达梦数据库

基本概念

0、数据库实例

所有的操作都基于一个数据库实例之上，创建一个实例会占用一个监听端口，实例名不能重复。

1、用户(user)

用来连接数、访问、管理数据库实例。
用户权利划分：国产软件平台要求三权分立（系统管理员、安全管理员、审计管理员）
达梦用户：管理用户(SYSDBA)、系统用户(SYS)、审计用户(SYSAUDITOR)、安全用户(SYSSSO)。

2、模式(schema)

模式对象是数据库的逻辑结构。
模式与用户的关系：oracle中是一对一的关系，db2一个用户有多个模式，达梦一个用户有多个模式，Mysql中没有模式概念。
模式规则：
- 在同一模式下不能存在同名对象(表)，但在不同模式中的对象名称可以相同。
- 在新建表时，可以指定表空间。如果不指定，默认保存到MAIN.DBF
- 用户可以直接访问其他模式下的对象(表)，前提是有的该对象权限(方式schema.table)

3、表空间

```
#创建实例时默认的几个表空间
系统表空间：SYSTEM.DBF
用户表空间：MAIN.DBF
回滚表空间：ROLL.DBF
临时表空间：TEMP.DBF

#新建表空间
create tablespace "pcsm" datafile 'D:\dmdbms\data\DAMENG\pcsm.dbf' size 32 CACHE
= NORMAL;
```

安装配置

官网下载安装，具体参考官方文档。

[达梦数据库-国产数据库]

常用查询语句

1、查询表空间信息

```
#查所有表空间信息
select * from dba_tablespaces;
select * from v$tablespace;

#查用户级的表空间信息
select * from user_tablespaces;
```

2、查询系统视图

```
select * from SYSTABLECOMMENTS;
```

3、模式切换

```
#查询所有模式
select object_name from all_objects where object_type = 'SCH';

#查看当前模式
select SYS_CONTEXT ('userenv', 'current_schema') from DUAL;
#切换模式
set schema "schema_name";
#非常安全的切换模式
alter session set CURRENT_SCHEMA = "schema_name";
```

4、表查询

```
#查所有模式下的所有表信息
select * from dba_tables;
#查所有模式下表的个数
select count(*) from dba_tables;

#查用户级的所有表信息
select * from user_tables;

#根据指定表空间下的所有表
select * from dba_tables where tablespace_name='xxx';

#查指定模式下的所有表信息
select * from dba_tables where OWNER = 'schema_name';

#根据模式名字，借助公共属性模式ID。UTAB用户表，STAB系统表
SELECT * FROM SYSOBJECTS WHERE schid = (SELECT object_id from all_objects where
object_name = '%s'
     and object_type = 'SCH') and (SUBTYPE$ = 'UTAB' or SUBTYPE$ = 'STAB');
```

4、数据库占用空间

```
#数据库占用空间(单位kb)
select sum(bytes) from dba_data_files;
#数据库读写状态及空间
select * from v$datafile;
#查询某模式下的表占用空间大小(名称大小写敏感)
select TABLE_USED_SPACE('schema_name', 'table_name') * 1024;
```

5、关闭数据库

```
#正常关闭
shutdown normal;

#立即关闭，在执行一些清除工作后关闭（终止会话、释放资源）
shutdown immediate;

#直接关闭，如有操作在执行，重启需要很长时间
shutdown abort;
```

6、其他常用查询语句

```
#查所有用户信息
select * from dba_users;

#查对象信息(模式/表都是对象？？)
select * from all_objects;
select * from dba_objects;
select * from user_objects;

#text数据类型，where查找不能用 '=' 号，要用 link
```

C/C++ 编程(达梦DPI)

官方文档:

达梦程序员手册 -- 第二章 DPI编程指南

```
//SDK在官方各版本安装包中
../source/drivers/dpi

//达梦数据库DPI头文件
#include "DPI.h"
#include "DPIext.h"
#include "DPItypes.h"

//linux版本库
libssl.so、libcrypto.so、libdmdpi.so

//win版本库比较多,也在安装包的../source/drivers/dpi路径下

//常用接口
```

编程示例

```
#include "DPI.h"
#include "DPIext.h"
#include "DPItypes.h"

#ifdef WIN32
```

```
#define CLASS_FUNC  __FUNCTION__
#else
#define CLASS_FUNC  __PRETTY_FUNCTION__
#endif

s8 g_szHost[] = "127.0.0.1";        // Host
s8 g_szUser[] = "SYSDBA";           // 账户
s8 g_szPwd[] = "SYSDBA";            // 密码
udint2 g_nPort = 5236;              // 默认端口
s8 g_szSchema[] = "pcsm";           // 模式
const s32 nDBConnectTimeOut = 300;  // 执行超时时间,单位: ms
const s32 nLoginTimeOut = 300;      // 登录超时时间,单位: ms

#define CREATE_MSPCHN "create table pcsm_chn(chnno varchar(255) primary key, area
varchar(255), devid int, chnid int, chngbno varchar(255), manu varchar(255),
platip int, rsv1 varchar(64), rsv2 varchar(64), rsv3 varchar(64), devip
varchar(255));"
#define UPDATE_MSPCHN "update pcsm_chn set area='%s', devid=%u, chnid=%d,
chngbno='%s', manu='%s',devip='%s' where chnno='%s';"
#define INSERT_MSPCHN "insert into pcsm_chn(chnno, area, devid, chnid, chngbno,
manu, platip,rsv1,rsv2,devip) values('%s','%s',%u,%d,'%s','%s',
%u,'%s','%s','%s');"
#define SELECT_ALL_CHN "select * from pcsm_chn;"

// DM 获取DPI错误信息接口
// 参数:   sdint2 HandleType  - 句柄类型: DSQL_HANDLE_ENV、DSQL_HANDLE_DBC、
DSQL_HANDLE_STMT、DSQL_HANDLE_STMT
//        dhandle dmHandle   - 句柄
// 返回值: s32               - Err code
s32 DmDPIErrMsgPrint(sdint2 HandleType, dhandle pDMHandle)
{
    sdint4 nErrCode = 0;
    sdint2 sMsgLen = 0;
    sdbyte achErrMsg[SDBYTE_MAX];

    // 获取错误信息集合
    dpi_get_diag_rec(HandleType, pDMHandle, 1, &nErrCode, achErrMsg,
sizeof(achErrMsg), &sMsgLen);

    // 打印具体错误信息
    printf("err_msg = %s, err_code = %d\n", achErrMsg, nErrCode);

    return nErrCode;
}

// 判断达梦数据库连接状态
bool DmPing(dhandle pDMConHandle)
{
    s32 nDMRet = 0;
    s32 nOutBufLen = 0;
    s8 achSchemaName[1024] = { 0 };

    // 尝试获取当前模式,以此来判断连接状态
    nDMRet = dpi_get_con_attr(pDMConHandle, DSQL_ATTR_CURRENT_SCHEMA,
achSchemaName, sizeof(achSchemaName), &nOutBufLen);
    if (!DSQL_SUCCEEDED(nDMRet))
    {
        DmDPIErrMsgPrint(DSQL_HANDLE_DBC, pDMConHandle);
```

```
        return false;
    }

    return true;
}

// 达梦直接执行sql语句接口
s32 DmExecDirect(PDBhandle pDBHandle, const s8* pchSql, BOOL32 bPrintf/* =
TRUE*/)
{
    s32 nRet = DSQL_ERROR;

    // 打印执行的语句
    if (bPrintf == TRUE && pchSql != NULL)
    {
        printf("[%s] SQL:\n %s\n", CLASS_FUNC, pchSql);
    }

    // 申请DPI语句句柄
    dhstmt pHStmt;
    nRet = dpi_alloc_stmt((dhcon)pDBHandle, &pHStmt);
    if (!DSQL_SUCCEEDED(nRet))
    {
        DmDPIErrMsgPrint(DSQL_HANDLE_STMT, pHStmt);
        return nRet;
    }

    // 执行 sql
    nRet = dpi_exec_direct(pHStmt, (sdbyte *)pchSql);
    if (!DSQL_SUCCEEDED(nRet))
    {
        DmDPIErrMsgPrint(DSQL_HANDLE_STMT, pHStmt);
    }

    // 释放DPI语句句柄
    dpi_free_stmt(pHStmt);

    return nRet;
}

// 打印查询MSPCHN表结果
void DMFetchValue_MSPCHN(const dhstmt pStmtHandle)
{
    s32 nRet = 0;

    // 绑定列数据输出地址
    // 表字段: create table pcsm_chn(chnno varchar(255) primary key,area
varchar(255),
    // devid int,chnid int,chngbno varchar(255),manu varchar(255),platip int,
rsv1 varchar(64),
    // rsv2 varchar(64), rsv3 varchar(64), devip varchar(255));

    s32 nOutDataLen = 0;
    s8 achChnNo[255] = { 0 };
    s8 achArea[255] = { 0 };
    s32 nDevID = 0;
    s32 nChnID = 0;
    s8 achChnGbNo[255] = { 0 };
```

```
    s8 achManu[255] = { 0 };
    s32 nPlatIP = 0;
    s8 achRsv1[64] = { 0 };
    s8 achRsv2[64] = { 0 };
    s8 achRsv3[64] = { 0 };
    s8 achDevIP[255] = { 0 };

    nRet = dpi_bind_col(pStmtHandle, 1, DSQL_C_NCHAR, &achChnNo,
sizeof(achChnNo), &nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 2, DSQL_C_NCHAR, &achArea, sizeof(achArea),
&nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 3, DSQL_C_SLONG, &nDevID, sizeof(nDevID),
&nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 4, DSQL_C_SLONG, &nChnID, sizeof(nChnID),
&nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 5, DSQL_C_NCHAR, &achChnGbNo,
sizeof(achChnGbNo), &nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 6, DSQL_C_NCHAR, &achManu, sizeof(achManu),
&nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 7, DSQL_C_SLONG, &nPlatIP, sizeof(nPlatIP),
&nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 8, DSQL_C_NCHAR, &achRsv1, sizeof(achRsv1),
&nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 9, DSQL_C_NCHAR, &achRsv2, sizeof(achRsv2),
&nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 10, DSQL_C_NCHAR, &achRsv3,
sizeof(achRsv3), &nOutDataLen);
    nRet = dpi_bind_col(pStmtHandle, 11, DSQL_C_NCHAR, &achDevIP,
sizeof(achDevIP), &nOutDataLen);


    // 遍历行
    s32 nRowNum = 0;
    nRet = dpi_fetch(pStmtHandle, (ulength *)&nRowNum);
    while (nRet != DSQL_NO_DATA && nRet != DSQL_ERROR && nRet !=
DSQL_INVALID_HANDLE)
    {
        printf("nRowNum[%d], achChnNo: %s, achArea: %s, nDevID: %d, nChnID: %d,
achChnGbNo: %s, achManu: %s, nPlatIP: %d, achRsv1: %s, achRsv2: %s, achRsv3: %s,
achDevIP: %s\n",
            nRowNum, achChnNo, achArea, nDevID, nChnID, achChnGbNo, achManu,
nPlatIP, achRsv1, achRsv2, achRsv3, achDevIP);

        // 取下一行数据
        nRet = dpi_fetch(pStmtHandle, (ulength *)&nRowNum);
    }
}

int main(int argc, char* argv[])
{
    DPIRETURN dmRet = DSQL_SUCCESS;
    dhenv pDMHenv = NULL; //达梦环境句柄
    dhcon pDMHcon = NULL; //连接句柄
    dhstmt pHStmt = NULL; //语句句柄
    s8 pchSql[1024] = { 0 };

    //Step -1 初始化
    dmRet = dpi_module_init();
```

```
    if (!DSQL_SUCCEEDED(dmRet))
    {
        printf("[%s] dpi_module_init failed!", CLASS_FUNC);
        return 0;
    }

    do
    {
        // 达梦DPI创建环境句柄(一个环境句柄可以包含多个连接句柄)
        dmRet = dpi_alloc_env(&pDMHenv);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_ENV, pDMHenv);
            break;
        }
        // 设置环境参数: 本地编码格式、语言
        dmRet = dpi_set_env_attr(pDMHenv, DSQL_ATTR_LOCAL_CODE,
(dpointer)PG_UTF8, sizeof(PG_UTF8));
        dmRet = dpi_set_env_attr(pDMHenv, DSQL_ATTR_LANG_ID,
(dpointer)LANGUAGE_EN, sizeof(LANGUAGE_EN));



        // 申请连接句柄
        dmRet = dpi_alloc_con(pDMHenv, &pDMHcon);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_DBC, pDMHcon);
            break;
        }
        // 设置连接属性
        dmRet = dpi_set_con_attr(pDMHcon, DSQL_ATTR_CONNECTION_TIMEOUT,
(dpointer)nDBConnectTimeOut, sizeof(sdint4));//执行超时时间
        dmRet = dpi_set_con_attr(pDMHcon, DSQL_ATTR_LOGIN_TIMEOUT,
(dpointer)nLoginTimeOut, sizeof(sdint4));  // 登录超时时间
        dmRet = dpi_set_con_attr(pDMHcon, DSQL_ATTR_LOGIN_PORT,
(dpointer)g_nPort, sizeof(udint2));            // 服务端口(默认5236)
        dmRet = dpi_set_con_attr(pDMHcon, DSQL_ATTR_AUTOCOMMIT,
(dpointer)DSQL_AUTOCOMMIT_OFF, sizeof(DSQL_AUTOCOMMIT_ON)); // 关闭自动提交事务设置

        // 连接数据库服务器
        dmRet = dpi_login(pDMHcon, (sdbyte *)g_szHost, (sdbyte *)g_szUser,
(sdbyte *)g_szPwd);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_DBC, pDMHcon);
            break;
        }


         // 申请语句句柄
        dmRet = dpi_alloc_stmt((dhcon)pDMHcon, &pHStmt);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_STMT, pHStmt);
            return false;
        }
```

```
        // 创建模式
        sprintf(pchSql, "create schema %s authorization %s;", g_szSchema,
g_szUssr);
        dmRet = dpi_exec_direct(pHStmt, (sdbyte *)pchSql);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_STMT, pHStmt);
            break;
        }

        // 切换模式
        sprintf(pchSql, "set schema %s;", m_tDBInfo.m_szDBName);
        dmRet = dpi_exec_direct(pHStmt, (sdbyte *)pchSql);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_STMT, pHStmt);
            break;
        }

        // 提交事务
        dmRet = dpi_commit(pDMHcon);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_DBC, (dhcon)pDMHcon);
            break;
        }


        // 建表MSP_CHN
        sprintf(pchSql, CREATE_MSPCHN);
        dmRet = dpi_exec_direct(pHStmt, (sdbyte *)pchSql);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_STMT, pHStmt);
            break;
        }

        // 插入数据

        // 更新数据

        // 查询所有数据结果接
        sprintf(pchSql, SELECT_ALL_CHN);
        dmRet = dpi_exec_direct(pHStmt, (sdbyte *)pchSql);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_STMT, pHStmt);
            break;
        }
        DMFetchValue_MSPCHN(pHStmt);


         // 提交事务
        dmRet = dpi_commit(pDMHcon);
        if (!DSQL_SUCCEEDED(dmRet))
        {
            DmDPIErrMsgPrint(DSQL_HANDLE_DBC, (dhcon)pDMHcon);
            break;
```

```
        }
    } while (0);



    if(pHStmt)
    {
        dpi_free_stmt(pHStmt);
        pHStmt = NULL;
    }
    if(pDMHcon)
    {
        dpi_logout(pDMHcon);
        dpi_free_con(pDMHcon);
        pDMHcon = NULL;
    }
    if(pDMHenv)
    {
        dpi_free_env(pDMHenv);
        pDMHenv = NULL;
    }

    //逆初始化
    dmRet = dpi_module_deinit();
    if (!DSQL_SUCCEEDED(dmRet))
    {
        printf("[%s] dpi_module_deinit failed!", CLASS_FUNC);
    }

    return 0;
}
```