% 到 19.999 …% 的值，第二个桶表示从 20.00% 到 39.999 …% 的值，第五个桶表示从 80.00% 到 100.00% 的值。

- 所选桶是通过计算 (5*(14-5)/(30-5)) + 1（桶数乘以指定值距最小值的偏移与可能值范围的比率，再加上一，即 (5*9/25) + 1，结果是 2.8）来确定的。此值是编号为 2 的桶的值范围（2.0 到 2.999……），因此选择编号为 2 的桶。

WIDTH_BUCKET 示例

以下示例基于 `credit_limit` 列为示例表中马萨诸塞州的客户创建十桶直方图，并为每名客户返回桶号 ("Credit Group")。信用额度超过最大值的客户将被分配到溢出桶 11：

注意： 此示例仅用于进行说明，不会使用 iqdemo 数据库生成此示例。

```
SELECT customer_id, cust_last_name, credit_limit,
  WIDTH_BUCKET(credit_limit, 100, 5000, 10) "Credit
  Group"
  FROM customers WHERE territory = 'MA'
  ORDER BY "Credit Group";
```

```
CUSTOMER_ID  CUST_LAST_NAME  CREDIT_LIMIT  Credit Group
-----------  --------------  ------------  ------------
825          Dreyfuss                 500             1
826          Barkin                   500             1
853          Palin                    400             1
827          Siegel                   500             1
843          Oates                    700             2
844          Julius                   700             2
835          Eastwood                1200             3
840          Elliott                 1400             3
842          Stern                   1400             3
841          Boyer                   1400             3
837          Stanton                 1200             3
836          Berenger                1200             3
848          Olmos                   1800             4
847           Streep                     5000                  11
```

如果将这些限定反转过来，桶将成为半开半闭区间。例如：**WIDTH_BUCKET** (*credit_limit*, *5000*, *0*, *5*)。在此示例中，桶号 1 的上下限为 (4000, 5000]，桶号 2 的上下限为 (3000, 4000]，桶号 5 的上下限为 (0, 1000]。上溢桶的编号为 0 (5000, +infinity)，下溢桶的编号为 6 (-infinity, 0]。

BIT_LENGTH 函数 [字符串]

返回不带符号的 64 位值，该值包含列参数的位长度。

语法

```
BIT_LENGTH( column-name )
```

参数

参数	描述
column-name	列的名称

返回
INT

注释
空值参数的返回值为空。

BIT_LENGTH 函数支持所有 SAP Sybase IQ 数据类型。

如果您有权使用非结构化数据分析功能，则可以将此函数与大对象数据一起使用。

请参见《非结构化数据分析》中的函数支持。

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - SQL Anywhere 或 Adaptive Server 不支持。

CEIL 函数 [数值]

返回大于或等于指定表达式的最小整数。

CEIL 是 **CEILING** 的同义词。

语法

```
CEIL ( numeric-expression )
```

参数

参数	描述
表达式	数据类型为精确数值、近似数值、货币或任何可隐式转换为这些类型之一的类型的列、变量或表达式。对于其它数据类型，**CEIL** 会产生错误。返回值与所提供的值属于相同的数据类型。

注释
对于给定表达式，**CEIL** 函数将采用一个参数。例如，**CEIL (-123.45)** 返回 -123。**CEIL (123.45)** 返回 124。

标准和兼容性

- SQL - 符合 ISO/ANSI SQL 标准。
- Sybase - 与 Adaptive Server Enterprise 兼容。

CEILING 函数 [数值]

返回一个数字的上限（不小于的最小整数）。

CEIL 是 **CEILING** 的同义词。

语法

```
CEILING ( numeric-expression )
```

参数

参数	描述
numeric-expression	要计算其上限的数字。

返回
DOUBLE

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 与 Adaptive Server 兼容。

示例
以下语句返回值 60.00000：

```
SELECT CEILING( 59.84567 ) FROM iq_dummy
```

以下语句返回值 123：

```
SELECT CEILING( 123 ) FROM iq_dummy
```

以下语句返回值 124.00：

```
SELECT CEILING( 123.45 ) FROM iq_dummy
```

以下语句返回值 -123.00：

```
SELECT CEILING( -123.45 ) FROM iq_dummy
```

EXP 函数 [数值]

返回指数函数，即 e 的指定数字次乘方。

语法

```
EXP ( numeric-expression )
```

参数

表 5. 参数

参数	描述
numeric-expression	指数。

返回
DOUBLE

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 与 Adaptive Server Enterprise 兼容。

示例
以下语句返回值 3269017.3724721107：

```
SELECT EXP( 15 ) FROM iq_dummy
```

FLOOR 函数 [数值]
返回一个数字的下限（不大于的最大整数）。

语法

```
FLOOR ( numeric-expression )
```

参数

表 6. 参数

参数	描述
numeric-expression	数字，通常是浮点数。

返回
DOUBLE

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 与 Adaptive Server Enterprise 兼容。

示例
以下语句返回值 123.00：

```
SELECT FLOOR ( 123 ) FROM iq_dummy
```

以下语句返回值 123：

```
SELECT FLOOR ( 123.45 ) FROM iq_dummy
```

以下语句返回值 -124.00。

```
SELECT FLOOR ( -123.45 ) FROM iq_dummy
```

LN 函数 [数值]

返回指定表达式的自然对数。

语法

```
LN ( numeric-expression )
```

参数

参数	描述
numeric-expression	数据类型为精确数值、近似数值、货币或可隐式转换为这些类型的任何类型的列、变量或表达式。对于其它数据类型，**LN** 函数会产生错误。返回值的数据类型为 DOUBLE。

注释

LN 采用一个参数。例如，**LN** (*20*) 返回 2.995732。

LN 函数是 **LOG** 函数的别名。

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 不受 Adaptive Server Enterprise 支持。请改用 LOG 函数。

POWER 函数 [数值]

以一个数字为底数另一个数字为指数计算乘方值。

语法

```
POWER ( numeric-expression1, numeric-expression2 )
```

参数

参数	描述
numeric-expression1	底数。
numeric-expression2	指数。

返回

DOUBLE

注释

将 *numeric-expression1* 增加至乘方 *numeric-expresson2*。

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 与 Adaptive Server Enterprise 兼容。

示例
以下语句返回值 64：

```
SELECT Power( 2, 6 ) FROM iq_dummy
```

SQRT 函数 [数值]
返回一个数字的平方根。

语法

```
SQRT ( numeric-expression )
```

参数

参数	描述
numeric-expression	要计算其平方根的数字。

返回
DOUBLE

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 与 Adaptive Server Enterprise 兼容。

示例
以下语句返回值 3：

```
SELECT SQRT( 9 ) FROM iq_dummy
```

WIDTH_BUCKET 函数 [数值]
对于给定表达式，**WIDTH_BUCKET** 函数返回表元号，即在计算此表达式后将为其结果分配的表元号。

语法

```
WIDTH_BUCKET ( expression, min_value, max_value, num_buckets )
```

参数

参数	描述
expression	为其创建直方图的表达式。此表达式的计算结果必须是数值或日期时间值，或者是可以隐式转换为数值或日期时间值的值。如果 *expr* 的计算结果为空值，则表达式返回空值。
min_value	解析为 *expr* 的可接受范围的端点的表达式。还必须计算为数值或日期时间值，不能计算为空。
max_value	解析为 *expr* 的可接受范围的端点的表达式。还必须计算为数值或日期时间值，不能计算为空。
num_buckets	是解析为表示表元数的常量的表达式。此表达式的计算结果必须是正整数。

注释

您可以使用 **WIDTH_BUCKET** 函数生成等宽直方图。等宽直方图将数据集分为区间大小（最大值到最小值）相等的表元。每个表元含有的行数将会不同。相关函数 **NTILE** 创建等高表元。

仅能为数字、日期或日期时间数据类型生成等宽直方图；因此，前三个参数应该全部是数值表达式或者全部是日期表达式。不允许其它类型的表达式。如果第一个参数为 NULL，则结果为 NULL。如果第二个或第三个参数为 NULL，则返回错误消息，原因是：空值无法以日期或数值形式表示某个范围的终点（或任意点）。最后一个参数 (表元数) 应该是计算结果为正整数值的数值表达式；0、NULL 或负值将导致出现错误。

表元的编号为 0 到 (n+1)。表元 0 含有的值数小于最小数量。表元 (n+1) 含有的值数大于或等于指定的最大值。

标准和兼容性

- SQL - ISO/ANSI SQL 语法的供应商扩展。
- Sybase - 不受 Adaptive Server Enterprise 支持。

示例

以下示例基于 `credit_limit` 列为示例表中马萨诸塞州的客户创建十域桶直方图，并为每名客户返回域桶号 ("Credit Group")。信用额度超过最大值的客户将被分配到溢出域桶 11：

```
select EmployeeID, Surname, Salary, WIDTH_BUCKET(Salary, 29000,
60000, 4) "Wages" from Employees where State = 'FL' order by "Wages"
```

```
EMPLOYEEID   SURNAME              SALARY         Wages
----------   -------              ------         -----
888          Charlton          28300.000             0
1390         Litton            58930.000             4
207          Francis           53870.000             4
```

```
266          Gowda              59840.000          4
445          Lull               87900.000          5
1021         Sterling           64900.000          5
902          Kelly              87500.000          5
1576         Evans              68940.000          5
```

如果将这些限定反转过来，桶将成为半开半闭区间。例如：**WIDTH_BUCKET**（*credit_limit*、*5000*、*0*、*5*）。在此示例中，桶号 1 的上下限为 (4000, 5000]，桶号 2 的上下限为 (3000, 4000]，桶号 5 的上下限为 (0, 1000]。上溢桶的编号为 0（5000，正无穷），下溢桶的编号为 6（负无穷，0]。

OLAP 规则和限制

下文概述了控制 OLAP 功能的规则和限制。

可以使用 OLAP 函数
SAP Sybase IQ 为 SQL OLAP 函数提供了一些规则、约束和限制。

- 在 **SELECT** 列表中
- 在表达式中
- 作为标量函数的参数
- 在最终 **ORDER BY** 子句中（通过在查询中的其它位置使用 OLAP 函数的别名或位置引用）

不能使用 OLAP 函数
在以下情况下，不能使用 OLAP 函数：

- 在子查询中。
- 在 **WHERE** 子句的搜索条件中。
- 作为 **SET**（集合）函数的参数。例如，以下表达式无效：
  ```
  SUM(RANK() OVER(ORDER BY dollars))
  ```
- 窗口集合不能是另一个参数的参数，但如果在视图或派生表中生成内部参数则例外。对于排名函数，同样如此。
- 窗口集合函数和 **RANK** 函数不得在 **HAVING** 子句中使用。
- 窗口集合函数不得指定 **DISTINCT**。
- 窗口函数不能嵌套在其它窗口函数内部。
- **OVER** 子句不支持逆分布函数。
- 不得在窗口定义子句中使用外部引用。
- 允许在 OLAP 函数中使用相关引用，但不允许使用相关列别名。

OLAP 函数引用的列必须是同一查询块（OLAP 函数和 **GROUP BY** 子句显示在该查询块中）中的分组列或集合函数。OLAP 处理发生在分组和集合操作之后、应用最终 **ORDER BY** 子句之前；因此，必须可以从这些中间结果派生 OLAP 表达式。如果查询块中没有 **GROUP BY** 子句，则 OLAP 函数可以引用选择列表中的其它列。

SAP Sybase IQ 限制
SAP Sybase IQ 对 SQL OLAP 函数的限制如下：

- 不支持窗口构架定义中用户定义的函数。
- 在窗口框架定义中使用的常量必须是无符号的数值，且不得超过最大值 `BIG INT` 2^{63-1}。
- 窗口集合函数和 **RANK** 函数不能在 **DELETE** 和 **UPDATE** 语句中使用。
- 窗口集合函数和 **RANK** 函数不得在子查询中使用。
- CUME_DIST 当前不受支持。
- 分组集当前不受支持。
- 相关函数和线性回归函数当前不受支持。

其它 OLAP 示例

此节提供了其它 OLAP 函数使用示例。

当处理中间结果行时，窗口的起点和终点可能会改变。例如，计算累计总和将涉及一个窗口，该窗口的起点固定在每个分区的第一行，终点沿分区行滑动以包括当前行。

像另一示例一样，窗口的起点和终点都是可变的，但它能为整个分区定义固定数量的行。使用这种构造，用户可以编写计算移动平均值的查询；例如，返回三天的股票价格的移动平均值的 SQL 查询。

示例：查询中的窗口函数

此查询列出了 2005 年 7 月和 8 月装运的所有产品以及到装运日期为止的累计装运数量：

```
SELECT p.id, p.description, s.quantity, s.shipdate,
```

```
SUM(s.quantity) OVER (PARTITION BY productid ORDER BY s.shipdate rows
between unbounded preceding and current row)FROM SalesOrderItems s
JOIN Products p on(s.ProductID =p.id) WHERE s.ShipDate BETWEEN
'2001-05-01' and '2001-08-31' AND s.quantity > 40 ORDER BY p.id;
```

```
ID   description   quantity   ship_date   sum quantity
---  -----------   --------   ---------   ------------
302  Crew Neck           60  2001-07-02             60
400  Cotton Cap          60  2001-05-26             60
400  Cotton Cap          48  2001-07-05            108
401  Wool cap            48  2001-06-02             48
401  Wool cap            60  2001-06-30            108
401  Wool cap            48  2001-07-09            156
500  Cloth Visor         48  2001-06-21             48
501  Plastic Visor       60  2001-05-03             60
501  Plastic Visor       48  2001-05-18            108
501  Plastic Visor       48  2001-05-25            156
501  Plastic Visor       60  2001-07-07            216
601  Zipped Sweatshirt   60  2001-07-19             60
```

```
700  Cotton Shorts       72  2001-05-18             72
700  Cotton Shorts       48  2001-05-31            120
```

在此示例中，要在连接两个表和应用查询的 **WHERE** 子句之后，才执行 **SUM** 窗口函数的计算。查询使用行内窗口规格，该规范指定按如下方式处理的连接的输入行：

1. 根据 prod_id 属性的值将输入行分区（分组）。
2. 在每个分区中，按 ship_date 属性对行进行排序。
3. 对于分区中的每一行，计算数量属性的 **SUM()** 函数，并使用包括每个分区的第一（排序）行的滑动窗口，直到包含当前行。

另一种构造查询的方式是指定与使用它的函数不同的窗口。当根据同一窗口指定多个窗口函数时，这非常有用。在使用窗口函数的查询中，使用窗口子句（声明由累计标识的窗口）的构造如下所示：

```
SELECT p.id, p.description, s.quantity, s.shipdate, SUM(s.quantity)
OVER(cumulative ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW )
cumulative FROM SalesOrderItems s JOIN Products p On (s.ProductID
=p.id)WHERE s.shipdate BETWEEN ‘2001-07-01’ and ‘2001-08-31’Window
cumulative as (PARTITION BY s.productid ORDER BY s.shipdate)ORDER BY
p.id;
```

窗口子句显示在查询规范中的 **ORDER BY** 子句之前。当使用窗口子句时，存在以下限制：

- 行内窗口规格不得包含 **PARTITION BY** 子句。
- 在窗口子句中指定的窗口不得包含窗口构架子句。

  ```
  <WINDOW FRAME CLAUSE> ::=
    <WINDOW FRAME UNIT>
    <WINDOW FRAME EXTENT>
  ```
- 行内窗口规格或在窗口子句中指定的窗口规范都可以（但不能同时）包含窗口排序子句。

  ```
  <WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
  ```

示例：含多个函数的窗口

此查询定义一个（命名）窗口并通过它计算多个函数结果：

```
SELECT p.ID, p.Description, s.quantity, s.ShipDate, SUM(s.Quantity)
   OVER ws1, MIN(s.quantity) OVER ws1
FROM SalesOrderItems s
JOIN Products p ON (s.ProductID =p.ID)
  WHERE s.ShipDate BETWEEN '2000-01-09' AND'2000-01-17'
  AND s.Quantity > 40 window ws1
  AS(PARTITION BY productid
  ORDER BY shipdate rows between unbounded preceding and current row)
ORDER BY p.id;
```

```
ID   Description   quantity   shipDate     SUM   MIN
---  -----------   --------   -----------  ---   ---
400  Cotton Cap          48    2000-01-09   48    48
401  Wool cap            48    2000-01-09   48    48
500  Cloth Visor         60    2000-01-14   60    60
```

```
500  Cloth Visor          60    2000-01-15  120    60
501  Plastic Visor        60    2000-01-14   60    60
```

示例：计算累计总和

以下查询按部门和 **ORDER BY**start_date 计算薪水的累计总和。

```
SELECT dept_id, start_date, name, salary,
  SUM(salary) OVER (PARTITION BY dept_id ORDER BY
  start_date ROWS BETWEEN UNBOUNDED PRECEDING AND
  CURRENT ROW)
FROM emp1
ORDER BY dept_id, start_date;
```

```
DepartmentID  start_date   name    salary   sum(salary)

--------     ----------    ----    ------   -----
100          1996-01-01    Anna       18000       18000
100                        1997-01-01    Mike    28000    46000
100                        1998-01-01    Scott   29000    75000
100                        1998-02-01    Antonia 22000    97000
100                        1998-03-12    Adam    25000   122000
100                        1998-12-01    Amy     18000   140000
200                        1998-01-01    Jeff    18000    18000
200                        1998-01-20    Tim     29000    47000
200                        1998-02-01    Jim     22000    69000
200                        1999-01-10    Tom     28000    97000
300                        1998-03-12    Sandy   55000    55000
300                        1998-12-01    Lisa    38000    93000
300                        1999-01-10    Peter   48000   141000
```

示例：计算移动平均值

以下查询生成连续三个月的销售额的移动平均值。窗口构架的大小是三行：两个前面的行加一个当前行。窗口从分区的开始滑至结尾。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
  BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

```
prod_id   month_num      sales       avg(sales)
-------   ---------      ------      ----------
10                1        100          100.00
10                2        120          110.00
10                3        100          106.66
10                4        130          116.66
10                5        120          116.66
10                6        110          120.00
20                1         20           20.00
20                2         30           25.00
20                3         25           25.00
20                4         30           28.33
20                5         31           28.66
20                6         20           27.00
```

```
30                1          10           10.00
30                2          11           10.50
30                3          12           11.00
30                4           1            8.00
```

示例： ORDER BY 结果

在此示例中，查询顶部的 **ORDER BY** 子句应用到窗口函数的最终结果。窗口子句中的 **ORDER BY** 应用到窗口函数的输入数据。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
  BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id desc, month_num;
```

```
prod_id  month_num       sales   avg(sales)
-------  ---------       -----   ----------
30               1          10       10.00
30               2          11       10.50
30               3          12       11.00
30               4           1        8.00
20               1          20       20.00
20               2          30       25.00
20               3          25       25.00
20               4          30       28.33
20               5          31       28.66
20               6          20       27.00
10               1         100      100.00
10               2         120      110.00
10               3         100      106.66
10               4         130      116.66
10               5         120      116.66
10               6         110      120.00
```

示例：查询中的多个集合函数

此示例针对查询中的不同窗口计算集合值。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (WS1 ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS
  CAvg, SUM(sales) OVER(WS1 ROWS BETWEEN UNBOUNDED
  PRECEDING AND CURRENT ROW) AS CSum
FROM sale WHERE rep_id = 1 WINDOW WS1 AS (PARTITION BY
  prod_id
ORDER BY month_num)
ORDER BY prod_id, month_num;
```

```
prod_id  month_num    sales     CAvg     CSum
-------  ---------    -----     ----     ----
10               1      100   110.00      100
10               2      120   106.66      220
10               3      100   116.66      320
10               4      130   116.66      450
10               5      120   120.00      570
10               6      110   115.00      680
```

```
20                1        20    25.00        20
20                2        30    25.00        50
20                3        25    28.33        75
20                4        30    28.66       105
20                5        31    27.00       136
20                6        20    25.50       156
30                1        10    10.50        10
30                2        11    11.00        21
30                3        12     8.00        33
30                4         1     6.50        34
```

示例：对 ROWS 和 RANGE 进行比较的窗口构架

此查询对 ROWS 和 RANGE 进行比较。数据在每个 **ORDER BY** 子句中包含重复的 ROWS。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (ws1 RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) AS
    Range_sum, SUM(sales) OVER
  (ws1 ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS
    Row_sum
FROM sale window ws1 AS (PARTITION BY prod_id ORDER BY
  month_num)
ORDER BY prod_id, month_num;
```

```
prod_id  month_num   sales   Range_sum   Row_sum
-------  ---------   -----   ----------  -------
10                1     100         250       100
10                1     150         250       250
10                2     120         370       370
10                3     100         470       370
10                4     130         350       350
10                5     120         381       350
10                5      31         381       281
10                6     110         391       261
20                1      20          20        20
20                2      30          50        50
20                3      25          75        75
20                4      30          85        85
20                5      31          86        86
20                6      20          81        81
30                1      10          10        10
30                2      11          21        21
30                3      12          33        33
30                4       1          25        24
30                4       1          25        14
```

示例：不包括当前行的窗口构架

在此示例中，您可以定义窗口构架以排除当前行。查询计算四个行（不包括当前行）的总和。

```
SELECT prod_id, month_num, sales, sum(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
  BETWEEN 6 PRECEDING AND 2 PRECEDING)
```

```
FROM sale
ORDER BY prod_id, month_num;
```

```
prod_id   month_num       sales    sum(sales)
-------   ---------       -----    ----------
10                1         100        (NULL)
10                1         150        (NULL)
10                2         120        (NULL)
10                3         100           250
10                4         130           370
10                5         120           470
10                5          31           470
10                6         110           600
20                1          20        (NULL)
20                2          30        (NULL)
20                3          25            20
20                4          30            50
20                5          31            75
20                6          20           105
30                1          10        (NULL)
30                2          11        (NULL)
30                3          12            10
30                4           1            21
30                4           1            21
```

示例：RANGE 的窗口构架

此查询说明 RANGE 窗口构架。汇总中使用的行数是可变的。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
  BETWEEN 1 FOLLOWING AND 3 FOLLOWING)
FROM sale
ORDER BY prod_id, month_num;
```

```
prod_id   month_num       sales    sum(sales)
-------   ---------       -----    ----------
10                1         100           350
10                1         150           350
10                2         120           381
10                3         100           391
10                4         130           261
10                5         120           110
10                5          31           110
10                6         110        (NULL)
20                1          20            85
20                2          30            86
20                3          25            81
20                4          30            51
20                5          31            20
20                6          20        (NULL)
30                1          10            25
30                2          11            14
30                3          12             2
30                4           1        (NULL)
30                4           1        (NULL)
```

示例：Unbounded Preceding and Unbounded Following

在此示例中，窗口构架可以包括分区中的所有行。查询计算整个分区（一个月中没有重复行）的最大销售额。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
  BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

```
prod_id   month_num       sales    SUM(sales)
-------   ---------       -----    ----------
10                1         100           680
10                2         120           680
10                3         100           680
10                4         130           680
10                5         120           680
10                6         110           680
20                1          20           156
20                2          30           156
20                3          25           156
20                4          30           156
20                5          31           156
20                6          20           156
30                1          10            34
30                2          11            34
30                3          12            34
30                4           1            34
```

本示例中的查询相当于：

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id )
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

示例：RANGE 的缺省窗口构架

此查询说明 RANGE 的缺省窗口构架：

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num)
FROM sale
ORDER BY prod_id, month_num;
```

```
prod_id   month_num       sales    SUM(sales)
-------   ---------       -----    ----------
10                1         100           250
10                1         150           250
10                2         120           370
10                3         100           470
10                4         130           600
10                5         120           751
10                5          31           751
```

```
10                  6          110            861
20                  1           20             20
20                  2           30             50
20                  3           25             75
20                  4           30            105
20                  5           31            136
20                  6           20            156
30                  1           10             10
30                  2           11             21
30                  3           12             33
30                  4            1             35
30                  4            1             35
```

本示例中的查询相当于：

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
  BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM sale
ORDER BY prod_id, month_num;
```

OLAP 函数的 BNF 语法

Backus-Naur Form 语法概述了各种 ANSI SQL 分析函数（其中有许多在 SAP Sybase IQ 中实现）的特定语法支持。

语法规则 1

```
<SELECT LIST EXPRESSION> ::=
  <EXPRESSION>
  | <GROUP BY EXPRESSION>
  | <AGGREGATE FUNCTION>
  | <GROUPING FUNCTION>
  | <TABLE COLUMN>
  | <WINDOWED TABLE FUNCTION>
```

语法规则 2

```
<QUERY SPECIFICATION> ::=
  <FROM CLAUSE>
  [ <WHERE CLAUSE> ]
  [ <GROUP BY CLAUSE> ]
  [ <HAVING CLAUSE> ]
  [ <WINDOW CLAUSE> ]
[ <ORDER BY CLAUSE> ]
```

语法规则 3

```
<ORDER BY CLAUSE> ::= <ORDER SPECIFICATION>
```

语法规则 4

```
<GROUPING FUNCTION> ::=
  GROUPING <LEFT PAREN> <GROUP BY EXPRESSION>
  <RIGHT PAREN>
```

语法规则 5

```
<WINDOWED TABLE FUNCTION> ::=
  <WINDOWED TABLE FUNCTION TYPE> OVER <WINDOW NAME OR
  SPECIFICATION>
```

语法规则 6

```
<WINDOWED TABLE FUNCTION TYPE> ::=
  <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
  | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
  | <WINDOW AGGREGATE FUNCTION>
```

语法规则 7

```
<RANK FUNCTION TYPE> ::=
  RANK | DENSE RANK | PERCENT RANK | CUME_DIST
```

语法规则 8

```
<WINDOW AGGREGATE FUNCTION> ::=
  <SIMPLE WINDOW AGGREGATE FUNCTION>
  | <STATISTICAL AGGREGATE FUNCTION>
```

语法规则 9

```
<AGGREGATE FUNCTION> ::=
  <DISTINCT AGGREGATE FUNCTION>
  | <SIMPLE AGGREGATE FUNCTION>
  | <STATISTICAL AGGREGATE FUNCTION>
```

语法规则 10

```
<DISTINCT AGGREGATE FUNCTION> ::=
  <BASIC AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <DISTINCT> <EXPRESSION> <RIGHT PAREN>
  | LIST <LEFT PAREN> DISTINCT <EXPRESSION>
  [ <COMMA> <DELIMITER> ]
  [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

语法规则 11

```
<BASIC AGGREGATE FUNCTION TYPE> ::=
  SUM | MAX | MIN | AVG | COUNT
```

语法规则 12

```
<SIMPLE AGGREGATE FUNCTION> ::=
  <SIMPLE AGGREGATE FUNCTION TYPE> <LEFT PAREN>
```

```
  <EXPRESSION> <RIGHT PAREN>
  | LIST <LEFT PAREN> <EXPRESSION> [ <COMMA>
  <DELIMITER> ]
  [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

语法规则 13

```
<SIMPLE AGGREGATE FUNCTION TYPE> ::= <SIMPLE WINDOW AGGREGATE
FUNCTION TYPE>
```

语法规则 14

```
<SIMPLE WINDOW AGGREGATE FUNCTION> ::=
  <SIMPLE WINDOW AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <EXPRESSION> <RIGHT PAREN>
| GROUPING FUNCTION
```

语法规则 15

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
  <BASIC AGGREGATE FUNCTION TYPE>
  | STDDEV | STDDEV_POP | STDDEV_SAMP
  | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

语法规则 16

```
<STATISTICAL AGGREGATE FUNCTION> ::=
  <STATISTICAL AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <DEPENDENT EXPRESSION> <COMMA> <INDEPENDENT
  EXPRESSION> <RIGHT PAREN>
```

语法规则 17

```
<STATISTICAL AGGREGATE FUNCTION TYPE> ::=
  CORR | COVAR_POP | COVAR_SAMP | REGR_R2 |
  REGR_INTERCEPT | REGR_COUNT | REGR_SLOPE |
  REGR_SXX | REGR_SXY | REGR_SYY | REGR_AVGY |
  REGR_AVGX
```

语法规则 18

```
<WINDOW NAME OR SPECIFICATION> ::=
  <WINDOW NAME> | <IN-LINE WINDOW SPECIFICATION>
```

语法规则 19

```
<WINDOW NAME> ::= <IDENTIFIER>
```

语法规则 20

```
<IN-LINE WINDOW SPECIFICATION> ::= <WINDOW SPECIFICATION>
```

语法规则 21

```
<WINDOW CLAUSE> ::= <WINDOW WINDOW DEFINITION LIST>
```

语法规则 22

```
<WINDOW DEFINITION LIST> ::=
  <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
  } . . . ]
```

语法规则 23

```
<WINDOW DEFINITION> ::=
  <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>
```

语法规则 24

```
<NEW WINDOW NAME> ::= <WINDOW NAME>
```

语法规则 25

```
<WINDOW SPECIFICATION> ::=
  <LEFT PAREN> <WINDOW SPECIFICATION> <DETAILS> <RIGHT
  PAREN>
```

语法规则 26

```
<WINDOW SPECIFICATION DETAILS> ::=
  [ <EXISTING WINDOW NAME> ]
  [ <WINDOW PARTITION CLAUSE> ]
  [ <WINDOW ORDER CLAUSE> ]
  [ <WINDOW FRAME CLAUSE> ]
```

语法规则 27

```
<EXISTING WINDOW NAME> ::= <WINDOW NAME>
```

语法规则 28

```
<WINDOW PARTITION CLAUSE> ::=
  PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

语法规则 29

```
<WINDOW PARTITION EXPRESSION LIST> ::=
  <WINDOW PARTITION EXPRESSION>
  [ { <COMMA> <WINDOW PARTITION EXPRESSION> } . . . ]
```

语法规则 30

```
<WINDOW PARTITION EXPRESSION> ::= <EXPRESSION>
```

语法规则 31

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

语法规则 32

```
<WINDOW FRAME CLAUSE> ::=
  <WINDOW FRAME UNIT>
  <WINDOW FRAME EXTENT>
```

语法规则 33

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

语法规则 34

```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW FRAME
BETWEEN>
```

语法规则 35

```
<WINDOW FRAME START> ::=
  UNBOUNDED PRECEDING
  | <WINDOW FRAME PRECEDING>
  | CURRENT ROW
```

语法规则 36

```
<WINDOW FRAME PRECEDING> ::= <UNSIGNED VALUE SPECIFICATION>
PRECEDING
```

语法规则 37

```
<WINDOW FRAME BETWEEN> ::=
  BETWEEN <WINDOW FRAME BOUND 1> AND <WINDOW FRAME
  BOUND 2>
```

语法规则 38

```
<WINDOW FRAME BOUND 1> ::= <WINDOW FRAME BOUND>
```

语法规则 39

```
<WINDOW FRAME BOUND 2> ::= <WINDOW FRAME BOUND>
```

语法规则 40

```
<WINDOW FRAME BOUND> ::=
  <WINDOW FRAME START>
  | UNBOUNDED FOLLOWING
  | <WINDOW FRAME FOLLOWING>
```

语法规则 41

```
<WINDOW FRAME FOLLOWING> ::= <UNSIGNED VALUE SPECIFICATION>
FOLLOWING
```

语法规则 42

```
<GROUP BY EXPRESSION> ::= <EXPRESSION>
```

语法规则 43

```
<SIMPLE GROUP BY TERM> ::=
  <GROUP BY EXPRESSION>
  | <LEFT PAREN> <GROUP BY EXPRESSION> <RIGHT PAREN>
  | <LEFT PAREN> <RIGHT PAREN>
```

语法规则 44

```
<SIMPLE GROUP BY TERM LIST> ::=
  <SIMPLE GROUP BY TERM> [ { <COMMA> <SIMPLE GROUP BY
  TERM> } . . . ]
```

语法规则 45

```
<COMPOSITE GROUP BY TERM> ::=
  <LEFT PAREN> <SIMPLE GROUP BY TERM>
  [ { <COMMA> <SIMPLE GROUP BY TERM> } . . . ]
  <RIGHT PAREN>
```

语法规则 46

```
<ROLLUP TERM> ::= ROLLUP <COMPOSITE GROUP BY TERM>
```

语法规则 47

```
<CUBE TERM> ::= CUBE <COMPOSITE GROUP BY TERM>
```

语法规则 48

```
<GROUP BY TERM> ::=
  <SIMPLE GROUP BY TERM>
  | <COMPOSITE GROUP BY TERM>
  | <ROLLUP TERM>
  | <CUBE TERM>
```

语法规则 49

```
<GROUP BY TERM LIST> ::=
  <GROUP BY TERM> [ { <COMMA> <GROUP BY TERM> } ⋯ ]
```

语法规则 50

```
<GROUP BY CLAUSE> ::= GROUP BY <GROUPING SPECIFICATION>
```

语法规则 51

```
<GROUPING SPECIFICATION> ::=
  <GROUP BY TERM LIST>
  | <SIMPLE GROUP BY TERM LIST> WITH ROLLUP
```

```
  | <SIMPLE GROUP BY TERM LIST> WITH CUBE
```

语法规则 52
不支持。

语法规则 53

```
<ORDER SPECIFICATION> ::= ORDER BY <SORT SPECIFICATION LIST>
  <SORT SPECIFICATION LIST> ::= <SORT SPECIFICATION>
  [ { <COMMA> <SORT SPECIFICATION> } . . . ]
  <SORT SPECIFICATION> ::= <SORT KEY>
  [ <ORDERING SPECIFICATION> ] [ <NULL ORDERING> ]
  <SORT KEY> ::= <VALUE EXPRESSION>
  <ORDERING SPECIFICATION> ::= ASC | DESC
  <NULL ORDERING> := NULLS FIRST | NULLS LAST
```

附录：访问远程数据

SAP Sybase IQ 可以访问位于独立服务器 SAP Sybase 和非 SAP Sybase 上的数据，就像数据存储在本地服务器中一样。

可以使用此功能将数据迁移至 SAP Sybase IQ 数据库或者跨数据库查询数据。

SAP Sybase IQ 和远程数据

通过 SAP Sybase IQ 远程数据访问，可以访问其它数据源中的数据。可以使用此功能将数据迁移至 SQL Anywhere 数据库或者跨数据库查询数据。

Sybase Open Client 和 jConnect 连接的特性

当 SAP Sybase IQ 通过 TDS 为应用程序提供服务时，会自动将相关数据库选项的值设置为与 Adaptive Server 的缺省行为兼容。这些选项是临时设置的，它们仅在连接期间有效。客户端应用程序可以随时替换它们。

缺省设置
在使用 TDS 的连接上设置的数据库选项包括：

选项	设置为
allow_nulls_by_default	Off
ansinull	Off
chained	Off
close_on_endtrans	Off
date_format	YYYY-MM-DD
date_order	MDY
escape_character	Off
isolation_level	1
on_tsql_error	继续
quoted_identifier	Off
time_format	HH:NN:SS.SSS
timestamp_format	YYYY-MM-DD HH:NN:SS.SSS

选项	设置为
tsql_variables	On

如何设置启动选项

TDS 连接的缺省数据库选项使用名为 sp_tsql_environment 的系统过程进行设置。此过程设置以下选项：

```
SET TEMPORARY OPTION allow_nulls_by_default='Off';
SET TEMPORARY OPTION ansinull='Off';
SET TEMPORARY OPTION chained='Off';
SET TEMPORARY OPTION close_on_endtrans='Off';
SET TEMPORARY OPTION date_format='YYYY-MM-DD';
SET TEMPORARY OPTION date_order='MDY';
SET TEMPORARY OPTION escape_character='Off';
SET TEMPORARY OPTION isolation_level='1';
SET TEMPORARY OPTION on_tsql_error='Continue';
SET TEMPORARY OPTION quoted_identifier='Off';
SET TEMPORARY OPTION time_format='HH:NN:SS.SSS';
SET TEMPORARY OPTION timestamp_format='YYYY-MM-DD HH:NN:SS.SSS';
SET TEMPORARY OPTION tsql_variables='On';
```

注意： 不要改动 sp_tsql_environment 过程。该过程仅供系统使用。

该过程仅为使用 TDS 通信协议的连接设置选项。其中包括使用 jConnect 的 Sybase Open Client 和 JDBC 连接。其它连接（ODBC 和嵌入式 SQL）使用数据库的缺省设置。

虽然 SAP Sybase IQ 允许使用更长的用户名和口令，但 TDS 客户端名称和口令不能超过 30 个字节。如果口令或用户 ID 超过 30 个字节，则通过 TDS 尝试连接（如使用 jConnect）时将返回无效用户或口令错误。

更改 TDS 连接的选项设置

当 SAP Sybase IQ 通过 TDS 为应用程序提供服务时，会自动将相关数据库选项的值设置为与 Adaptive Server 的缺省行为兼容。您随时可以更改 TDS 连接的选项。

更改 TDS 连接的选项设置

当 SAP Sybase IQ 通过 TDS 为应用程序提供服务时，会自动将相关数据库选项的值设置为与 Adaptive Server 的缺省行为兼容。您随时可以更改 TDS 连接的选项。

1. 创建一个用于设置所需数据库选项的过程。
2. 将 login_procedure 选项设置为新过程的名称。

以后的连接将使用该过程。可以将该过程配置为因用户 ID 而异。

访问远程数据的要求

访问远程数据需要具备几个基本元素。

远程表映射

SAP Sybase IQ 将表提供给客户端应用程序的方式就像表中的所有数据都存储在与该应用程序连接的数据库中一样。在内部，当执行涉及远程表的查询时，先确定其存储位置，然后访问远程位置以便能够检索数据。

要访问远程表中的数据，必须进行如下设置。

1. 必须定义远程数据所在的远程服务器。其中包括远程服务器的服务器类和位置。使用 CREATE SERVER 语句执行此操作。
2. 如果访问远程服务器上数据库所需的证书与所连接的数据库不同，则必须定义远程服务器用户登录信息。使用 CREATE EXTERNLOGIN 语句执行此操作。
3. 必须创建代理表定义。这将指定本地代理表到远程表的映射。这其中包括远程表所在的服务器、数据库名、所有者名称、表名和远程表的列名。使用 CREATE EXISTING TABLE 语句执行此操作。另外，还可使用 CREATE TABLE 语句在远程服务器上创建新表。

若要管理远程服务器定义、外部登录和代理表映射，，也可以使用 Interactive SQL 等工具执行 SQL 语句。

警告！ 小心

一些远程服务器（例如 Microsoft Access、Microsoft SQL Server 和 Sybase Adaptive Server Enterprise）不会跨越 COMMIT 和 ROLLBACK 保留游标。但是，只要关闭自动提交（这是 Interactive SQL 中的缺省行为），仍可使用 Interactive SQL 来查看和编辑这些代理表中的数据。其它 RDBMS（包括 Oracle 数据库、IBM DB2 和 SAP Sybase IQ）没有此限制。

用于远程数据访问的服务器类

您在 CREATE SERVER 语句中指定的服务器类决定远程连接的行为。这些服务器类为 SAP Sybase IQ 提供详细的服务器功能信息。SAP Sybase IQ 会将 SQL 语句的格式设置为适用于服务器的功能。

所有服务器类均基于 ODBC。每一个服务器类都具有一组独特的特性，您需要知道这些特性，才能配置用于进行远程数据访问的服务器。应参考关于服务器类类别的常规信息以及特定于各个服务器类的信息。

服务器类包括：

SAODBC
ULODBC
ADSODBC
ASEODBC
DB2ODBC
HANAODBC
IQODBC

MSACCESSODBC
MSSODBC
MYSQLODBC
ODBC
ORAODBC

注意： 注释

当使用远程数据访问时，如果使用不支持 Unicode 的 ODBC 驱动程序，则对来自此 ODBC 驱动程序的数据不执行字符集转换。

ODBC 外部服务器定义

定义基于 ODBC 的远程服务器的最常用方法是使其以 ODBC 数据源为基础。为此，可使用 ODBC 数据源管理器创建数据源。

定义完数据源之后，CREATE SERVER 语句中的 USING 子句应引用 ODBC 数据源名称 (DSN)。

例如，要配置名为 mydb2 的 IBM DB2 服务器（其数据源名称也是 mydb2），请使用：

```
CREATE SERVER mydb2
CLASS 'DB2ODBC'
USING 'mydb2';
```

所使用的驱动程序必须与数据库服务器的位数相匹配。

在 Windows 中，还必须定义位数与数据库服务器匹配的系统数据源名名称（系统 DSN）。例如，使用 32 位 ODBC 数据源管理器创建 32 位系统 DSN。用户 DSN 没有位数。

使用连接字符串代替数据源

有一种替代方法可以避免使用数据源名称，即在 CREATE SERVER 语句的 USING 子句中提供连接字符串。若要提供连接字符串，必须知道您所使用的 ODBC 驱动程序的连接参数。例如，与 SAP Sybase IQ 数据库服务器之间的连接可能如下所示：

```
CREATE SERVER TestSA
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=myhost;Server=TestSA;DBN=sample';
```

上述语句定义了与名为 TestSA 的数据库服务器间的连接，该服务器在名为 myhost 的计算机上运行，其中数据库是 sample，使用的协议是 TCP-IP。

CREATE SERVER 语句中的 USING 子句

必须对要访问的每个远程 SAP Sybase IQ 数据库单独发出 CREATE SERVER 语句。例如，如果名为 TestSA 的 SAP Sybase IQ 服务器在计算机 Banana 上运行，并且该服务器拥有三个数据库（db1、db2、db3），则应按如下所示设置远程服务器：

```
CREATE SERVER TestSAdb1
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=Banana;Server=TestSA;DBN=db1';
```

```
CREATE SERVER TestSAdb2
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=Banana;Server=TestSA;DBN=db2';

CREATE SERVER TestSAdb3
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=Banana;Server=TestSA;DBN=db3';
```

如果不指定数据库名称，则远程连接使用远程 SAP Sybase IQ 服务器缺省数据库。

服务器类 SAODBC

服务器类为 SAODBC 的远程服务器是 SAP Sybase IQ 数据库服务器。对于 SAP Sybase IQ 数据源，并没有任何特殊的配置要求。

要访问支持多个数据库的 SAP Sybase IQ 数据库服务器，可以对每一个数据库创建一个用于与其连接的 ODBC 数据源名称。对创建的每个 ODBC 数据源名称执行 CREATE SERVER 语句。

示例

在 CREATE SERVER 语句的 USING 子句中提供连接字符串以连接到 SAP Sybase IQ 数据库。

```
CREATE SERVER TestSA
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=myhost;Server=TestSA;DBN=sample';
```

服务器类 ADSODBC

当您执行 CREATE TABLE 语句后，SAP Sybase IQ 会使用以下数据类型转换关系自动将数据类型转换为对应的 Advantage Database Server 数据类型。

SAP Sybase IQ 数据类型	ADS 缺省数据类型
BIT	Logical
VARBIT(*n*)	Binary(*n*)
LONG VARBIT	Binary(2G)
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Numeric(32)
UNSIGNED TINYINT	Numeric(11)
UNSIGNED SMALLINT	Numeric(11)
UNSIGNED INTEGER	Numeric(11)
UNSIGNED BIGINT	Numeric(32)

SAP Sybase IQ 数据类型	ADS 缺省数据类型
CHAR(*n*)	Character(*n*)
VARCHAR(*n*)	VarChar(*n*)
LONG VARCHAR	VarChar(65000)
NCHAR(*n*)	NChar(*n*)
NVARCHAR(*n*)	NVarChar(*n*)
LONG NVARCHAR	NVarChar(32500)
BINARY(*n*)	Binary(*n*)
VARBINARY(*n*)	Binary(*n*)
LONG BINARY	Binary(2G)
DECIMAL(*precision*, *scale*)	Numeric(*precision*+3)
NUMERIC(*precision*, *scale*)	Numeric(*precision*+3)
SMALLMONEY	Money
MONEY	Money
REAL	Double
DOUBLE	Double
FLOAT(*n*)	Double
DATE	Date
TIME	Time
TIMESTAMP	TimeStamp
TIMESTAMP WITH TIMEZONE	Char(254)
BIGDATE	datetime
BIGDATETIME	datetime
XML	Binary(2G)
ST_GEOMETRY	Binary(2G)
UNIQUEIDENTIFIER	Binary(2G)

服务器类 ASEODBC

服务器类为 ASEODBC 的远程服务器是 Adaptive Server Enterprise（10 及更高版本）数据库服务器。SAP Sybase IQ 需要安装 Adaptive Server Enterprise ODBC 驱动程序和

Open Client 连接库，才能连接到类为 ASEODBC 的远程 Adaptive Server Enterprise 数据库服务器。

注释

- Open Client 的版本应为 11.1.1, EBF 7886 或更高。在安装 ODBC 并配置 SAP Sybase IQ 之前，请先安装 Open Client 并验证与 Adaptive Server Enterprise 服务器的连接。Sybase ODBC 驱动程序的版本应为 11.1.1, EBF 7911 或更高。
- quoted_identifier 选项的本地设置控制是否对 Adaptive Server Enterprise 使用加引号的标识符。例如，如果在本地将 quoted_identifier 选项设置为 [Off]，则会为 Adaptive Server Enterprise 关闭标识符加引号。
- 在 **“Configuration Manager”** 中配置用户数据源的以下属性：
  - **“General” 选项卡** – 在 **“Data Source Name”** 中键入任意值。此值将在 CREATE SERVER 语句的 USING 子句中使用。

    服务器名应匹配 Sybase 接口文件中的服务器的名称。
  - **“Advanced” 选项卡** – 单击 **“Application Using Threads”** 和 **“Enable Quoted Identifiers”** 选项。
  - **“Connection” 选项卡** – 设置字符集字段，使其与您的 SAP Sybase IQ 字符集匹配。

    将语言字段设置为您的用于显示错误消息的首选语言。
  - **“Performance” 选项卡** – 将 **“Prepare Method”** 设置为 **[2-Full]**。

    将 **“Fetch Array Size”** 设置得尽可能大，以获得最佳性能。这会增加内存要求，因为该值是必须被高速缓存到内存中的行数。Adaptive Server Enterprise 建议使用值 100。

    将 **“Select Method”** 设置为 **[0-Cursor]**。

    将 **“Packet Size”** 设置为尽可能大的值。Adaptive Server Enterprise 建议使用值 -1。

    将 **“Connection Cache”** 设置为 1。

数据类型转换：ODBC 和 Adaptive Server Enterprise

当您执行 CREATE TABLE 语句后，SAP Sybase IQ 会自动将数据类型转换为对应的 Adaptive Server Enterprise 数据类型。下表列出了 SAP Sybase IQ 与 Adaptive Server Enterprise 之间的数据类型转换关系。

SAP Sybase IQ 数据类型	Adaptive Server Enterprise 缺省数据类型
BIT	bit
VARBIT(*n*)	if (n <= 255) varbinary(*n*) else image
LONG VARBIT	image

SAP Sybase IQ 数据类型	Adaptive Server Enterprise 缺省数据类型
TINYINT	tinyint
SMALLINT	smallint
INT, INTEGER	int
BIGINT	numeric(20,0)
UNSIGNED TINYINT	tinyint
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	numeric(11,0)
UNSIGNED BIGINT	numeric(20,0)
CHAR(*n*)	if (n <= 255) char(*n*) else text
VARCHAR(*n*)	if (n <= 255) varchar(*n*) else text
LONG VARCHAR	text
NCHAR(*n*)	if (n <= 255) nchar(*n*) else ntext
NVARCHAR(*n*)	if (n <= 255) nvarchar(*n*) else ntext
LONG NVARCHAR	ntext
BINARY(*n*)	if (n <= 255) binary(*n*) else image
VARBINARY(*n*)	if (n <= 255) varbinary(*n*) else image
LONG BINARY	image
DECIMAL(*prec,scale*)	decimal(*prec,scale*)
NUMERIC(*prec,scale*)	numeric(*prec,scale*)
SMALLMONEY	numeric(10,4)
MONEY	numeric(19,4)
REAL	real
DOUBLE	float
FLOAT(*n*)	float(*n*)
DATE	datetime
TIME	datetime
SMALLDATETIME	smalldatetime

SAP Sybase IQ 数据类型	Adaptive Server Enterprise 缺省数据类型
TIMESTAMP	datetime
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	text
ST_GEOMETRY	image
UNIQUEIDENTIFIER	binary(16)

示例

在 CREATE SERVER 语句的 USING 子句中提供连接字符串以连接到 Adaptive Server Enterprise 数据库。

```
CREATE SERVER TestASE
CLASS 'ASEODBC'
USING 'DRIVER=SYBASE ASE ODBC
Driver;Server=TestASE;Port=5000;Database=testdb;UID=username;PWD=pa
ssword'
```

服务器类 DB2ODBC

服务器类为 DB2ODBC 的远程服务器是 IBM DB2 数据库服务器。

注释

- 经 Sybase 认证，可以使用 IBM 的 DB2 Connect 版本 5（安装有补丁程序包 WR09044）。根据该产品的说明进行配置，并测试您的 ODBC 配置。SAP Sybase IQ 对 IBM DB2 数据源的配置没有任何特殊要求。
- 以下是 CREATE EXISTING TABLE 语句的示例，该语句适用于一个 IBM DB2 服务器，其 ODBC 数据源名为 mydb2：

  ```
  CREATE EXISTING TABLE ibmcol
  AT 'mydb2..sysibm.syscolumns';
  ```

数据类型转换：IBM DB2

当您执行 CREATE TABLE 语句时，SAP Sybase IQ 会自动将数据类型转换为对应的 IBM DB2 数据类型。下表列出了 SAP Sybase IQ 与 IBM DB2 数据类型之间的转换关系。

SAP Sybase IQ 数据类型	IBM DB2 缺省数据类型
BIT	smallint
VARBIT(*n*)	如果 (n <= 4000)，则对于位数据为 varchar(*n*)，否则对于位数据为 long varchar
LONG VARBIT	对于位数据为 long varchar
TINYINT	smallint

SAP Sybase IQ 数据类型	IBM DB2 缺省数据类型
SMALLINT	smallint
INTEGER	int
BIGINT	decimal(20.0)
UNSIGNED TINYINT	int
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	decimal(11.0)
UNSIGNED BIGINT	decimal(20.0)
CHAR(*n*)	如果 (n < 255)，则为 char(*n*)，否则，如果 (n <= 4000)，则为 varchar(*n*)，否则为 long varchar
VARCHAR(*n*)	如果 (n <= 4000)，则为 varchar(*n*)，否则为 long varchar
LONG VARCHAR	long varchar
NCHAR(*n*)	不支持
NVARCHAR(*n*)	不支持
LONG NVARCHAR	不支持
BINARY(*n*)	如果 (n <= 4000)，则对于位数据为 varchar(*n*)，否则对于位数据为 long varchar
VARBINARY(*n*)	如果 (n <= 4000)，则对于位数据为 varchar(*n*)，否则对于位数据为 long varchar
LONG BINARY	对于位数据为 long varchar
DECIMAL(*prec,scale*)	decimal(*prec,scale*)
NUMERIC(*prec,scale*)	decimal(*prec,scale*)
SMALLMONEY	decimal(10,4)
MONEY	decimal(19.4)
REAL	real
DOUBLE	float
FLOAT(*n*)	float(*n*)
DATE	date
TIME	time

SAP Sybase IQ 数据类型	IBM DB2 缺省数据类型
TIMESTAMP	timestamp
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	对于位数据为 long varchar
ST_GEOMETRY	对于位数据为 long varchar
UNIQUEIDENTIFIER	对于位数据为 varchar(16)

服务器类 HANAODBC
服务器类为 HANAODBC 的远程服务器是 SAP HANA 数据库服务器。

注释

- 以下是 CREATE EXISTING TABLE 语句的示例，该语句适用于一个 SAP HANA 数据库服务器，其 ODBC 数据源名为 mySAPHANA：

```
CREATE EXISTING TABLE hanatable
AT 'mySAPHANA..dbo.hanatable';
```

数据类型转换：SAP HANA
执行 CREATE TABLE 语句时，SAP Sybase IQ 会自动将数据类型转换为对应的 SAP HANA 数据类型。下表列出了 SAP Sybase IQ 与 SAP HANA 数据类型之间的转换关系。

SAP Sybase IQ 数据类型	SAP HANA 缺省数据类型
BIT	TINYINT
VARBIT(*n*)	如果 (n <= 5000)，则为 VARBINARY(*n*)，否则为 BLOB
LONG VARBIT	BLOB
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT
UNSIGNED TINYINT	TINYINT
UNSIGNED SMALLINT	INTEGER
UNSIGNED INTEGER	BIGINT
UNSIGNED BIGINT	DECIMAL(20.0)

SAP Sybase IQ 数据类型	SAP HANA 缺省数据类型
CHAR(*n*)	如果 (n <= 5000)，则为 VARCHAR(*n*)，否则为 CLOB
VARCHAR(*n*)	如果 (n <= 5000)，则为 VARCHAR(*n*)，否则为 CLOB
LONG VARCHAR	CLOB
NCHAR(*n*)	如果 (n <= 5000)，则为 NVARCHAR(*n*)，否则为 NCLOB
NVARCHAR(*n*)	如果 (n <= 5000)，则为 NVARCHAR(*n*)，否则为 NCLOB
LONG NVARCHAR	NCLOB
BINARY(*n*)	如果 (n <= 5000)，则为 VARBINARY(*n*)，否则为 BLOB
VARBINARY(*n*)	如果 (n <= 5000)，则为 VARBINARY(*n*)，否则为 BLOB
LONG BINARY	BLOB
DECIMAL(*precision*, *scale*)	DECIMAL(*precision*, *scale*)
NUMERIC(*precision*, *scale*)	DECIMAL(*precision*, *scale*)
SMALLMONEY	DECIMAL(13.4)
MONEY	DECIMAL(19.4)
REAL	REAL
DOUBLE	FLOAT
FLOAT(*n*)	FLOAT
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
TIMESTAMP WITH TIMEZONE	VARCHAR(254)
XML	BLOB
ST_GEOMETRY	BLOB
UNIQUEIDENTIFIER	VARBINARY(16)

服务器类 IQODBC

服务器类为 IQODBC 的远程服务器是 SAP Sybase IQ 数据库服务器。对于 SAP Sybase IQ 数据源，并没有任何特殊的配置要求。

要访问支持多个数据库的 SAP Sybase IQ 数据库服务器，可以对每一个数据库创建一个用于与其连接的 ODBC 数据源名称。对创建的每个 ODBC 数据源名称执行 CREATE SERVER 语句。

服务器类 MSACCESSODBC

Access 数据库存储在 `.mdb` 文件中。使用 ODBC 管理器创建一个 ODBC 数据源，并将该数据源映射到一个这类文件。可通过 ODBC 管理器创建新的 `.mdb` 文件。如果在通过 SAP Sybase IQ 创建表时没有指定其它缺省文件，此数据库文件将成为缺省文件。

假定有一个名为 access 的 ODBC 数据源，则可以使用以下任何一个语句访问数据：

- ```
  CREATE TABLE tab1 (a int, b char(10))
  AT 'access...tab1';
  ```
- ```
  CREATE TABLE tab1 (a int, b char(10))
  AT 'access;d:\\pcdb\\data.mdb;;tab1';
  ```
- ```
  CREATE EXISTING TABLE tab1
  AT 'access;d:\\pcdb\\data.mdb;;tab1';
  ```

Access 不支持所有者名称限定，将其保留为空。

数据类型转换：Microsoft Access

SAP Sybase IQ 数据类型	Microsoft Access 缺省数据类型
BIT	TINYINT
VARBIT(*n*)	如果 (n <= 4000)，则为 BINARY(*n*)，否则为 IMAGE
LONG VARBIT	IMAGE
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	DECIMAL(19,0)
UNSIGNED TINYINT	TINYINT
UNSIGNED SMALLINT	INTEGER
UNSIGNED INTEGER	DECIMAL(11,0)
UNSIGNED BIGINT	DECIMAL(20,0)
  ```

SAP Sybase IQ 数据类型	Microsoft Access 缺省数据类型
CHAR(*n*)	如果 (n < 255)，则为 CHARACTER(*n*)，否则为 TEXT
VARCHAR(*n*)	如果 (n < 255)，则为 CHARACTER(*n*)，否则为 TEXT
LONG VARCHAR	TEXT
NCHAR(*n*)	不支持
NVARCHAR(*n*)	不支持
LONG NVARCHAR	不支持
BINARY(*n*)	如果 (n <= 4000)，则为 BINARY(*n*)，否则为 IMAGE
VARBINARY(*n*)	如果 (n <= 4000)，则为 BINARY(*n*)，否则为 IMAGE
LONG BINARY	IMAGE
DECIMAL(*precision, scale*)	DECIMAL(*precision, scale*)
NUMERIC(*precision, scale*)	DECIMAL(*precision, scale*)
SMALLMONEY	MONEY
MONEY	MONEY
REAL	REAL
DOUBLE	FLOAT
FLOAT(*n*)	FLOAT
DATE	DATETIME
TIME	DATETIME
TIMESTAMP	DATETIME
TIMESTAMP WITH TIMEZONE	CHARACTER(254)
XML	XML
ST_GEOMETRY	IMAGE
UNIQUEIDENTIFIER	BINARY(16)

服务器类 MSSODBC

服务器类 MSSODBC 用于通过其中一个 ODBC 驱动程序访问 Microsoft SQL Server。

注释

- 已经使用的 Microsoft SQL Server ODBC 驱动程序版本有：
  - Microsoft SQL Server ODBC 驱动程序版本 06.01.7601
  - Microsoft SQL Server Native Client 版本 10.00.1600
- 以下是 Microsoft SQL Server 的示例：

```
CREATE SERVER mysqlserver
CLASS 'MSSODBC'
USING 'DSN=MSSODBC_cli';

CREATE EXISTING TABLE accounts
AT 'mysqlserver.master.dbo.accounts';
```

- quoted_identifier 选项的本地设置控制是否对 Microsoft SQL Server 使用加引号的标识符。例如，如果在本地将 quoted_identifier 选项设置为 [Off]，则会为 Microsoft SQL Server 关闭标识符加引号。

数据类型转换：Microsoft SQL Server

当您执行 CREATE TABLE 语句后，SAP Sybase IQ 会使用以下数据类型转换关系自动将数据类型转换为对应的 Microsoft SQL Server 数据类型。

SAP Sybase IQ 数据类型	Microsoft SQL Server 缺省数据类型
BIT	bit
VARBIT(*n*)	如果 (n <= 255)，则为 varbinary(*n*)，否则为 image
LONG VARBIT	image
TINYINT	tinyint
SMALLINT	smallint
INTEGER	int
BIGINT	numeric(20.0)
UNSIGNED TINYINT	tinyint
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	numeric(11.0)
UNSIGNED BIGINT	numeric(20.0)
CHAR(*n*)	如果 (n <= 255)，则为 char(*n*)，否则为 text

SAP Sybase IQ 数据类型	Microsoft SQL Server 缺省数据类型
VARCHAR(*n*)	如果 (n <= 255)，则为 varchar(*n*)，否则为 text
LONG VARCHAR	text
NCHAR(*n*)	如果 (n <= 4000)，则为 nchar(*n*)，否则为 ntext
NVARCHAR(*n*)	如果 (n <= 4000)，则为 nvarchar(*n*)，否则为 ntext
LONG NVARCHAR	ntext
BINARY(*n*)	如果 (n <= 255)，则为 binary(*n*)，否则为 image
VARBINARY(*n*)	如果 (n <= 255)，则为 varbinary(*n*)，否则为 image
LONG BINARY	image
DECIMAL(*precision*, *scale*)	decimal(*precision*, *scale*)
NUMERIC(*precision*, *scale*)	numeric(*precision*, *scale*)
SMALLMONEY	smallmoney
MONEY	money
REAL	REAL
DOUBLE	float
FLOAT(*n*)	float(*n*)
DATE	datetime
TIME	datetime
SMALLDATETIME	smalldatetime
DATETIME	datetime
TIMESTAMP	datetime
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	xml
ST_GEOMETRY	image
UNIQUEIDENTIFIER	binary(16)

服务器类 MYSQLODBC

当您执行 CREATE TABLE 语句后，SAP Sybase IQ 会使用以下数据类型转换关系自动将数据类型转换为对应的 MySQL 数据类型。

SAP Sybase IQ 数据类型	MySQL 缺省数据类型
BIT	bit(1)
VARBIT(*n*)	如果 (n <= 4000)，则为 varbinary(*n*)，否则为 longblob
LONG VARBIT	longblob
TINYINT	tinyint unsigned
SMALLINT	smallint
INTEGER	int
BIGINT	bigint
UNSIGNED TINYINT	tinyint unsigned
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	bigint
UNSIGNED BIGINT	decimal(20,0)
CHAR(*n*)	如果 (n < 255)，则为 char(*n*)，否则如果 (n <= 4000)，则为 varchar(*n*)，否则为 longtext
VARCHAR(*n*)	如果 (n <= 4000)，则为 varchar(*n*)，否则为 longtext
LONG VARCHAR	longtext
NCHAR(*n*)	if (n < 255)，则为 national character(*n*)，否则如果 (n <= 4000)，则为 national character varying(*n*)，否则为 longtext
NVARCHAR(*n*)	如果 (n <= 4000)，则为 national character varying(*n*)，否则为 longtext
LONG NVARCHAR	longtext
BINARY(*n*)	如果 (n <= 4000)，则为 varbinary(*n*)，否则为 longblob
VARBINARY(*n*)	如果 (n <= 4000)，则为 varbinary(*n*)，否则为 longblob
LONG BINARY	longblob
DECIMAL(*precision*, *scale*)	decimal(*precision*, *scale*)
NUMERIC(*precision*, *scale*)	decimal(*precision*, *scale*)
SMALLMONEY	decimal(10,4)
MONEY	decimal(19,4)

SAP Sybase IQ 数据类型	MySQL 缺省数据类型
REAL	real
DOUBLE	float
FLOAT(*n*)	float(*n*)
DATE	date
TIME	time
TIMESTAMP	datetime
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	longblob
ST_GEOMETRY	longblob
UNIQUEIDENTIFIER	varbinary(16)

示例

在 CREATE SERVER 语句的 USING 子句中提供连接字符串以连接到 MySQL 数据库。

```
CREATE SERVER TestMySQL
CLASS 'MYSQLODBC'
USING 'DRIVER=MySQL ODBC 5.1
Driver;DATABASE=mydatabase;SERVER=mySQLHost;UID=me;PWD=secret'
```

服务器类 ODBC

没有自己的服务器类的 ODBC 数据源使用 ODBC 服务器类。可以使用任何 ODBC 驱动程序。经 Sybase 认证，可以使用以下 ODBC 数据源：

Microsoft Excel (Microsoft 3.51.171300)
Microsoft FoxPro (Microsoft 3.51.171300)
Lotus Notes SQL

可通过 Microsoft Download Center 上的 Microsoft 数据访问组件（Microsoft Data Access Components，简称 MDAC）获取最新版本的 Microsoft ODBC 驱动程序。以上列出的 Microsoft 驱动程序版本是 MDAC 2.0 的一部分。

Microsoft Excel (Microsoft 3.51.171300)

对于 Excel，可以将每一个 Excel 工作簿在逻辑上视为包含若干个表的数据库。数据库中的表可以映射到 Excel 工作簿中的工作表。当您在 ODBC 驱动程序管理器中配置 ODBC 数据源名时，可以指定与该数据源相关联的缺省工作簿名称。不过，当您执行 CREATE TABLE 语句时，可以替换该缺省值并在位置字符串中指定工作簿名称。这样，就可以只使用一个 ODBC DSN 访问所有 Excel 工作簿。

创建一个名为 excel、连接至 Microsoft Excel ODBC 驱动程序的远程服务器。

```
CREATE SERVER excel
CLASS 'ODBC'
USING 'DRIVER=Microsoft Excel Driver (*.xls);DBQ=d:\
\work1.xls;READONLY=0;DriverID=790'
```

要创建名为 `work1.xls` 的工作簿，其中包含称为 mywork 的工作表（或表），请执行以下语句：

```
CREATE TABLE mywork (a int, b char(20))
AT 'excel;d:\\work1.xls;;mywork';
```

要再创建一个工作表（或表），请执行下面的语句：

```
CREATE TABLE mywork2 (x float, y int)
AT 'excel;d:\\work1.xls;;mywork2';
```

可使用 CREATE EXISTING 语句将现有工作表导入 SAP Sybase IQ，前提是表的第一行包含列名称。

```
CREATE EXISTING TABLE mywork
AT'excel;d:\\work1;;mywork';
```

如果 SAP Sybase IQ 报告未找到该表，则可能需要明确说明您想要映射到的列和行范围。例如：

```
CREATE EXISTING TABLE mywork
AT 'excel;d:\\work1;;mywork$';
```

如果在工作表名中添加 $，则表示应选择整个工作表。

请注意，在 AT 指定的位置字符串中，使用分号而不是句点作为字段分隔符。这是因为句点出现在文件名中。Excel 不支持所有者名字段，因此请不要填写该字段。

Excel 也不支持删除操作。此外，由于 Excel 驱动程序不支持定位更新，因此可能无法进行某些更新。

示例

以下语句可创建名为 TestExcel 的数据库服务器，该服务器使用 ODBC DSN 访问 Excel 工作簿 `LogFile.xlsx` 并将其表导入 SAP Sybase IQ。

```
CREATE SERVER TestExcel
CLASS 'ODBC'
USING 'DRIVER=Microsoft Excel Driver (*.xls);DBQ=c:\\temp\
\LogFile.xlsx;READONLY=0;DriverID=790'

CREATE EXISTING TABLE MyWorkbook
AT 'TestExcel;c:\\temp\\LogFile.xlsx;;Logfile$';

SELECT * FROM MyWorkbook;
```

Microsoft FoxPro (Microsoft 3.51.171300)

您可以将多个 FoxPro 表一起存储在一个 FoxPro 数据库文件 (`.dbc`) 中；也可以将每个表存储在各自的 `.dbf` 文件中。使用 `.dbf` 文件时，请确保文件名已填充到位置字符串中；否则，将使用 SAP Sybase IQ 的启动目录。

```
CREATE TABLE fox1 (a int, b char(20))
AT 'foxpro;d:\\pcdb;;fox1';
```

如果您在 ODBC 驱动程序管理器中选择了 **“Free Table Directory”** 选项，则此语句会创建一个名为 `d:\pcdb\fox1.dbf` 的文件。

Lotus Notes SQL

要获得此驱动程序，请访问 Lotus NotesSQL 网站 http://www.ibm.com/developerworks/lotus/products/notesdomino/notessql/。请阅读附随的文档，以了解有关 Notes 数据与关系表映射关系的说明。可以轻松地将 SAP Sybase IQ 表映射到 Notes 表单。

下面介绍如何设置 SAP Sybase IQ 以访问 Lotus Notes 连接。

- 确保 Lotus Notes 程序文件夹位于路径中（如 C:\Program Files (x86)\IBM\Lotus\Notes）。
- 使用 NotesSQL ODBC 驱动程序创建 32 位 ODBC 数据源。此示例使用 `names.nsf` 数据库。应启用 **“Map Special Characters”** 选项。对于本示例，**“数据源名”** 是 `my_notes_dsn`。
- 使用连接到 32 位数据库服务器的 Interactive SQL 创建远程数据访问服务器。下面是一个示例：

  ```
  CREATE SERVER NotesContacts
  CLASS 'ODBC'
  USING 'my_notes_dsn';
  ```
- 为 Lotus Notes 服务器创建外部登录。下面是一个示例：

  ```
  CREATE EXTERNLOGIN "DBA" TO "NotesContacts"
  REMOTE LOGIN 'John Doe/SYBASE' IDENTIFIED BY 'MyNotesPassword';
  ```
- 将 Person 表单中的部分列映射到 SAP Sybase IQ 表中：

  ```
  CREATE EXISTING TABLE PersonDetails
  ( DisplayName CHAR(254),
    DisplayMailAddress CHAR(254),
    JobTitle CHAR(254),
    CompanyName CHAR(254),
    Department CHAR(254),
    Location CHAR(254),
    OfficePhoneNumber CHAR(254) )
  AT 'NotesContacts...Person';
  ```
- 查询该表：

  ```
  SELECT * FROM PersonDetails
  WHERE Location LIKE 'Waterloo%';
  ```

服务器类 ORAODBC

服务器类为 ORAODBC 的远程服务器是 8.0 或更高版的 Oracle 数据库。

注释

- 经 Sybase 认证，可以使用 Oracle 数据库 8.0.03 版 ODBC 驱动程序。根据该产品的说明进行配置，并测试您的 ODBC 配置。
- 以下是名为 myora 的 Oracle Database 服务器的 CREATE EXISTING TABLE 语句的示例：

```
CREATE EXISTING TABLE employees
AT 'myora.database.owner.employees';
```

数据类型转换：Oracle 数据库

执行 CREATE TABLE 语句时，SAP Sybase IQ 会使用以下数据类型转换关系自动将数据类型转换为对应的 Oracle 数据库数据类型。

SAP Sybase IQ 数据类型	Oracle 数据库数据类型
BIT	number(1.0)
VARBIT(*n*)	如果 (n <= 255)，则为 raw(*n*)，否则为 long raw
LONG VARBIT	long raw
TINYINT	number(3.0)
SMALLINT	number(5.0)
INTEGER	number(11.0)
BIGINT	number(20.0)
UNSIGNED TINYINT	number(3.0)
UNSIGNED SMALLINT	number(5.0)
UNSIGNED INTEGER	number(11.0)
UNSIGNED BIGINT	number(20.0)
CHAR(*n*)	如果 (n <= 255)，则为 char(*n*)，否则为 long
VARCHAR(*n*)	如果 (n <= 2000)，则为 varchar(*n*)，否则为 long
LONG VARCHAR	long
NCHAR(*n*)	如果 (n <= 255)，则为 nchar(*n*)，否则为 nclob
NVARCHAR(*n*)	如果 (n <= 2000)，则为 nvarchar(*n*)，否则为 nclob

SAP Sybase IQ 数据类型	Oracle 数据库数据类型
LONG NVARCHAR	nclob
BINARY(*n*)	如果 (n > 255)，则为 long raw，否则为 raw(*n*)
VARBINARY(*n*)	如果 (n > 255)，则为 long raw，否则为 raw(*n*)
LONG BINARY	long raw
DECIMAL(*precision*, *scale*)	number(*precision*, *scale*)
NUMERIC(*precision*, *scale*)	number(*precision*, *scale*)
SMALLMONEY	numeric(13,4)
MONEY	number(19,4)
REAL	real
DOUBLE	float
FLOAT(*n*)	float
DATE	date
TIME	date
TIMESTAMP	date
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	long raw
ST_GEOMETRY	long raw
UNIQUEIDENTIFIER	raw(16)

示例

在 CREATE SERVER 语句的 USING 子句中提供连接字符串以连接到 Oracle 数据库。

```
CREATE SERVER TestOracle
CLASS 'ORAODBC'
USING 'DRIVER=Oracle ODBC
Driver;DBQ=mydatabase;UID=username;PWD=password'
```

远程服务器

必须先定义远程对象所在的远程服务器，然后才能将远程对象映射到本地代理表。

创建远程服务器

使用 **CREATE SERVER** 语句设置远程服务器定义。

对于某些系统（包括 SAP Sybase IQ 和 SQL Anywhere），每个数据源都描述一个数据库，因此每个数据库都需要一个单独的远程服务器定义。

另请参见

- CREATE SERVER 语句 （第 656 页）

访问远程 Oracle 数据前

要访问远程 Oracle 数据，请使用必备软件配置您的系统。

1. 检查前提条件
   检查系统是否有使用组件集成服务 (CIS) 访问 Oracle 数据所需的软件组件。
2. 创建 Oracle 数据源名称
   使用 iqdsn 实用程序在 .odbc.ini 文件中创建一个条目。
3. 为 Oracle 数据访问设置环境变量
   启动 SAP Sybase IQ 服务器访问 Oracle 数据前，必须设置特定环境变量。
4. 启动 SAP Sybase IQ 服务器
   启动可用作访问 Oracle 数据的前端的 SAP Sybase IQ 服务器。

检查前提条件

检查系统是否有使用组件集成服务 (CIS) 访问 Oracle 数据所需的软件组件。

前提条件为：

- Oracle 数据库
- Oracle 客户端软件（基本包），包括 `network/admin/tnsnames.ora` 文件。
- 平台特定的驱动程序（与 SAP Sybase IQ 一同安装）：

平台	文件
AIX 64	`$IQDIR16/libxx/libdboraodbc12_r.so`
HPiUX	`$IQDIR16/libxx/libdboraodbc12_r.so.1`
Linux64	`$IQDIR16/libxx/libdboraodbc12_r.so.1`
SunOS64	`$IQDIR16/libxx/libdboraodbc12_r.so.1`
WinAMD64	`%$IQDIR16%\bin64\dboraodbc12.dll`

创建 Oracle 数据源名称

使用 **iqdsn** 实用程序在 `.odbc.ini` 文件中创建一个条目。

1. 显示 Oracle 连接关键字：

   ```
   % iqdsn -cl -or
   ```

   ```
   Driver
   UserID              UID
   Password            PWD
   SID                 SID
   Encrypted Password  ENP
   ProcResults         PROC
   ArraySize           SIZE
   EnableMSDTC         EDTC
   ProcOwner           POWNER
   ```

2. 创建 `.odbc.ini` 文件条目：

   ```
   % iqdsn -or -y -w "MyOra2" -c
   "UID=system;PWD=manager;SID=QAORA"
   ```

   ```
   [MyOra2]
   Driver=/Sybase/IQ-16_0/lib64/libdboraodbc12_r.so
   UserID=system
   Password=manager
   SID=QAORA
   ```

为 Oracle 数据访问设置环境变量

启动 SAP Sybase IQ 服务器访问 Oracle 数据前，必须设置特定环境变量。

设置以下变量以便访问 Oracle：

- ORACLE_HOME
  `setenv ORACLE_HOME`
- ODBCINI
  `setenv ODBCINI` *`<location of .odbc.ini file with Oracle entry>`*
- 平台的库路径变量

平台	命令
AIX	`setenv LIBPATH` *`<path to platform-specific Oracle client directory>`* `$LIBPATH`
其它 UNIX 平台	`setenv LD_LIBRARY_PATH` *`<path to platform-specific Oracle client directory>`*`;$LD_LIBRARY_PATH`

启动 SAP Sybase IQ 服务器
启动可用作访问 Oracle 数据的前端的 SAP Sybase IQ 服务器。

```
start_iq -n myserver
```

连接到 Oracle 数据库
通过组件集成服务将 SAP Sybase IQ 连接到远程 Oracle 数据。

前提条件

登录到 **dbisql** 或 **iqisql**。

过程

1. 使用 .odbc.ini 文件中的数据源名称创建服务器：
   ```
   CREATE SERVER myora CLASS 'oraodbc' USING 'MyOra2'
   ```
2. 创建外部登录名：
   ```
   CREATE EXTERNLOGIN DBA TO myora REMOTE LOGIN system
   IDENTIFIED BY manager
   ```
3. 确认连接：
   ```
   sp_remote_tables myora
   ```
4. 创建 Oracle 数据表：
   ```
   CREATE EXISTING TABLE my_oratable at
   'myora..system.oratable'
   ```
5. 通过选择数据检验连接是否有效：
   ```
   SELECT * FROM my_oratable
   ```

排除 Oracle 数据库访问故障
如果 Oracle 数据访问返回错误，请检查相应的配置组件。

1. 装载驱动程序时出错
   驱动程序装载错误可能指示环境变量或配置信息文件出现问题。
2. 解析连接标识符时出错
   解析连接标识符错误可能指示 Oracle 定义、环境变量或配置信息文件出现问题。

装载驱动程序时出错
驱动程序装载错误可能指示环境变量或配置信息文件出现问题。

如果您遇到“无法装载驱动程序”错误：

- 检查 .odbc.ini 条目是否列出了正确的驱动程序。

- 检查 Oracle 客户端软件是否已添加到 LD_LIBRARY_PATH 定义中。

解析连接标识符时出错
解析连接标识符错误可能指示 Oracle 定义、环境变量或配置信息文件出现问题。

如果您看到错误“ORA-12154: TNS:无法解析连接标识符”：

- 检查 Oracle 定义是否正确。
- 检查 ORACLE_HOME 设置是否正确。
- 检查 .odbc.ini 中输入的网关系统标识符 (SID) 是否正确。

装载没有本机类的远程数据
使用 DirectConnect™ 装载数据。

本机类使用 DirectConnect 来访问远程数据源：

- 在 64 位 UNIX 平台上
- 在没有 ODBC 驱动程序可用的 32 位平台上（例如 Microsoft SQL Server）

将 MS SQL Server 数据装载到 UNIX 上的 SAP Sybase IQ 服务器
本远程数据示例将 MS SQL Server 数据装载到 UNIX 上的 SAP Sybase IQ 服务器中。

对于本示例，假设以下情况：

- 名为 *mssql* 的 Enterprise Connect Data Access (ECDA) 服务器位于 UNIX 主机 *myhostname* 端口 12530 上。
- 要从主机 *myhostname* 端口 1433 上的 MS SQL Server *2000* 中检索数据。

1. 使用 DirectConnect 文档为您的数据源配置 DirectConnect。
2. 确保 ECDA 服务器 (*mssql*) 列出在 SAP Sybase IQ 接口文件中：

```
mssql
master tcp ether myhostname 12530
query tcp ether myhostname 12530
```

3. 使用服务器 *mssql* 的用户 ID 和口令，添加新用户：

```
isql -Udba -Psql -Stst_iqdemo
grant connect to chill identified by chill
grant dba to chill
```

4. 以新用户身份登录，在 SAP Sybase IQ 上创建本地表：

```
isql -Uchill -Pchill -Stst_iqdemo
create table billing(status char(1), name varchar(20), telno int)
```

5. 插入数据：

```
insert into billing location ‘mssql.pubs’ { select * from
billing }
```

查询没有本机类的数据

请遵循以下准则来查询没有本机类的数据。

1. 使用远程服务器和代理配置 ASE/CIS，以通过 DirectConnect 连接。例如，将 DirectConnect for Oracle 用于 Oracle 服务器。
2. 使用 Adaptive Server 服务器的 ASEJDBCC 类通过远程服务器配置 SAP Sybase IQ。（ASEODBC 类不可用，因为 Adaptive Server 没有 64 位 Unix ODBC 驱动程序。）
3. 使用 **CREATE EXISTING TABLE** 语句创建指向 ASE 中的代理表的代理表，而 ASE 中的代理表又指向 Oracle。

使用 DirectConnect 和 UNIX 中的代理表查询远程数据

使用 DirectConnect 查询数据。

本示例说明如何访问 MS SQL Server 数据。对于本示例，假定满足以下条件：

- SAP Sybase IQ 服务器位于主机 *myhostname* 上，端口为 7594。
- Adaptive Server 服务器位于主机 *myhostname* 上，端口为 4101。
- 名为 *mssql* 的 Enterprise Connect Data Access (ECDA) 服务器位于主机 *myhostname* 端口 12530 上。
- 要从主机 *myhostname* 端口 1433 上的 MS SQL Server *2000* 中检索数据。

设置 Adaptive Server 以查询 MS SQL Server

设置 Adaptive Server 和组件集成服务 (CIS) 以通过 DirectConnect 查询 MS SQL Server。

对于本示例，假设服务器名为 *jones_1207*。

1. 向 Adaptive Server 接口文件添加一个条目，用于连接至 *mssql*：

```
mssql

master tcp ether hostname 12530

query tcp ether hostname 12530
```

2. 从 ASE 服务器启用 CIS 和远程过程调用处理。例如，如果缺省情况下，CIS 已启用：

```
sp_configure 'enable cis'

Parameter Name Default Memory Used Config Value Run Value

enable cis                          1          0
1              1

(1 row affected)
(return status=0)

sp_configure 'cis rpc handling', 1

Parameter Name Default Memory Used Config Value Run Value
```

```
enable cis                          0          0
0             1
```

```
(1 row affected)
Configuration option changed. The SQL Server need not be restarted
since the option is dynamic.
```

3. 将 DirectConnect 服务器添加到 Adaptive Server 服务器的 SYSSERVERS 系统表。

```
sp_addserver mssql, direct_connect, mssql
```

```
Adding server 'mssql', physical name 'mssql'
Server added.
(Return status=0)
```

4. 在 Adaptive Server 中创建用户，以在 SAP Sybase IQ 中连接到 Adaptive Server。

```
sp_addlogin tst, tsttst
```

```
Password correctly set.
Account unlocked. New login created.
(return status = 0)
```

```
grant role sa_role to tst
use tst_db
sp_adduser tst
```

```
New user added.
(return status = 0)
```

5. 从 master 数据库添加外部登录名：

```
use master
sp_addexternlogin mssql, tst, chill, chill
```

```
User 'tst' will be known as 'chill' in remote server 'mssql'.
(return status = 0)
```

6. 从所需数据库，以所添加用户身份创建 ASE 代理表：

```
isql -Utst -Ttsttst
use test_db
create proxy_table billing_tst at 'mssql.pubs..billing'
select * from billing_tst
```

```
status                name                telno
------                ------------              -----
D                BOTANICALLY               1
B                BOTANICALL                2
(2 rows affected)
```

设置 SAP Sybase IQ 以连接到 Adaptive Server 服务器

按照以下步骤操作，以查询 Adaptive Server 数据。

1. 向 SAP Sybase IQ 接口文件添加一个条目：

```
jones_1207
master tcp ether jones 4101
query tcp ether jones 4101
```

2. 创建连接至 Adaptive Server 的用户：

```
GRANT CONNECT TO tst IDENTIFIED BY tsttst
GRANT dba TO tst
```

3. 以所添加用户的身份登录，创建“asejdbc”服务器类并添加外部登录名：

```
isql -Utst -Ptsttst -Stst_iqdemo
CREATE SERVER jones_1207 CLASS 'asejdbc' USING 'jones:4101/tst_db'
CREATE EXISTING TABLE billing_iq AT
'jones_1207.tst_db..billing_txt'
SELECT * from billing_iq
```

```
status                name          telno
------    -----------   -----
D         BOTANICALLY   1
B         BOTANICALL    2
(2 rows affected)
```

删除远程服务器

使用 **DROP SERVER** 语句从 ISYSSERVER 系统表中删除远程服务器。

必须删除在该服务器上定义的所有远程表才能使此操作成功。

示例

以下语句删除名为 RemoteSA 的服务器：

```
DROP SERVER RemoteSA;
```

另请参见

- DROP SERVER 语句 （第 673 页）

更改远程服务器

使用 **ALTER SERVER** 语句修改服务器的属性。这些更改直到下一次与远程服务器连接时才生效。

执行 **ALTER SERVER** 语句。

下面的语句将名为 RemoteASE 的服务器的服务器类更改为 aseodbc。在此示例中，该服务器的数据源名称是 RemoteASE

```
ALTER SERVER RemoteASE
CLASS 'aseodbc';
```

另请参见

- ALTER SERVER 语句 （第 651 页）

列出远程服务器上的表 (SQL)

您可以使用系统过程来查看远程服务器上所有表的部分列表或完整列表。

前提条件

无。

过程

调用 sp_remote_tables 系统过程以返回远程服务器上表的列表。

如果指定 *@table_name* 或 *@table_owner*，则列表中仅包含与这些参数匹配的表。

返回所有表的列表或表的部分列表。

远程服务器的功能

sp_servercaps 系统过程显示有关远程服务器功能的信息。SAP Sybase IQ 使用此功能信息来确定可以向远程服务器传递多少 SQL 语句。

您还可以通过查询 SYSCAPABILITY 和 SYSCAPABILITYNAME 系统视图查看远程服务器的功能信息。在 SAP Sybase IQ 第一次连接到远程服务器前，这些系统视图是空的。

当使用 sp_servercaps 系统过程时，指定的 *server-name* 必须与 CREATE SERVER 语句中使用的 *server-name* 相同。

按如下方式执行存储过程 sp_servercaps：

```
CALL sp_servercaps('server-name');
```

外部登录

SAP Sybase IQ 代表其客户端连接到远程服务器时会使用其客户端的名称和口令。但是，您可以通过创建外部登录来替换这一行为。

外部登录名是与远程服务器通信时使用的替代登录名和口令。

当 SAP Sybase IQ 连接到远程服务器时，**INSERT...LOCATION** 将对当前连接的用户 ID 使用远程登录，但前提是已使用 **CREATE EXTERNLOGIN** 创建远程登录，并且已使用 **CREATE SERVER** 语句定义了远程服务器。

如果未定义远程服务器，或者未为当前连接的用户 ID 创建远程登录，则 SAP Sybase IQ 将使用当前连接的用户 ID 和口令连接。

注意： 如果您使用的是缺省用户 ID 和口令，并且用户更改了口令，则您必须停止并重新启动服务器，新的口令才能在远程服务器上生效。更改缺省用户 ID 的口令不会影响使用 **CREATE EXTERNLOGIN** 创建的远程登录。

如果使用集成登录，SAP Sybase IQ 客户端的 SAP Sybase IQ 名称和口令将与 SAP Sybase IQ 用户 ID 在 `syslogins` 中所映射到的数据库登录 ID 和口令相同。

代理表

远程数据的位置透明性是通过创建映射到远程对象的本地**代理表**来实现的。您可以使用代理表访问远程数据库以代理表候选对象方式导出的任何对象（包括表、视图和实例化视图）。使用以下语句之一创建代理表：

- 如果该表已存在于远程存储位置，则使用 CREATE EXISTING TABLE 语句。此语句为远程服务器上的现有表定义代理表。

- 如果该表未存在于远程存储位置，则使用 CREATE TABLE 语句。此语句在远程服务器上创建新表，并且还为该表定义代理表。

注意： 处于保存点中时不能修改代理表中的数据。

当触发器在代理表上触发时，使用的权限是导致触发器触发的用户的权限，而不是代理表所有者的权限。

代理表位置

CREATE TABLE 和 CREATE EXISTING TABLE 语句都可以使用 AT 关键字定义现有对象的位置。该位置字符串由四部分组成，每部分都由句点或分号隔开。分号分隔符允许在数据库和所有者字段中使用文件名和扩展名。

AT 子句的语法是：

```
... AT 'server.database.owner.table-name'
```

- **server** – 这是在当前数据库中使用的服务器名称（在 CREATE SERVER 语句中指定的名称）。对于所有远程数据源，此字段是必需的。
- **database** – database 字段的含义取决于数据源。此字段有时不适用，应该留空。但是，分隔符仍是必需的。

  如果数据源是 Adaptive Server Enterprise，则 *database* 指定表所在的数据库。例如 master 或 pubs2。

  如果数据源是 SAP Sybase IQ，则此字段不适用；将其留空。

  如果数据源是 Excel、Lotus Notes 或 Access，则必须包括包含表的文件的名称。如果该文件名中包含句点，请使用分号分隔符。
- **owner** – 如果数据库支持所有权的概念，则此字段表示所有者名。只有当多个所有者具有使用相同名称的表时才需要此字段。
- **table-name** – 此字段指定表的名称。如果是 Excel 电子表格，则这是工作簿中工作表的名称。如果将 *table-name* 留空，则假定远程表名与本地代理表名相同。

示例

以下示例说明位置字符串的用法：

- SAP Sybase IQ：

```
'RemoteSA..GROUPO.Employees'
```

- Adaptive Server Enterprise：

```
'RemoteASE.pubs2.dbo.publishers'
```

- Excel：

```
'RemoteExcel;d:\pcdb\quarter3.xls;;sheet1$'
```

- Access：

```
'RemoteAccessDB;\\server1\production\inventory.mdb;;parts'
```

创建代理表 (SQL)

可使用 CREATE TABLE 或 CREATE EXISTING TABLE 语句在 Interactive SQL 中创建代理表。

前提条件

必须具有 CREATE PROXY TABLE 系统特权才能创建由自己所拥有的代理表。必须具有 CREATE ANY TABLE 或 CREATE ANY OBJECT 系统特权才能创建由他人所拥有的代理表。

过程

CREATE TABLE 语句在远程服务器上创建新表，如果使用 AT 子句，则可以为该表定义代理表。使用 SAP Sybase IQ 数据类型定义列。SAP Sybase IQ 会自动将数据转换为远程服务器的本机类型。

如果您使用 CREATE TABLE 语句创建本地和远程表，并随后使用 DROP TABLE 语句删除代理表，则远程表也将被删除。但是，您可以使用 DROP TABLE 语句删除使用 CREATE EXISTING TABLE 语句创建的代理表。在这种情况下，不会删除远程表。

CREATE EXISTING TABLE 语句创建映射到远程服务器上现有表的代理表。SAP Sybase IQ 从位于远程位置的对象派生列属性和索引信息。

1. 连接到主机数据库。
2. 执行 CREATE EXISTING TABLE 语句。

将创建代理表。

另请参见

- CREATE EXISTING TABLE 语句 （第 653 页）
- CREATE TABLE 语句 （第 658 页）

列出远程表上的列

在执行 CREATE EXISTING TABLE 语句之前，获得远程表上可用列的列表可能很有帮助。sp_remote_columns 系统过程生成远程表上列的列表和这些数据类型的说明。以下是 sp_remote_columns 系统过程的语法：

```
CALL sp_remote_columns( @server_name, @table_name [, @table_owner [,
@table_qualifier ] ] )
```

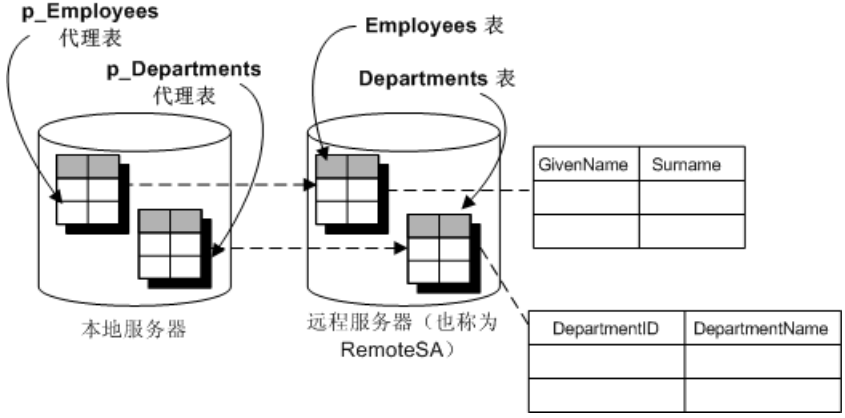
如果给定了表名、所有者或数据库名，则列的列表仅限于那些符合条件的列。

例如，以下语句返回名为 asetest 的 Adaptive Server Enterprise 服务器上 production 数据库中 sysobjects 表中列的列表：

```
CALL sp_remote_columns('asetest, 'sysobjects', null, 'production');
```

远程表之间的连接

下图说明本地数据库服务器上的代理表映射到了远程服务器 RemoteSA 上 SAP Sybase IQ 示例数据库中的远程表 Employees 和 Departments。



您可以在不同 SAP Sybase IQ 数据库上的表之间使用连接。下面是一个简单的示例，该示例只使用一个数据库来说明这些原则。

示例

在两个远程表之间进行连接：

1. 创建一个名为 `empty.db` 的新数据库。
   此数据库未保存任何数据。它只用来定义远程对象和访问 SAP Sybase IQ 示例数据库。
2. 启动运行 `empty.db` 的数据库服务器。可通过运行以下命令完成此操作：

   ```
   iqsrv16 empty
   ```
3. 从 Interactive SQL 以用户 DBA 身份连接到 `empty.db`。
4. 在新数据库中，创建一个名为 RemoteSA 的远程服务器。其服务器类为 SAODBC，连接字符串引用 SAP Sybase IQ 16 Demo ODBC 数据源：

   ```
   CREATE SERVER RemoteSA
   CLASS 'SAODBC'
   USING 'SAP Sybase IQ 16 Demo';
   ```
5. 在本示例中，您在远程数据库和本地数据库上使用相同的用户 ID 和口令，因此不需要外部登录。
   有时，连接远程服务器上的数据库时必须提供用户 ID 和口令。在新数据库中，您能够创建到远程服务器的外部登录。在本例中，为简单起见，本地登录名和远程用户 ID 均为 DBA：

   ```
   CREATE EXTERNLOGIN DBA
   TO RemoteSA
   REMOTE LOGIN DBA
   IDENTIFIED BY sql;
   ```

6. 定义 p_Employees 代理表：

```
CREATE EXISTING TABLE p_Employees
AT 'RemoteSA..GROUPO.Employees';
```

7. 定义 p_Departments 代理表：

```
CREATE EXISTING TABLE p_Departments
AT 'RemoteSA..GROUPO.Departments';
```

8. 在 SELECT 语句中使用代理表执行连接。

```
SELECT GivenName, Surname, DepartmentName
FROM p_Employees JOIN p_Departments
ON p_Employees.DepartmentID = p_Departments.DepartmentID
ORDER BY Surname;
```

多个本地数据库中两个表之间的连接

SAP Sybase IQ 服务器上可能有多个本地数据库在同时运行。通过将其它本地 SAP Sybase IQ 数据库中的表定义为远程表，您可以执行跨数据库连接。

示例

假设您正在使用数据库 db1，您要访问数据库 db2 的表中的数据。您需要建立指向数据库 db2 中的表的代理表定义。例如，在名为 RemoteSA 的 SAP Sybase IQ 服务器上，您可能有三个数据库可用：db1、db2 和 db3。

1. 如果使用 ODBC，则需要为您将访问的每个数据库创建一个 ODBC 数据源名称。
2. 连接到将从中执行连接的数据库。例如，连接到 db1。
3. 为每个您将访问的其它本地数据库执行 CREATE SERVER 语句。这将建立到 SAP Sybase IQ 服务器的**回送**连接。

```
CREATE SERVER remote_db2
CLASS 'SAODBC'
USING 'RemoteSA_db2';
CREATE SERVER remote_db3
CLASS 'SAODBC'
USING 'RemoteSA_db3';
```

4. 通过执行 CREATE EXISTING TABLE 语句为您要访问的其它数据库中的表创建代理表定义。

```
CREATE EXISTING TABLE Employees
AT 'remote_db2...Employees';
```

本机语句和远程服务器

使用 FORWARD TO 语句将一个或多个语句以其本机语法发送到远程服务器。可以通过两种方式使用此语句：

- 将语句发送到远程服务器。
- 让 SAP Sybase IQ 进入直通模式以便向远程服务器发送一系列语句。

FORWARD TO 语句可用于验证是否正确配置了服务器。如果您将语句发送到远程服务器，SAP Sybase IQ 未返回错误消息，则远程服务器配置正确。

不能在过程或批处理中使用 FORWARD TO 语句。

如果无法建立与指定服务器的连接，则会向用户返回一条消息。如果建立了连接，则会将任何结果转换为客户端程序可以识别的形式。

示例 1

下面的语句通过选择版本字符串校验与名为 RemoteASE 的服务器的连接：

```
FORWARD TO RemoteASE {SELECT @@version};
```

示例 2

下面的语句显示与名为 RemoteASE 的服务器的直通会话：

```
FORWARD TO RemoteASE;
    SELECT * FROM titles;
    SELECT * FROM authors;
FORWARD TO;
```

远程过程调用 (RPC)

SAP Sybase IQ 用户可以向支持过程调用功能的远程服务器发出过程调用。

SAP Sybase IQ、SQL Anywhere、Adaptive Server、Oracle 和 DB2 支持此功能。发出远程过程调用与使用本地过程调用类似。

创建远程过程

管理员可在 Interactive SQL 中创建远程过程。

前提条件

您必须具有 MANAGE REPLICATION 系统特权。

过程

如果远程过程可以返回结果集，即使并不是始终都返回，本地过程定义也必须包含 RESULT 子句。

1. 连接到主机数据库。
2. 执行一个用以定义过程的语句。例如：

   ```
   CREATE PROCEDURE RemoteWho()
   AT 'bostonase.master.dbo.sp_who'
   ```

本示例在调用远程过程时指定一个参数：

```
CREATE PROCEDURE RemoteUser ( IN username CHAR( 30 ) )
AT 'bostonase.master.dbo.sp_helpuser';
CALL RemoteUser( 'joe' );
```

远程事务

涉及远程服务器的事务管理使用*两阶段提交*协议。

SAP Sybase IQ 执行的策略可以确保大多数情况的事务完整性。

远程事务管理

管理涉及远程服务器的事务的方法使用两阶段提交协议。SAP Sybase IQ 执行的策略可以确保大多数情况的事务完整性。但是，当一个事务中调用了多个远程服务器时，仍然存在分布式工作单位处于不确定状态的可能性。即使使用了两阶段提交协议，也不包括恢复过程。

管理用户事务的一般逻辑如下：

1. SAP Sybase IQ 以 BEGIN TRANSACTION 通知向远程服务器宣布工作。
2. 当准备好提交事务时，SAP Sybase IQ 向已成为事务一部分的每个远程服务器发送一个 PREPARE TRANSACTION 通知。这可以确保远程服务器准备好提交事务。
3. 如果 PREPARE TRANSACTION 请求失败，则将指示所有远程服务器回退当前事务。

   如果所有 PREPARE TRANSACTION 请求都成功，则服务器将向该事务涉及的每个远程服务器发送一个 COMMIT TRANSACTION 请求。

以 BEGIN TRANSACTION 开头的任何语句都可以开始一个事务。其它语句被发送到远程服务器作为单个、远程工作单元执行。

远程事务限制

远程事务管理具有保存点和嵌套语句限制。

事务管理的限制包括：

- 不将保存点传播到远程服务器。
- 如果涉及远程服务器的事务中包含嵌套 的 **BEGIN TRANSACTION** 和 **COMMIT TRANSACTION** 语句， 则只处理最外层的一组语句。系统不会将包含 **BEGIN TRANSACTION** 和 **COMMIT TRANSACTION** 语句的不一致的一组语句传输到远程服务器。

内部操作

本节介绍 SAP Sybase IQ 代表客户端应用程序在远程服务器上执行的基础步骤。

查询分析

当数据库服务器从客户端接收到语句时，将对该语句进行分析。如果语句不是有效的 SQL Anywhere SQL 语句，数据库服务器将报告错误。

查询规范化

在查询规范化中，系统将验证引用的对象并检查数据类型兼容性。

例如，请考虑下列查询：

```
SELECT *
FROM t1
WHERE c1 = 10
```

查询规范化阶段验证系统表中是否存在具有列 `c1` 的表 `t1`。它还验证列 `c1` 的数据类型是否和值 10 兼容。例如，如果该列的数据类型为 `DATETIME`，则系统将拒绝此语句。

查询预处理

查询预处理准备对查询进行优化。它可能会更改语句的表示形式，因此，即使在语义上等效，SAP Sybase IQ 生成的用于传递到远程服务器的 SQL 语句在语句构成上也将不同于原始语句。

预处理执行视图扩展，以便查询可以对视图引用的表执行操作。可以对表达式进行重新排序并对子查询进行转换以提高处理效率。例如，可以将某些子查询转换为连接。

语句的完整直通

为提高效率，SAP Sybase IQ 会尽可能多地将语句传递给远程服务器。通常，这将是最初提供给 SAP Sybase IQ 的完整语句。

当满足以下条件时，SAP Sybase IQ 将传递出完整的语句：

- 语句中的所有表都驻留在同一个远程服务器上。
- 远程服务器能够处理语句中的所有语法。

在少数情况下，让 SAP Sybase IQ 执行某些工作实际上可能比由远程服务器来执行这些工作更高效。例如，SAP Sybase IQ 可能具有更好的排序算法。在这种情况下，您可以考虑使用 ALTER SERVER 语句变更远程服务器的功能。

语句的部分直通

如果某个语句包含对多个服务器的引用，或者使用远程服务器不支持的 SQL 功能，则会将查询分解为多个较简单的部分。

SELECT

通过删除不能传递的部分并让 SAP Sybase IQ 执行工作，可以对 SELECT 语句进行分解。例如，假设远程服务器不能处理以下语句中的 ATAN2 函数：

```
SELECT a,b,c
WHERE ATAN2( b, 10 ) > 3
AND c = 10;
```

发送到远程服务器的语句将被转换为：

```
SELECT a,b,c WHERE c = 10;
```

然后，SAP Sybase IQ 在本地将 [`WHERE ATAN2( b, 10 ) > 3`] 应用到中间结果集。

连接

如语句中含有在多个位置的不同表之间的连接，IQ 会尝试将并置表连接推送到其所在的服务器上。然后，IQ 会将该连接的结果与其它远程表或本地表的结果连接。IQ 总是倾向于将尽可能多的连接工作推送到远程服务器上。当 IQ 连接远程表与本地 IQ 表时，IQ 可选择使用其支持的任何连接算法。

算法的选择以预计的成本为依据。这些算法可包含嵌套循环、散列或排序-合并连接。

当在 IQ 与远程表间选择嵌套循环连接时，应尽力确保远程表是连接中最外侧的表。原因在于查找远程表的网络 I/O 成本要比查找本地表高很多。

UPDATE 和 DELETE

找到符合条件的行时，如果 SAP Sybase IQ 不能将 UPDATE 或 DELETE 语句完整地传递给远程服务器，则它必须将该语句转换成一个表扫描，该表扫描包含原始 WHERE 子句的尽可能多的部分，后跟指定 WHERE CURRENT OF *cursor-name* 的定位 UPDATE 或 DELETE 语句。

例如，如果远程服务器不支持函数 ATAN2：

```
UPDATE t1
SET a = atan2( b, 10 )
WHERE b > 5;
```

将被转换为以下内容：

```
SELECT a,b
FROM t1
WHERE  b > 5;
```

每当找到一行时，SAP Sybase IQ 都会计算出 a 的新值并执行：

```
UPDATE t1
SET a = 'new value'
WHERE CURRENT OF CURSOR;
```

如果 a 已有等于该新值的值，则没有必要执行定位的 UPDATE，因此也不会远程发送定位的 UPDATE。

为了处理需要表扫描的 UPDATE 或 DELETE 语句，远程数据源必须支持执行定位 UPDATE 或 DELETE (WHERE CURRENT OF *cursor-name*) 的功能。某些数据源不支持此功能。

注意： 不能更新临时表

如果需要中间临时表，则不能执行UPDATE或DELETE。这种情况发生在具有ORDER BY 的查询和某些具有子查询的查询中。

远程数据访问故障排除

本节提供远程服务器访问故障排除的一些建议。

远程数据不支持的功能

不支持对远程数据执行以下 SAP Sybase IQ 功能：

- 对远程表执行 ALTER TABLE 语句。
- 在代理表上定义的触发器。
- 引用远程表的外键。
- READTEXT、WRITETEXT 和 TEXTPTR 函数。
- 定位的 UPDATE 和 DELETE 语句。
- 需要中间临时表的 UPDATE 和 DELETE 语句。
- 向后滚动对远程数据打开的游标 Fetch 语句必须是 NEXT 或 RELATIVE 1。
- 调用包含引用代理表的表达式的函数。
- 如果远程表上的某列的名称是远程服务器上的关键字，则不能访问该列中的数据。您可以执行 CREATE EXISTING TABLE 语句并导入定义，但不能选择该列。

区分大小写

您的 SAP Sybase IQ 数据库的区分大小写设置应该和所访问的任何远程服务器使用的设置匹配。

缺省情况下，创建 SAP Sybase IQ 数据库时不区分大小写。如果使用此配置，当从区分大小写的数据库进行选择时将出现不可预知的结果。根据 ORDER BY 或字符串比较是被传递到远程服务器还是由本地 SAP Sybase IQ 服务器执行，将出现不同的结果。

连接测试

执行以下步骤以确保您可以连接到远程服务器：

- 在配置 SAP Sybase IQ 之前，使用客户端工具（例如 Interactive SQL）确保您可以连接到远程服务器。
- 执行到远程服务器的简单的直通语句以检查您的连接和远程登录配置。例如：

  ```
  FORWARD TO RemoteSA {SELECT @@version};
  ```
- 打开远程跟踪以跟踪与远程服务器的交互。例如：

  ```
  SET OPTION cis_option = 7;
  ```

  一旦开启远程跟踪功能，跟踪信息就会出现在数据库服务器消息窗口中。通过在启动数据库服务器时指定 -o 服务器选项，您可以将此输出记录到文件中。

通过 ODBC 执行的远程数据访问连接

如果您通过 ODBC 访问远程数据库，则会为到远程服务器的连接指定一个名称。可以使用该名称删除连接以取消远程请求。

连接会被命名为 ASACIS_*conn-name*，其中 *conn-name* 是本地连接的连接 ID。该连接 ID 可以从 sa_conn_info 存储过程获得。

Multiplex 服务器的远程数据访问

当访问辅助服务器上的新代理服务器时，计时问题可能会导致服务器紧急关机。

如果发生服务器紧急关机的情况，请重新连接服务器，或等待一段时间后启动一个新事务，然后再尝试使用新的代理表。

附录：SQL 参考

本文档的相应任务中使用的 SQL 语句的参考资料。

ALTER SERVER 语句

修改远程服务器的属性。**ALTER SERVER** 所做的更改会直到下一次与远程服务器连接时才生效。

快速链接：

转至参数 （第 651 页）

转至示例 （第 652 页）

转至用法 （第 653 页）

转至标准 （第 653 页）

转至权限 （第 653 页）

语法

```
ALTER SERVER server-name
    [ CLASS 'server-class' ]
    [ USING 'connection-info' ]
    [ CAPABILITY 'cap-name' { ON | OFF } ]
    [ CONNECTION CLOSE [ CURRENT | ALL | connection-id ] ]

server-class - (back to Syntax)
    { ASAJDBC
        | ASEJDBC
        | SAODBC
        | ASEODBC
        | DB2ODBC
        | MSSODBC
        | ORAODBC
        | ODBC }

connection-info - (back to Syntax)
    { machine-name:port-number [ /dbname ] | data-source-name }
```

参数

(返回顶部) （第 651 页）

- **cap-name** – 服务器功能的名称
- **CLASS** – 更改服务器类。

- **USING** – 如果使用基于 JDBC 的服务器类，则 USING 子句为 *hostname:port-number [/dbname]*，其中：
  - **hostname** – 运行远程服务器的计算机。
  - **portnumber** – 远程服务器监听的 TCP/IP 端口号。SAP Sybase IQ 和 SAP Sybase SQL Anywhere® 的缺省端口号是 2638。
  - **dbname** – 对于 SQL Anywhere 远程服务器，如果未指定 *dbname*，则使用缺省数据库。对于 Adaptive Server，缺省值为 master 数据库，除使用 *dbname* 之外，另一个选择是通过其它某些方法（例如，在 **FORWARD TO** 语句中）指定另一个数据库。

  如果使用基于 ODBC 的服务器类，则 USING 子句为 *data-source-name*，即 ODBC 数据源名称。

- **CAPABILITY** – 将服务器功能设置为 ON 或 OFF。服务器功能存储在系统表 `SYSCAPABILITY` 中。这些功能的名称存储在系统表 `SYSCAPABILITYNAME` 中。只有在首次与远程服务器建立连接后，`SYSCAPABILITY` 表才会包含该服务器的相应条目。首次连接时，SAP Sybase IQ 会询问服务器的功能，然后填充 `SYSCAPABILITY`。对于后面的连接，系统从该表中获取服务器的功能。

  通常情况下，不需要变更服务器的功能。可能需要变更 ODBC 类的通用服务器的功能。

- **CONNECTION CLOSE** – 当用户创建与远程服务器的连接时，此远程连接直到用户与本地数据库断开连接后才关闭。CONNECTION CLOSE 子句允许您显式关闭到远程服务器的连接。当远程连接处于非活动状态或不再需要时，您会发现此功能很有用。

  这些 SQL 语句是等效的，均可关闭到远程服务器的当前连接：

  ```
  ALTER SERVER server-name CONNECTION CLOSE
  ALTER SERVER server-name CONNECTION CLOSE CURRENT
  ```

  可以使用此语法关闭与远程服务器的 ODBC 连接和 JDBC 连接。不需要 SERVER OPERATOR 系统特权便可执行这些语句。

  还可以通过指定连接 ID 来断开特定远程 ODBC 连接，或通过指定 ALL 关键字来断开所有远程 ODBC 连接。如果试图通过指定连接 ID 或 ALL 关键字来关闭 JDBC 连接，则会发生错误。如果由 *connection-id* 标识的连接不是当前本地连接，用户必须拥有 SERVER OPERATOR 系统特权才能关闭连接。

示例

（返回顶部） （第 651 页）

- **示例 1** – 更改名为 `ase_prod` 的 Adaptive Server 服务器的服务器类，使其与 SAP Sybase IQ 的连接基于 ODBC。数据源名称为 `ase_prod`。

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING 'ase_prod'
```

- **示例 2** – 更改服务器 infodc 的功能：

```
ALTER SERVER infodc
CAPABILITY 'insert select' OFF
```

- **示例 3** – 关闭与名为 rem_test 的远程服务器的所有连接：

```
ALTER SERVER rem_test
CONNECTION CLOSE ALL
```

- **示例 4** – 关闭与连接 ID 为 142536、名为 rem_test 的远程服务器的连接：

```
ALTER SERVER rem_test
CONNECTION CLOSE 142536
```

用法

(返回顶部) （第 651 页）

副作用：

- 自动提交

标准

(返回顶部) （第 651 页）

- SQL - ISO/ANSI SQL 语法的服务商扩充。
- SAP Sybase 数据库产品 - 受 Open Client/Open Server 支持。

权限

(返回顶部) （第 651 页）

需要 SERVER OPERATOR 系统特权。

CREATE EXISTING TABLE 语句

创建表示远程服务器上现有表的新代理表。

快速链接：

转至参数 （第 654 页）

转至示例 （第 654 页）

转至用法 （第 655 页）

转至标准 （第 656 页）

转至权限 （第 656 页）

语法

```
CREATE EXISTING TABLE [owner.]table_name
    [ ( column-definition, ⋯ ) ]
    AT 'location-string'

column-definition – (back to Syntax)
    column-name data-type [ NOT NULL ]

location-string – (back to Syntax)
    remote-server-name.[db-name].[owner].object-name | remote-server-
name;[db-name];[owner];object-name
```

参数

(返回顶部) （第 653 页）

- **column-definition** – 如果未指定列定义，SAP Sybase IQ 会根据它从远程表中获得的元数据来派生列的列表。如果指定列定义，SAP Sybase IQ 会对其进行验证。SAP Sybase IQ 检查列名、数据类型、长度和空值属性时：
  - 列名必须完全匹配（但忽略大小写）。
  - **CREATE EXISTING TABLE** 中的数据类型必须匹配或者可转换成远程位置的列的数据类型。例如，本地列的数据类型定义为 NUMERIC，而远程列的数据类型则为 MONEY。如果从数据类型不匹配或存在其它不一致的表中进行选择，可能会遇到错误。
  - 检查每列的 NULL 属性。如果本地列的 NULL 属性与远程列的 NULL 属性不同，则会发出警告消息，但不会中止语句。
  - 检查每列的长度。如果 CHAR、VARCHAR、BINARY、DECIMAL 和 NUMERIC 列的长度不匹配，则会发出警告消息，但不会中止命令。可以选择在 **CREATE EXISTING** 语句中仅包含实际远程列列表的子集。
- **AT** – 指定远程对象的位置。AT 子句支持将分号 (;) 用作分隔符。如果分号出现在位置字符串中的任何位置，则分号将用作字段分隔符。如果没有分号，则使用句号作为字段分隔符。这样一来，便可在数据库和所有者字段中使用文件名和扩展名。分号字段分隔符主要用于当前不支持的服务器类；但是，您也可以在句点也用作字段分隔符的情况下使用它们。

  例如，此语句将表 proxy_a1 映射到远程服务器 myasa 上的 SQL Anywhere 数据库 mydb：

  ```
  CREATE EXISTING TABLE
  proxy_a1
  AT 'myasa;mydb;;a1'
  ```

示例

(返回顶部) （第 653 页）

- **示例 1** – 在远程服务器 server_a 上，为 nation 表创建名为 nation 的代理表：

```
CREATE EXISTING TABLE nation
( n_nationkey int,
  n_name char(25),
  n_regionkey int,
  n_comment char(152))
AT 'server_a.db1.joe.nation'
```

- **示例 2** – 在远程服务器 server_a 上，为 blurbs 表创建名为 blurbs 的代理表。SAP Sybase IQ 将根据它从远程表中获得的元数据来派生列的列表：

```
CREATE EXISTING TABLE blurbs
AT 'server_a.db1.joe.blurbs'
```

- **示例 3** – 在 SAP Sybase IQ 远程服务器 remote_iqdemo_srv 上，为 Employees 表创建名为 rda_employee 的代理表：

```
CREATE EXISTING TABLE rda_employee
AT 'remote_iqdemo_srv..dba.Employees'
```

用法

(返回顶部) (第 653 页)

CREATE EXISTING TABLE 是 **CREATE TABLE** 语句的变体。EXISTING 关键字与 **CREATE TABLE** 一起使用时，指定已存在于远程位置的表，并指定其元数据将导入到 SAP Sybase IQ 中。以这种方式可将远程表建立为用户能够看得见的实体。SAP Sybase IQ 在创建表前会校验它是否存在于外部位置。

作为代理表使用的表，其名称长度不能超过 30 个字符。

如果对象不存在（主机数据文件或远程服务器对象），此语句将被拒绝并伴随出现错误消息。

从主机数据文件或远程服务器表中提取索引信息，并将其用于创建系统表 sysindexes 的行。这定义了服务器术语中的索引和键，并使查询优化程序考虑此表上可能存在的任何索引。

参照约束在适当的时候传递到远程位置。

在 Simplex 环境中，不能在同一节点上创建引用远程表的代理表。在 Multiplex 环境中，不能创建引用在 Multiplex 中定义的远程表的代理表。

例如，在 Simplex 环境中，如果尝试创建对在同一节点上定义的基表 Employees 进行引用的代理表 proxy_e，则系统会拒绝 **CREATE EXISTING TABLE** 语句，并返回错误消息。在 Multiplex 环境中，如果从引用 Multiplex 中定义的远程表 Employees 的任意节点（协调器或辅助节点）创建代理表 proxy_e，则系统会拒绝 **CREATE EXISTING TABLE** 语句。

标准

（返回顶部） （第 653 页）

- SQL - 符合 ISO/ANSI SQL 标准。
- SAP Sybase 数据库产品 - 受 Open Client/Open Server 支持。

权限

（返回顶部） （第 653 页）

如果希望表由自己拥有，则需要下列特权之一：

- CREATE ANY TABLE 系统特权。
- CREATE ANY OBJECT 系统特权。

如果希望表由任何用户拥有，则需要 CREATE ANY TABLE 系统特权。

CREATE SERVER 语句

向 ISYSSERVER 表中添加服务器。

快速链接：

转至参数 （第 657 页）

转至示例 （第 657 页）

转至用法 （第 657 页）

转至标准 （第 658 页）

转至权限 （第 658 页）

语法

```
CREATE SERVER server-name
    CLASS 'server-class'
    USING 'connection-info'
    [ READ ONLY ]

server-class - (back to Syntax)
    { ASAJDBC
        | ASEJDBC
        | SAODBC
        | ASEODBC
        | DB2ODBC
        | MSSODBC
        | ORAODBC
        | ODBC }

connection-info - (back to Syntax)
    { machine-name:port-number [ /dbname ] | data-source-name }
```

参数

(返回顶部) (第 656 页)

- **USING** – 如果使用基于 JDBC 的服务器类，则 USING 子句为 *hostname:port-number [/dbname]*，其中：
  - **hostname** – 运行远程服务器的计算机。
  - **portnumber** – 远程服务器监听的 TCP/IP 端口号。SAP Sybase IQ 和 SAP Sybase SQL Anywhere® 的缺省端口号是 2638。
  - **dbname** – 对于 SQL Anywhere 远程服务器，如果未指定 *dbname*，则使用缺省数据库。对于 Adaptive Server，缺省值为 master 数据库，除使用 *dbname* 之外，另一个选择是通过其它某些方法（例如，在 **FORWARD TO** 语句中）指定另一个数据库。

  如果使用基于 ODBC 的服务器类，则 USING 子句为 *data-source-name*，即 ODBC 数据源名称。
- **READ ONLY** – 指定远程服务器为只读数据源。SAP Sybase IQ 拒绝任何更新请求。

示例

(返回顶部) (第 656 页)

- **示例 1** – 为基于 JDBC 且名为 `ase_prod` 的 Adaptive Server 服务器创建远程服务器。其计算机名称为 "banana"，端口号为 3025。

```
CREATE SERVER ase_prod
CLASS 'asejdbc'
USING 'banana:3025'
```

- **示例 2** – 在计算机 "apple" 上创建名为 `testasa` 的 SQL Anywhere 远程服务器，监听端口号 2638：

```
CREATE SERVER testasa
CLASS 'asajdbc'
USING 'apple:2638'
```

- **示例 3** – 为名为 `oracle723` 的 Oracle 服务器创建远程服务器。它的 ODBC 数据源名称是 "oracle723"：

```
CREATE SERVER oracle723
CLASS 'oraodbc'
USING 'oracle723'
```

用法

(返回顶部) (第 656 页)

CREATE SERVER 通过 SAP Sybase IQ 目录定义远程服务器。

副作用

- 自动提交

标准

(返回顶部) (第 656 页)

- SQL - 符合 ISO/ANSI SQL 标准。
- SAP Sybase 数据库产品 - 受 Open Client/Open Server 支持。

权限

(返回顶部) (第 656 页)

需要 SERVER OPERATOR 系统特权。

CREATE TABLE 语句

在数据库中或远程服务器上创建一个新表。

快速链接:

转至参数 (第 660 页)

转至示例 (第 667 页)

转至用法 (第 670 页)

转至标准 (第 671 页)

转至权限 (第 672 页)

语法

```
CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE
    [ IF NOT EXISTS ] [ owner. ]table-name
    ... ( column-definition [ column-constraint ] ...
    [ , column-definition [ column-constraint ] ...]
    [ , table-constraint ] ... )
    |{ ENABLE | DISABLE } RLV STORE

    ...[ IN dbspace-name ]
    ...[ ON COMMIT { DELETE | PRESERVE } ROWS ]
    [ AT location-string ]
    [PARTITION BY
       range-partitioning-scheme
       | hash-partitioning-scheme
       | composite-partitioning-scheme ]

column-definition - (back to Syntax)
    column-name data-type
      [ [ NOT ] NULL ]
      [ DEFAULT default-value | IDENTITY ]
```

```
    [ PARTITION | SUBPARTITION ( partition-name IN  dbspace-name
[ , ... ] ) ]

default-value – (back to column-definition)
    special-value
    | string
    | global variable
    | [ - ] number
    | ( constant-expression )
    | built-in-function( constant-expression )
    | AUTOINCREMENT
    | CURRENT DATABASE
    | CURRENT REMOTE USER
    | NULL
    | TIMESTAMP
    | LAST USER

special-value – (back to default value)
    CURRENT
    { DATE
    | TIME
    | TIMESTAMP
    | USER
    | PUBLISHER }
    | USER

column-constraint – (back to Syntax)
    [ CONSTRAINT constraint-name ] {
      { UNIQUE
          | PRIMARY KEY
          | REFERENCES table-name [ ( column-name ) ] [ action ]
       }
       [ IN dbspace-name ]
       | CHECK ( condition )
       | IQ UNIQUE ( integer )
     }

table-constraint – (back to Syntax)
    [ CONSTRAINT constraint-name ]
    {   { UNIQUE ( column-name [ , column-name ] ⋯ )
        | PRIMARY KEY ( column-name [ , column-name ] ⋯ )
        }
    [ IN dbspace-name ]
        | foreign-key-constraint
        | CHECK ( condition )
        | IQ UNIQUE ( integer )
    }

foreign-key-constraint – (back to table-constraint)
   FOREIGN KEY [ role-name ] [ ( column-name [ , column-name ] ⋯ ) ]
   ⋯REFERENCES table-name [ ( column-name [ , column-name ] ⋯ ) ]
   ⋯[ actions ] [ IN dbspace-name ]

actions – (back to foreign-key-constraint)
    [ ON { UPDATE | DELETE } RESTRICT ]
```

```
location-string – (back to Syntax) or (back to composite-partitioning-
scheme)
    { remote-server-name. [ db-name ].[ owner ].object-name
        | remote-server-name; [ db-name ]; [ owner ];object-name }

range-partitioning-scheme – (back to Syntax)
    RANGE ( partition-key ) ( range-partition-decl [,range-partition-decl ... ] )

partition-key – (back to range-partitioning-scheme) or (back to hash-
partitioning-scheme)
    column-name

range-partition-decl – (back to range-partitioning-scheme)
    VALUES <= ( {constant-expr
        | MAX } [ , { constant-expr
        | MAX }]... )
    [ IN dbspace-name ]

hash-partitioning-scheme – (back to Syntax) or (back to composite-
partitioning-scheme)
    HASH ( partition-key [ , partition-key, … ] )

composite-partitioning-scheme – (back to Syntax)
    hash-partitioning-scheme SUBPARTITION range-partitioning-scheme
```

参数

(返回顶部) (第 658 页)

- **IN** – 在 column-definition、column-constraint、table-constraint、foreign-key 以及 partition-decl 子句中用于指定将创建对象的 dbspace。如果省略 IN 子句，SAP Sybase IQ 将在已分配表的 dbspace 中创建对象。

  指定 SYSTEM 和该子句，以将永久表或临时表放在目录存储库中。指定 IQ_SYSTEM_TEMP 以将临时用户对象（表、分区或表索引）存储在 IQ_SYSTEM_TEMP 中，或者在 **TEMP_DATA_IN_SHARED_TEMP** 选项设置为 'ON' 且 IQ_SHARED_TEMP dbspace 包含 RW 文件时，存储在 IQ_SHARED_TEMP 中。（无法同时指定 IN 子句和 IQ_SHARED_TEMP。）IN 子句的所有其它用法均被忽略。缺省情况下，所有永久表均放置在主 IQ 存储库中，所有临时表放置在临时 IQ 存储库中。全局临时表和局部临时表决不能出现在 IQ 存储库中。

  不支持下列语法：

  ```
  CREATE LOCAL TEMPORARY TABLE tab1(c1 int) IN IQ_SHARED_TEMP
  ```

  无法将 BIT 数据类型列显式置于 dbspace 中。BIT 数据类型不支持以下内容：

  ```
  CREATE TABLE t1(c1_bit bit IN iq_main);
  ```

- **ON COMMIT** – 仅允许用于临时表。缺省情况下，临时表的行将在 COMMIT（提交）时被删除。

- **AT** – 创建映射到 location-string 子句指定的远程位置的代理表。代理表名不能超过 30 个字符。AT 子句支持分号 (;) 作为分隔符。如果分号出现在 location-string 子句中的任何位置，则分号将用作字段分隔符。如果没有分号，则使用句号作为字段分隔符。这样一来，便可在数据库和所有者字段中使用文件名和扩展名。

  分号字段分隔符主要用于当前不支持的服务器类；但是，您也可以在句点用作字段分隔符的情况下使用它们。例如，此语句将表 proxy_a 映射到远程服务器 myasa 上的 SQL Anywhere 数据库 mydb：

  ```
  CREATE TABLE proxy_a1
  AT 'myasa;mydb;;a1'
  ```

  忽略远程表的外键定义。引用远程表的局部表的外键定义也会被忽略。如果远程服务器支持主键，则主键定义会发送到此服务器。

  在 Simplex 环境中，不能在同一节点上创建引用远程表的代理表。在 Multiplex 环境中，不能创建引用在 Multiplex 中定义的远程表的代理表。

- **IF NOT EXISTS** – 如果指定的对象已存在，则不进行任何更改，也不会返回错误。

- **{ ENABLE | DISABLE } RLV STORE** – 将该表注册到 RLV 存储库以进行实时内存更新。IQ 临时表对此不支持。此值替代数据库选项 **BASE_TABLES_IN_RLV** 的值。将此值设置为 ENABLE 需要 CREATE TABLE 系统特权以及 RLV 存储 dbspace 的 CREATE 权限。

- **column-definition** – 定义表列。所允许的数据类型在《参考：构件块、表和过程》>“SQL 数据类型”中进行了介绍。同一表中的两列不能同名。最多可以创建 45,000 列；但如果一个表中的列多于 10,000，则可能会导致性能下降。

  - **[NOT] NULL]** – 包括或排除 NULL 值。如果指定 NOT NULL，或者列具有 UNIQUE 或 PRIMARY KEY 约束，则该列不能含有任何 NULL 值。每个表中允许 NULL 的列数存在限制，最多为大约 8*(database-page-size - 30)。
  - **DEFAULT default-value** – 通过 CREATE TABLE（和 ALTER TABLE）语句中的 DEFAULT 关键字指定缺省列值。DEFAULT 值用作未指定列值的任何 INSERT（或 LOAD）语句中列的值。
  - **DEFAULT AUTOINCREMENT** – DEFAULT AUTOINCREMENT 列的值唯一地标识表中的每一行。此类型的列也称为 IDENTITY 列，可与 Adaptive Server 兼容。IDENTITY/DEFAULT AUTOINCREMENT 列存储在插入和更新期间自动生成的顺序编号。使用 IDENTITY 或 DEFAULT AUTOINCREMENT 时，列必须是小数位数为 0 的整数数据类型之一或精确的数字类型。列值也可以是 NULL。必须使用所有者名称来限定指定的表名。

    ON 可向表中插入数据。如果没有指定 IDENTITY/DEFAULT AUTOINCREMENT 列的值，则生成一个比列中的任何其它值都大的唯一值。如果 INSERT 指定了列值，则使用该值；如果指定的值不大于列的当前最大值，该值将用作后续插入的起点。

删除行不会递减 IDENTITY/AUTOINCREMENT 计数器的值。由于删除行而产生的间隙只能由使用插入时的显式赋值填充。数据库选项 IDENTITY_INSERT 必须设置为表名，才能向 IDENTITY/AUTOINCREMENT 列执行插入操作。

例如，以下示例创建一个带有 IDENTITY 列的表，并向其显式添加一些数据：

```
CREATE TABLE mytable(c1 INT IDENTITY);
SET TEMPORARY OPTION IDENTITY_INSERT = "DBA".mytable;
INSERT INTO mytable VALUES(5);
```

显式插入小于该列最大值的行号后，后面没有显式赋值的行仍自动递增为比上一个最大值大 1 的值。

通过检查 @@identity 全局变量，可以找到最近一次插入的列值。

- **IDENTITY** – 使用 AUTOINCREMENT 缺省值的替代方法，与 Transact-SQL® 兼容。在 SAP Sybase IQ 中，可使用 IDENTITY 或 DEFAULT AUTOINCREMENT 子句来创建标识列。

- **table-constraint** – 帮助确保数据库中数据的完整性。共有四种类型的完整性约束：
  - **UNIQUE** – 标识唯一识别表中各行的一列或多列。表中任何两行的值在所有指定的列中不能相同。表可以有多个唯一约束。
  - **PRIMARY KEY** – 与 UNIQUE 约束相同，但表只能有一个主键约束。不能为同一列同时指定 PRIMARY KEY 约束和 UNIQUE 约束。主键通常标识行的最佳标识符。例如，客户号可能是 customer 表的主键。
  - **FOREIGN KEY** – 将某组列的值局限于与另一个表的主键或唯一约束中的值匹配。例如，外键约束可用于确保 invoice 表中的客户号与 customer 表中的客户号相对应。

    不能在局部临时表上创建外键约束。全局临时表必须用 ON COMMIT PRESERVE ROWS 创建。
  - **CHECK** – 允许对任意条件进行校验。例如，检查约束可用于确保名为 `Gender` 的列只包含值 male 或 female。表中的任何行都不能违反约束。如果 **INSERT** 或 **UPDATE** 语句会导致行违反约束，则不允许执行相应操作并且撤消语句的作用。

    列检查约束中以符号 '@' 开头的列标识符是实际列名的占位符。以下格式的语句：

    ```
    CREATE TABLE t1(c1 INTEGER CHECK (@foo < 5))
    ```

    与下面的语句完全相同：

    ```
    CREATE TABLE t1(c1 INTEGER CHECK (c1 < 5))
    ```

    表检查约束中以符号 '@' 开头的列标识符不是占位符。

  如果一条语句会引起数据库的更改并且这些更改会使数据库违反完整性约束，则该语句不会得到有效执行，并且系统会报告一条错误。（有效是指系统检测到错误之前该语句所做的任何更改都被撤消。）

  SAP Sybase IQ 通过为该列创建 HG 索引来强制执行单列 UNIQUE 约束。

注意： 不能将含有 BIT 数据类型的列定义为 UNIQUE 或 PRIMARY KEY 约束。另外，BIT 数据类型的列缺省设置为不允许 NULL 值；您可以通过显式定义该列允许 NULL 值来更改这种情况。

- **column-constraint** – 限制列可以容纳的值。列约束和表约束有助于确保数据库中数据的完整性。如果语句会导致违反约束，则该语句的执行不会完成。该语句在检测到错误前所做的任何更改都被撤消并报告错误。列约束是相应表约束的缩写。例如，下列语句是等效的：

```
CREATE TABLE Products (
    product_num integer UNIQUE
)
CREATE TABLE Products (
    product_num integer,
    UNIQUE ( product_num )
)
```

通常使用列约束，除非约束引用了表中的多个列。此类情况下，必须使用表约束。

- **IQ UNIQUE** – 定义列的预期基数并确定是作为 Flat FP 还是 NBit FP 来装载该列。IQ UNIQUE(n) 值显式设置为 0 会作为 Flat FP 装载该列。不带 IQ UNIQUE 约束的列将隐式装载为 NBit，一直到 FP_NBIT_AUTOSIZE_LIMIT、FP_NBIT_LOOKUP_MB 和 FP_NBIT_ROLLOVER_MAX_MB 选项定义的限制：

  - FP_NBIT_AUTOSIZE_LIMIT 限制装载为 NBit 的不同值的数量
  - FP_NBIT_LOOKUP_MB 为 NBit 字典总大小设置阈值
  - FP_NBIT_ROLLOVER_MAX_MB 为从 NBit 至 Flat FP 的隐式 `NBit` 切换设置字典大小
  - FP_NBIT_ENFORCE_LIMITS 强制执行 NBit 字典大小限制。此选项缺省设置为 OFF

  不必使用 n 值小于 FP_NBIT_AUTOSIZE_LIMIT 的 IQ UNIQUE。自动大小调整功能会自动将所有中低基数列的大小调整为 NBit。如果想要将列装载为 Flat FP，或者想要在不同值的数量超出 FP_NBIT_AUTOSIZE_LIMIT 时将列装载为 NBit，请使用 IQ UNIQUE。

注意：

- 当指定高 IQ UNIQUE 值时，请考虑内存的使用情况。如果计算机资源受到限制，应避免带有 FP_NBIT_ENFORCE_LIMITS='OFF'（缺省设置）的装载。在 SAP Sybase IQ 16.0 之前，IQ UNIQUE *n* 值 > 16777216 时会切换到 Flat FP。在 16.0 中，支持对较大的 IQ UNIQUE 值进行标识化，但鉴于基数和列宽，该值可能需要非常多的内存资源。
- BIT、BLOB 和 CLOB 数据类型不支持 NBit 字典压缩。如果 FP_NBIT_IQ15_COMPATIBILITY='OFF'，则在包含这些数据类型的 CREATE TABLE 或 ALTER TABLE 语句中指定的非零 IQ UNIQUE 列将返回一个错误。

- **column-constraint 和 table-constraint 子句** – 列约束和表约束有助于确保数据库中数据的完整性。

  - **PRIMARY KEY 或 PRIMARY KEY (column-name, …)** – 表的主键由列出的列组成，指定的列都不可含有任何 NULL 值。SAP Sybase IQ 确保表中的每一行都有唯一的主键值。表只能有一个 PRIMARY KEY。

    采用第二种格式（PRIMARY KEY 后面跟有列的列表）时，创建的主键所包括的列是按其定义的顺序排序的，而不是按列出的顺序排序。

    将列指定为 PRIMARY KEY、FOREIGN KEY 或 UNIQUE 时，SAP Sybase IQ 会自动为其创建 High_Group 索引。对于多列主键，该索引位于主键上，而不是各个列上。为了获得最佳性能，应单独用 HG 或 LF 索引对各列创建索引。
  - **REFERENCES primary-table-name [(primary-column-name)]** – 将列定义为主键的外键，或主表的唯一约束。通常，外键针对的是主键而不是唯一约束。如果指定主列名，则它必须与主表中的某一列相匹配，而该列受唯一约束或主键约束的制约，而且该约束必须仅包含这一列。否则，外键会引用第二个表的主键。主键与外键必须具有相同的数据类型、精度、小数位数和符号。仅会为单列外键创建非唯一单列 HG 索引。对于多列外键，SAP Sybase IQ 将创建非唯一复合 HG 索引。唯一或非唯一 HG 索引的多列组合键的最大宽度为 1KB。

    临时表不能有引用基表的外键，而基表不能有引用临时表的外键。局部临时表不能有外键，也不能被外键引用。
  - **FOREIGN KEY [role-name] [(...)] REFERENCES primary-table-name [(...)]** – 定义对另一个表中主键或唯一约束的外键引用。通常，外键针对的是主键而不是唯一约束。（此处所述的另一个表称为主表。）

    如果未指定主表列名，主表列即是表的主键中的列。如果未指定外键列名，则外键列的列名与主表的列名相同。如果指定外键列名，则还必须指定主键列名，列名按照列表中的位置成对出现。

    如果主表与外键表不一样，则表明唯一约束或主键约束已在被引用键上定义。被引用键和外键必须具有相同的列数，以及相同的数据类型、符号、精度和小数位数。

    行的外键值必须作为主表中某一行的候选键值出现，除非外键的允许 null 值的列中有一列或多列包含 null 值。

    所有非显式定义的外键列在创建时会被自动赋予与主表中相应列相同的数据类型。这些自动创建的列不能属于外表的主键。因此，必须显式创建同时在主键和外键中使用的列。

    role-name 是外键的名称。*role-name* 的主要作用是区分同一表的两个外键。如果未指定 *role-name*，则按如下方式分配角色名：

    1. 如果没有与表名同名的 *role-name* 外键，则将表名指派为 *role-name*。
    2. 如果表名已被使用，则 *role-name* 为表名加上表的唯一 3 位零填充数字。

参照完整性操作可定义为维护数据库中的外键关系而采取的操作。更改主键值或从数据库表中删除主键值时，其它表中可能有一些相应的外键值需要以某种方式修改。可以指定 ON DELETE 子句，后跟 RESTRICT 子句。

- **RESTRICT** – 如果在数据库的其它地方存在相应的外键时您试图更新或删除主键值，则将生成错误。如果试图更新外键，以便通过候选键创建不匹配的新值，则将生成错误。这是缺省操作，除非您指定 LOAD 可以拒绝违反参照完整性的行。这样便会在语句级别强制实施参照完整性。

  如果使用 CHECK ON COMMIT 时未指定任何操作，那么 RESTRICT 将被视为 DELETE 的一项操作。SAP Sybase IQ 不支持 CHECK ON COMMIT。

  全局临时表不能有引用基表的外键，而基表不能有引用全局临时表的外键。局部临时表不能有外键，也不能被外键引用。
- **CHECK（条件）** – 不允许行违反条件。如果 INSERT 语句会导致行违反条件，则不允许进行该操作并且撤消语句的作用。

  只有条件为 FALSE 时才会拒绝更改；特别是，当条件为 UNKNOWN 时，允许进行更改。SAP Sybase IQ 不会强制遵守 CHECK 条件。

注意： 如果可能，不要在 SAP Sybase IQ 中定义参照完整性外键-主键关系，除非您确定其中没有孤立外键。

- **远程表** – 忽略远程表的外键定义。引用远程表的局部表的外键定义也会被忽略。如果远程服务器支持主键，则主键定义会发送到此服务器。
- **PARTITION BY** – 将大表拆分为多个便于管理的小存储对象。各个分区共用父表的相同逻辑属性，但可放置在单独的 dbspace 中并分别进行管理。SAP Sybase IQ 支持多个表分区模式：
  - 散列分区
  - 范围分区
  - 复合分区

  分区键是包含表分区键的一个或多个列。分区键可包含 NULL 和 DEFAULT 值，但不能包含：
  - LOB（BLOB 或 CLOB）列
  - BINARY 或 VARBINARY 列
  - 长度超过 255 个字节的 CHAR 或 VARCHAR 列
  - BIT 列
  - FLOAT/DOUBLE/REAL 列
- **PARTITION BY RANGE** – 根据分区列中的值范围对行进行分区。范围分区被限定为有 1 个分区键列和最多 1024 个分区。在范围分区模式中，分区键是包含表分区键的列：

```
range-partition-decl:
  partition-name VALUES <= ( {constant-expr |  MAX } [ ,
```

```
{ constant-expr |  MAX }]... )
    [ IN dbspace-name ]
```

分区名称是存储表行的新分区的名称。在表分区集合中，分区名称必须唯一。分区名称是必需项。

- **VALUE** – 按升序为每个分区指定上限（含上限）。用户必须为每个范围分区指定分区条件，确保将每一行只分配到一个分区。分区列允许 NULL 值，将 NULL 作为分区键值的行属于第一个表分区。但 NULL 不能是界限值。

  第一个分区没有下限（MIN 值）。分区键第一列中单元格值为 NULL 的行属于第一个分区。对于最后一个分区，您可以指定一个上限（含上限）或 MAX。如果最后一个分区的上限值不是 MAX，则加载或插入其分区键值大于最后一个分区上限值的任何行都将生成错误。
- **Max** – 表示无限大的上限，只能为最后一个分区指定。
- **IN** – 在 partition-decl 中指定其中应该包含分区行的 dbspace。

以下限制影响范围分区表的分区键和界限值：

- 分区界限必须是常量，而不是常量表达式。
- 分区界限必须按照分区的创建顺序以升序排列。也就是说，第二个分区的上限必须大于第一个分区的上限，依此类推。

  此外，分区界限值必须与相应的分区键列数据类型兼容。例如，VARCHAR 与 CHAR 兼容。
- 如果某个界限值的数据类型不同于其对应分区键列的数据类型，SAP Sybase IQ 会将界限值的数据类型转换为分区键列的数据类型，但存在以下例外情况：
- 不允许进行显式转换。以下示例尝试将 INT 显式转换为 VARCHAR，并生成错误：

  ```
  CREATE TABLE Employees(emp_name VARCHAR(20))
  PARTITION BY RANGE(emp_name)
  (p1 VALUES <=(CAST (1 AS VARCHAR(20))),
  p2 VALUES <= (CAST (10 AS VARCHAR(20)))
  ```
- 不允许执行可导致数据丢失的隐式转换。在以下示例中，分区界限与分区键类型不兼容。舍入假设可导致数据丢失，并生成错误：

  ```
  CREATE TABLE emp_id (id INT) PARTITION BY RANGE(id) (p1 VALUES
  <= (10.5), p2 VALUES <= (100.5))
  ```
- 在以下示例中，分区界限与分区键数据类型兼容。界限值会直接转换为浮点值。无需进行舍入处理，且支持转换：

  ```
  CREATE TABLE id_emp (id FLOAT)
  PARTITION BY RANGE(id) (p1 VALUES <= (10),
  p2 VALUES <= (100))
  ```
- 不允许将非二进制数据类型转换为二进制数据类型。例如，不允许进行以下转换，并会返回错误：

  ```
  CREATE TABLE newemp (name BINARY)
  PARTITION BY RANGE(name)
  (p1 VALUES <= ("Maarten"),
  p2 VALUES <= ("Zymmerman")
  ```

  - 在范围分区表中，不能将 NULL 用作界限。
  - 如果分区键第一列中的单元格值计算结果为 NULL，则对应行将属于第一个分区。SAP Sybase IQ 仅支持一列分区键，因此分区键中的任何 NULL 都会将对应行分配到第一个分区。

- **PARTITION BY HASH** – 基于内部散列函数处理的分区-键值向各个分区映射数据。散列分区键被限定为最多有 8 个列，其声明的列宽总和小于等于 5300 个字节。对于散列分区，表创建者仅确定分区键列；分区的数量和位置将在内部确定。

  在散列分区声明中，分区键是一个或多个列，其合成值将确定存储有各行数据的分区：

```
hash-partitioning-scheme:
  HASH  ( partition-key [ , partition-key, … ] )
```

  - **限制** –

    - 只能对基表进行散列分区。试图对全局临时表或局部临时表进行分区将会引发错误。
    - 不能添加、删除、合并或拆分散列分区。
    - 不能为散列分区键添加列或从中删除列。

- **PARTITION BY HASH RANGE** – 按范围对已进行散列分区的表划分子分区。在散列范围分区模式声明中，SUBPARTITION BY RANGE 子句将向现有散列范围分区表添加一个新的范围子分区：

```
hash-range-partitioning-scheme:
PARTITION BY HASH  ( partition-key [ , partition-key, … ] )
    [ SUBPARTITION BY RANGE  ( range-partition-decl [ , range-
partition-decl ... ] ) ]
```

  散列分区指定如何在逻辑上分布和托管数据；范围子分区指定如何在物理上放置数据。新的范围子分区通过散列进行逻辑分区，其散列分区键与现有的散列范围分区表相同。范围子分区键被限定为仅一列。

  - **限制** –

    - 只能对基表进行散列分区。试图对全局临时表或局部临时表进行分区将会引发错误。
    - 不能添加、删除、合并或拆分散列分区。
    - 不能为散列分区键添加列或从中删除列。

  注意： 与散列范围分区相同，范围分区和复合分区模式要求有单独授权的 VLDB 管理组件。

示例

(返回顶部) (第 658 页)

- **示例 1** – 创建一个名为 `SalesOrders2` 的表，其中包含 5 列。`FinancialCode`、`OrderDate` 和 `ID` 列的数据页位于 dbspace `Dsp3` 中。整数列

CustomerID 的数据页位于 dbspace Dsp1 中。CLOB 列 History 的数据页位于 dbspace Dsp2 中。主键（ID 的 HG）的数据页位于 dbspace Dsp4 中：

```
CREATE TABLE SalesOrders2 (
FinancialCode CHAR(2),
CustomerID int IN Dsp1,
History CLOB IN Dsp2,
OrderDate TIMESTAMP,
ID BIGINT,
PRIMARY KEY(ID) IN Dsp4
) IN Dsp3
```

- **示例 2** – 创建一个名为 fin_code2 的表，其中包含 4 列。code、type 和 id 列的数据页位于数据库选项 DEFAULT_DBSPACE 的值所确定的缺省 dbspace 中。CLOB 列 description 的数据页位于 dbspace Dsp2 中。外键 fk1（c1 的 HG）中的数据页位于 dbspace Dsp4 中：

```
CREATE TABLE fin_code2 (
code INT,
type CHAR(10),
description CLOB IN Dsp2,
id BIGINT,
FOREIGN KEY fk1(id) REFERENCES SalesOrders(ID) IN Dsp4
)
```

- **示例 3** – 创建表 t1，其中分区 p1 与 p2 相邻，分区 p2 与 p3 相邻：

```
CREATE TABLE t1 (c1 INT, c2 INT)
PARTITION BY RANGE(c1)
(p1 VALUES <= (0), p2 VALUES <= (10), p3 VALUES <= (100))
```

- **示例 4** – 创建包含 6 列和 3 个分区的 RANGE 分区表 bar，用于基于日期向各个分区映射数据：

```
CREATE TABLE bar (
        c1 INT IQ UNIQUE(65500),
        c2 VARCHAR(20),
        c3 CLOB PARTITION (P1 IN Dsp11, P2 IN Dsp12,
           P3 IN Dsp13),
        c4 DATE,
        c5 BIGINT,
        c6 VARCHAR(500) PARTITION (P1 IN Dsp21,
           P2 IN Dsp22),
        PRIMARY KEY (c5) IN Dsp2) IN Dsp1
        PARTITION BY RANGE (c4)
        (P1 VALUES <= ('2006/03/31') IN Dsp31,
         P2 VALUES <= ('2006/06/30') IN Dsp32,
         P3 VALUES <= ('2006/09/30') IN Dsp33
        ) ;
```

每个分区的数据页分配：

分区	Dbspace	列
P1	Dsp31	c1、c2、c4、c5

分区	Dbspace	列
P1	Dsp11	c3
P1	Dsp21	c6
P2	Dsp32	c1、c2、c4、c5
P2	Dsp12	c3
P2	Dsp22	c6
P3	Dsp33	c1、c2、c4、c5、c6
P3	Dsp13	c3
P1、P2、P3	Dsp1	c1 的查找存储和其它共享数据
P1、P2、P3	Dsp2	主键（c5 的 HG）

- **示例 5** – 创建包含 PRIMARY KEY（列 c1）和 HASH PARTITION KEY（列 c4 和 c3）的 HASH 分区表 (table tbl42)。

```
CREATE TABLE tbl42 (
    c1 BIGINT NOT NULL,
    c2 CHAR(2) IQ UNIQUE(50),
    c3 DATE IQ UNIQUE(36524),
    c4 VARCHAR(200),
  PRIMARY KEY (c1)
    )
  PARTITION BY HASH ( c4, c3 )
```

- **示例 6** – 创建含有 PRIMARY KEY（列 c1）、散列分区键（列 c4 和 c2）和范围子分区键（列 c3）的散列范围分区表。

```
CREATE TABLE tbl42 (
    c1 BIGINT NOT NULL,
    c2 CHAR(2) IQ UNIQUE(50),
    c3 DATE,
    c4 VARCHAR(200),
  PRIMARY KEY (c1)) IN Dsp1

  PARTITION BY HASH ( c4, c2 )
  SUBPARTITION BY RANGE ( c3 )
  ( P1 VALUES <= (2011/03/31) IN Dsp31,
    P2 VALUES <= (2011/06/30) IN Dsp32,
    P3 VALUES <= (2011/09/30) IN Dsp33) ;
```

- **示例 7** – 为图书馆数据库创建用于保存借出图书信息的表：

```
CREATE TABLE borrowed_book (
date_borrowed DATE NOT NULL,
date_returned DATE,
book       CHAR(20)
```

```
            REFERENCES library_books (isbn),
CHECK( date_returned >= date_borrowed )
)
```

- **示例 8** – 在远程服务器 SERVER_A 上创建表 t1，并创建映射到该远程表的名为 t1 的代理表：

```
CREATE TABLE t1
( a  INT,
  b  CHAR(10))
AT 'SERVER_A.db1.joe.t1'
```

- **示例 9** – 创建表 tab1，其中包含以特殊常量 LAST USER 为缺省值的列 c1：

```
CREATE TABLE tab1(c1 CHAR(20) DEFAULT LAST USER)
```

- **示例 10** – 创建包含列 c1 的局部临时表 tab1：

```
CREATE LOCAL TEMPORARY TABLE tab1(c1 int) IN IQ_SYSTEM_TEMP
```

  在以下情况下，本例将在 IQ_SYSTEM_TEMP dbspace 中创建 tab1：

  - DQP_ENABLED 逻辑服务器策略选项设置为 ON，但在 IQ_SHARED_TEMP 中没有读写文件
  - DQP_ENABLED 选项为 OFF，TEMP_DATA_IN_SHARED_TEMP 逻辑服务器策略选项为 ON，但在 IQ_SHARED_TEMP 中没有读写文件
  - DQP_ENABLED 选项和 TEMP_DATA_IN_SHARED_TEMP 选项均设置为 OFF

  在以下情况下，本例将在 IQ_SHARED_TEMP dbspace 中创建相同的表 tab1：

  - DQP_ENABLED 设置为 ON，且在 IQ_SHARED_TEMP 中具有读写文件
  - DQP_ENABLED 设置为 OFF，TEMP_DATA_IN_SHARED_TEMP 设置为 ON，且在 IQ_SHARED_TEMP 中具有读写文件

- **示例 11** – 创建表 tab1，允许在内存 RLV 存储库中使用行级版本控制和实时存储。

```
CREATE TABLE tab1 ( c1 INT, c2 CHAR(25) ) ENABLE RLV STORE
```

用法

(返回顶部) (第 658 页)

通过指定所有者名称，可为其他用户创建表。如果未指定 GLOBAL TEMPORARY 或 LOCAL TEMPORARY，该表被称为基表。否则，该表为临时表。

与基表一样，所创建的全局临时表也存在于数据库中，并一直保留在数据库中，直到通过 DROP TABLE 语句将其显式删除。临时表中的行仅对插入这些行的连接可见。来自相同或不同应用程序的多个连接可同时使用同一个临时表，但每个连接只能看到它自己的那些行。给定的连接会继承当该连接首次引用全局临时表时该表的模式。连接结束时会删除临时表中的行。

创建局部临时表时，请不要指定所有者。如果在创建临时表时指定所有者（例如，`CREATE TABLE dbo.#temp(col1 int)`），就会将其错误地创建成基表。

如果连接中存在某一局部临时表，则尝试创建同名的基表或全局临时表将失败，因为 `owner.table` 无法唯一地标识新表。

但是，当存在基表或全局临时表时，可创建同名的局部临时表。引用表名时将访问局部临时表，因为将首先解析局部临时表。

例如，请考虑以下序列：

```
CREATE TABLE t1 (c1 int);
INSERT t1 VALUES (9);

CREATE LOCAL TEMPORARY TABLE t1 (c1 int);
INSERT t1 VALUES (8);

SELECT * FROM t1;
```

返回的结果为 8。对 `t1` 的任何引用都将引用局部临时表 `t1`，一直到局部临时表被连接删除。

在过程中，如果要创建一个在过程完成后仍然保留的表，可使用 CREATE LOCAL TEMPORARY TABLE 语句，而不是 DECLARE LOCAL TEMPORARY TABLE 语句。使用 CREATE LOCAL TEMPORARY TABLE 语句创建的局部临时表会一直保留到它被显式删除或连接终止。

利用使用 CREATE LOCAL TEMPORARY TABLE 的 IF 语句创建的局部临时表，在 IF 语句完成后也会继续保留。

SAP Sybase IQ 不支持将 CREATE TABLE ENCRYPTED 子句用于对 SAP Sybase IQ 表进行表级别加密。但 SAP Sybase IQ 数据库中的 SQL Anywhere 表支持 CREATE TABLE ENCRYPTED 子句。

Side Effects

- 自动提交

标准

（返回顶部） （第 658 页）

- SQL - `ISO/ANSI SQL` 语法的服务商扩充。
  以下是服务商扩充：
  - { **IN** | **ON** } *dbspace-name* 子句
  - **ON COMMIT** 子句
  - 部分缺省值
- SAP Sybase 数据库产品 - 受 Adaptive Server 支持，但存在一些差异。
  - **临时表** – 通过在 **CREATE TABLE** 语句中的表名前加上井号 (#)，可创建临时表。这些临时表是 SAP Sybase IQ 声明的临时表，只能在当前连接中可用。有

关声明的临时表的信息，请参见 DECLARE LOCAL TEMPORARY TABLE 语句。

- **实际放置** – SAP Sybase IQ 中表的实际放置方式不同于 Adaptive Server 中表的实际放置方式。Adaptive Server 支持的 **ON** *segment-name* 子句在 SAP Sybase IQ 中也受支持，但 *segment-name* 指的是 IQ dbspace。
- **约束** – SAP Sybase IQ 不支持已命名的约束或已命名的缺省值，但确实支持允许将约束和缺省值定义封装在数据类型定义中的用户定义数据类型。它还支持在 **CREATE TABLE** 语句中使用显式缺省值和 CHECK 条件。
- **NULL** – （缺省值）缺省情况下，Adaptive Server 中的列缺省设置为 NOT NULL，而在 SAP Sybase IQ 中，缺省设置为 NULL，从而允许 NULL 值。可使用 ALLOW_NULLS_BY_DEFAULT 选项控制此设置。请参见 ALLOW_NULLS_BY_DEFAULT 选项 [TSQL]。要使您的数据定义语句成为可移植的语句，应显式指定 NULL 或 NOT NULL。

权限

(返回顶部) （第 658 页）

表类型	所需特权
IQ 主存储库中的基表	自有表 - 针对创建表的 dbspace 需要具备 CREATE 特权。还需要具备以下特权之一： • CREATE TABLE 系统特权。 • CREATE ANY OBJECT 系统特权。 由任何用户拥有的表 - 需要对创建表的 dbspace 具有 CREATE 特权。还需要具备以下特权之一： • CREATE ANY TABLE 系统特权。 • CREATE ANY OBJECT 系统特权。
全局临时表	由自身拥有的表 – 需要具备以下特权之一： • CREATE TABLE 系统特权。 • CREATE ANY OBJECT 系统特权。 由任何用户拥有的表 – 需要具备以下特权之一： • CREATE ANY TABLE 系统特权。 • CREATE ANY OBJECT 系统特权。

表类型	所需特权
代理表	由自身拥有的表 – 需要具备以下特权之一： • CREATE PROXY TABLE 系统特权。 • CREATE ANY TABLE 系统特权。 • CREATE ANY OBJECT 系统特权。 由任何用户拥有的表 – 需要具备以下特权之一： • CREATE ANY TABLE 系统特权。 • CREATE ANY OBJECT 系统特权。

DROP SERVER 语句

从 SAP Sybase IQ 系统表中删除远程服务器。

快速链接：

转至示例 （第 673 页）

转至用法 （第 673 页）

转至标准 （第 674 页）

转至权限 （第 674 页）

语法

```
DROP SERVER server-name
```

示例

(返回顶部) （第 673 页）

- **示例 1** – 本例将删除服务器 `IQ_prod`：

```
DROP SERVER iq_prod
```

用法

(返回顶部) （第 673 页）

必须先删除已经为远程服务器定义的所有代理表，**DROP SERVER** 才会成功。

副作用

- 自动提交

标准

(返回顶部) （第 673 页）

- SQL - 符合 ISO/ANSI SQL 标准。
- SAP Sybase 数据库产品 - 受 Open Client/Open Server 支持。

权限

(返回顶部) （第 673 页）

需要 SERVER OPERATOR 系统特权。

索引

符号

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z