CLI/ODBC Examples

Application Development

Check CLI Installation

If the following files are included in include and lib of the directory where Machbase is installed, the environment in which the application can be developed is complete.

```
Mach@localhost:~/machbase_home$ ls -l include lib install/
include:
total 176
-rwxrwxr-x 1 mach mach 31449 Jun 18 19:26 machbase_sqlcli.h

install/:
total 12
-rw-rw-r-- 1 mach mach 1667 Jun 18 19:26 machbase_env.mk

lib:
total 16196
-rw-rw-r-- 1 mach mach  78603 Jun 18 19:26 machbase.jar
-rw-rw-r-- 1 mach mach 964290 Jun 18 19:26 libmachbasecli.a
```

Exam

ple

- SQLDescribe
- SQLColumns

Makefile Creation Guide

```
mach@localhost:~/machbase_home$ cd sample/
mach@localhost:~/machbase_home/sample$ cd cli/
mach@localhost:~/machbase_home/sample/cli$ ls
Makefile sample1_connect.c
```

If the Machbase package was installed, the sample program will be installed in the following path.

```
include $(MACHBASE_HOME)/install/machbase_env.mk
INCLUDES += $(LIBDIR_OPT)/$(MACHBASE_HOME)/include

all : sample1_connect

sample1_connect : sample1_connect.o
$(LD_CC) $(LD_FLAGS) $(LD_OUT_OPT)$@ $< $(LIB_OPT)machbasecli$(LIB_AFT) $(LIBDIR_OPT)$(MACHBASE_HOME)/lib
$(LD_LIBS)

sample1_connect.o : sample1_connect.c
$(COMPILE.cc) $(CC_FLAGS) $(INCLUDES) $(CC_OUT_OPT)$@ $<

clean :
rm -f sample1_connect
```

Compile and Link

Executing the following for a given sample creates an executable file.

```
mach@localhost:~/machbase_home/sample/cli$ make
gcc -c -g -W -Wall -rdynamic -O3 -finline-functions -fno-omit-frame-pointer -fno-strict-aliasing -m64 -
mtune=k8 -g -W -Wall -rdynamic -O3 -finline-functions -fno-omit-frame-pointer -fno-strict-aliasing -m64 -
mtune=k8 -I/home/machbase/machbase_home/include -I. -L//home/machbase/machbase_home/include -
osample1_connect.o sample1_connect.c
gcc -m64 -mtune=k8 -L/home/machbase/machbase_home/lib -osample1_connect sample1_connect.o -lmachbasecli -L
/home/machbase/machbase_home/lib -lm -lpthread -ldl -lrt -rdynamic
mach@localhost:~/machbase_home/sample/cli$ ls -al
total 1196
drwxrwxr-x 2 mach mach 4096 Jun 18 20:15 .
drwxrwxr-x 4 mach mach 4096 Jun 18 19:26 ..
-rw-rw-r-- 1 mach mach 483 Jun 18 19:26 Makefile
-rwxrwxr-x 1 mach mach 1196943 Jun 18 20:15 sample1_connect
-rw-rw-r-- 1 mach mach 549 Jun 18 19:26 sample1_connect.c
-rw-rw-r-- 1 mach mach 8168 Jun 18 20:15 sample1_connect.o
```

> You can write your application as necessary by modifying the sample Makefile above.

Sample Program

Connection Example

We will create an example program to connect using the CLI.
The sample file name is sample1_connect.c.
MACHBASE_PORT_NO must be the same as the PORT_NO value in the $MACHBASE_HOME/conf/machbase.conf file.

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

#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;

void connectDB()
{
    char connStr[1024];
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }
    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }
    sprintf(connStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);
    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)connStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");
        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &errNo,
                                      errMsg, 1024, &msgLength ))
        {
            printf("mach-%d : %s\n", errNo, errMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }
    printf("connected ... \n");
}

void disconnectDB()
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLDisconnect(gCon))
    {
        printf("disconnect error\n");

        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &errNo,
                                     errMsg, 1024, &msgLength ))
        {
            printf("mach-%d : %s\n", errNo, errMsg);
        }
    }

    SQLFreeConnect(gCon);
```

```
    SQLFreeEnv(gEnv);
}

int main()
{
    connectDB();
    disconnectDB();
    return 0;
}
```

If you register sample1_connect.c in Makefile, compile and run it, it will appear as follows.

```
[mach@localhost cli]$ make

[mach@localhost cli]$ ./sample1_connect
connected ...
```

Data Input and Output Example

In the example source below, we created a table using the CREATE TABLE statement, arbitrarily create simple data values, input data using the INSERT statement, and output the data using the SELECT statement. You will be able to see how to configure each type when entering and checking values directly.
The sample file name is sample2_insert.c.

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

#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB()
{
    char connStr[1024];
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];
    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }
    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }
    sprintf(connStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);
    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)connStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");
        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &errNo,
```

```
                                    errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }
    printf("connected ... \n");
}

void disconnectDB()
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];
    if (SQL_ERROR == SQLDisconnect(gCon)) {
        printf("disconnect error\n");
        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &errNo,
                                    errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
    }
    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg, SQLHSTMT stmt)
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];
    printf("ERROR : (%s)\n", aMsg);
    if (SQL_SUCCESS == SQLError( gEnv, gCon, stmt, NULL, &errNo,
                                errMsg, 1024, &msgLength ))
    {
        printf(" mach-%d : %s\n", errNo, errMsg);
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT stmt;
    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error", stmt);
    }
    if (SQLExecDirect(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error", stmt);
    }
    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error", stmt);
    }
}

void prepareExecuteSQL(const char *aSQL)
{
    SQLHSTMT stmt;
    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR)
    {
        outError("AllocStmt error", stmt);
    }
    if (SQLPrepare(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
```

```
        printf("Prepare error[%s]\n", aSQL);
        outError("Prepare error", stmt);
    }
    if (SQLExecute(stmt) == SQL_ERROR)
    {
        outError("prepared execute error", stmt);
    }
    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        outError("FreeStmt Error", stmt);
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE1", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE1(seq short, score integer, total long, percentage float,
ratio double, id varchar(10), srcip ipv4, dstip ipv6, reg_date datetime, textlog text, image binary)",
0);
}

void selectTable()
{
    SQLHSTMT stmt;
    const char *aSQL = "SELECT seq, score, total, percentage, ratio, id, srcip, dstip, reg_date,
textlog, image FROM CLI_SAMPLE1";
    int i=0;
    SQLLEN Len = 0;
    short seq;
    int score;
    long total;
    float percentage;
    double ratio;
    char id [11];
    char srcip[16];
    char dstip[40];
    SQL_TIMESTAMP_STRUCT regdate;
    char log [1024];
    char image[1024];
    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR) {
        outError("AllocStmt Error", stmt);
    }
    if (SQLPrepare(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR) {
        printf("Prepare error[%s] \n", aSQL);
        outError("Prepare error", stmt);
    }
    if (SQLExecute(stmt) == SQL_ERROR) {
        outError("prepared execute error", stmt);
    }
    SQLBindCol(stmt, 1, SQL_C_SHORT, &seq, 0, &Len);
    SQLBindCol(stmt, 2, SQL_C_LONG, &score, 0, &Len);
    SQLBindCol(stmt, 3, SQL_C_BIGINT, &total, 0, &Len);
    SQLBindCol(stmt, 4, SQL_C_FLOAT, &percentage, 0, &Len);
    SQLBindCol(stmt, 5, SQL_C_DOUBLE, &ratio, 0, &Len);
    SQLBindCol(stmt, 6, SQL_C_CHAR, id, sizeof(id), &Len);
    SQLBindCol(stmt, 7, SQL_C_CHAR, srcip, sizeof(srcip), &Len);
    SQLBindCol(stmt, 8, SQL_C_CHAR, dstip, sizeof(dstip), &Len);
    SQLBindCol(stmt, 9, SQL_C_TYPE_TIMESTAMP, &regdate, 0, &Len);
    SQLBindCol(stmt, 10, SQL_C_CHAR, log, sizeof(log), &Len);
    SQLBindCol(stmt, 11, SQL_C_CHAR, image, sizeof(image), &Len);
    while (SQLFetch(stmt) == SQL_SUCCESS)
    {
        printf("===== %d ========\n", i++);
        printf("seq = %d", seq);
        printf(", score = %d", score);
        printf(", total = %ld", total);
        printf(", percentage = %.2f", percentage);
        printf(", ratio = %g", ratio);
        printf(", id = %s", id);
        printf(", srcip = %s", srcip);
        printf(", dstip = %s", dstip);
```

```
        printf(", regdate = %d-%02d-%02d %02d:%02d:%02d",
               regdate.year, regdate.month, regdate.day,
               regdate.hour, regdate.minute, regdate.second);
        printf(", log = %s", log);
        printf(", image = %s\n", image);
    }
    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        outError("FreeStmt eror", stmt);
    }
}

void directInsert()
{
    int i;
    char query[2 * 1024];
    short seq;
    int score;
    long total;
    float percentage;
    double ratio;
    char id [11];
    char srcip [16];
    char dstip [40];
    char reg_date [40];
    char log [1024];
    char image [1024];
    for(i=1; i<10; i++)
    {
        seq = i;
        score = i+i;
        total = (seq + score) * 10000;
        percentage = (float)score/total;
        ratio = (double)seq/total;
        sprintf(id, "id-%d", i);
        sprintf(srcip, "192.168.0.%d", i);
        sprintf(dstip, "2001:0DB8:0000:0000:0000:0000:1428:%04d", i);
        sprintf(reg_date, "2015-03-31 15:26:%02d", i);
        sprintf(log, "text log-%d", i);
        sprintf(image, "binary image-%d", i);
        memset(query, 0x00, sizeof(query));
        sprintf(query, "INSERT INTO CLI_SAMPLE1 VALUES(%d, %d, %ld, %f, %f, '%s', '%s', '%s',TO_DATE('%
s','YYYY-MM-DD HH24:MI:SS'),'%s','%s')",
                seq, score, total, percentage, ratio, id, srcip, dstip, reg_date, log, image);
        prepareExecuteSQL(query);
        printf("%d record inserted\n", i);
    }
}

int main()
{
    connectDB();
    createTable();
    directInsert();
    selectTable();
    disconnectDB();
    return 0;
}
```

If you register sample2_insert.c in the Makefile, compile and run it, it will appear as follows.

```
[mach@localhost cli]$ make

[mach@localhost cli]$ ./sample2_insert

connected ...
1 record inserted
2 record inserted
3 record inserted
4 record inserted
5 record inserted
6 record inserted
7 record inserted
8 record inserted
9 record inserted
===== 0 ========
seq = 9, score = 18, total = 270000, percentage = 0.00, ratio = 3.3e-05, id = id-9, srcip = 192.168.0.9,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0009, regdate = 2015-03-31 15:26:09, log = text log-9, image =
62696E61727920696D6167652D39
===== 1 ========
seq = 8, score = 16, total = 240000, percentage = 0.00, ratio = 3.3e-05, id = id-8, srcip = 192.168.0.8,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0008, regdate = 2015-03-31 15:26:08, log = text log-8, image =
62696E61727920696D6167652D38
===== 2 ========
seq = 7, score = 14, total = 210000, percentage = 0.00, ratio = 3.3e-05, id = id-7, srcip = 192.168.0.7,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0007, regdate = 2015-03-31 15:26:07, log = text log-7, image =
62696E61727920696D6167652D37
===== 3 ========
seq = 6, score = 12, total = 180000, percentage = 0.00, ratio = 3.3e-05, id = id-6, srcip = 192.168.0.6,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0006, regdate = 2015-03-31 15:26:06, log = text log-6, image =
62696E61727920696D6167652D36
===== 4 ========
seq = 5, score = 10, total = 150000, percentage = 0.00, ratio = 3.3e-05, id = id-5, srcip = 192.168.0.5,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0005, regdate = 2015-03-31 15:26:05, log = text log-5, image =
62696E61727920696D6167652D35
===== 5 ========
seq = 4, score = 8, total = 120000, percentage = 0.00, ratio = 3.3e-05, id = id-4, srcip = 192.168.0.4,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0004, regdate = 2015-03-31 15:26:04, log = text log-4, image =
62696E61727920696D6167652D34
===== 6 ========
seq = 3, score = 6, total = 90000, percentage = 0.00, ratio = 3.3e-05, id = id-3, srcip = 192.168.0.3, dstip
= 2001:0DB8:0000:0000:0000:0000:1428:0003, regdate = 2015-03-31 15:26:03, log = text log-3, image =
62696E61727920696D6167652D33
===== 7 ========
seq = 2, score = 4, total = 60000, percentage = 0.00, ratio = 3.3e-05, id = id-2, srcip = 192.168.0.2, dstip
= 2001:0DB8:0000:0000:0000:0000:1428:0002, regdate = 2015-03-31 15:26:02, log = text log-2, image =
62696E61727920696D6167652D32
===== 8 ========
seq = 1, score = 2, total = 30000, percentage = 0.00, ratio = 3.3e-05, id = id-1, srcip = 192.168.0.1, dstip
= 2001:0DB8:0000:0000:0000:0000:1428:0001, regdate = 2015-03-31 15:26:01, log = text log-1, image =
62696E61727920696D6167652D31
```

Prepare Execute Example

Let's write an example program that binds and INSERTs data.
You can enter a value by binding data in Machbase. When you use this, you need to specify the types of data values clearly. In case of long string types, you must specify the length value.
The following example shows how to bind data for each type.
The file name is sample3_prepare.c.

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

```
#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB()
{
    char sConnStr[1024];

    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }

    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }

    sprintf(sConnStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)sConnStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");

        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &sErrorNo,
                                      sErrorMsg, 1024, &sMsgLength ))
        {
            printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }

    printf("connected ... \n");

}

void disconnectDB()
{
    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        printf("disconnect error\n");

        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &sErrorNo,
                                     sErrorMsg, 1024, &sMsgLength ))
        {
            printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
        }
    }

    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg, SQLHSTMT aStmt)
```

```
{
    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    printf("ERROR : (%s)\n", aMsg);

    if (SQL_SUCCESS == SQLError( gEnv, gCon, aStmt, NULL, &sErrorNo,
                                 sErrorMsg, 1024, &sMsgLength ))
    {
        printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT sStmt;

    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error", sStmt);
    }

    if (SQLExecDirect(sStmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error", sStmt);
    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error", sStmt);
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(seq short, score integer, total long, percentage float,
ratio double, id varchar(10), srcip ipv4, dstip ipv6, reg_date datetime, tlog text, image binary)", 0);
}

void selectTable()
{
    SQLHSTMT sStmt;
    const char *aSQL = "SELECT seq, score, total, percentage, ratio, id, srcip, dstip, reg_date, tlog,
image FROM CLI_SAMPLE";

    int i=0;
    short sSeq;
    int sScore;
    long sTotal;
    float sPercentage;
    double sRatio;
    char sId [20];
    char sSrcIp[20];
    char sDstIp[50];
    SQL_TIMESTAMP_STRUCT sRegDate;
    char sLog [1024];
    char sImage[1024];
    SQL_LEN sLen;

    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR) {
        outError("AllocStmt Error", sStmt);
    }

    if (SQLPrepare(sStmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR) {
```

```
        printf("Prepare error[%s] \n", aSQL);
        outError("Prepare error", sStmt);
    }

    if (SQLExecute(sStmt) == SQL_ERROR) {
        outError("prepared execute error", sStmt);
    }

    SQLBindCol(sStmt, 1, SQL_C_SSHORT, &sSeq, 0, &sLen);
    SQLBindCol(sStmt, 2, SQL_C_SLONG, &sScore, 0, &sLen);
    SQLBindCol(sStmt, 3, SQL_C_SBIGINT, &sTotal, 0, &sLen);
    SQLBindCol(sStmt, 4, SQL_C_FLOAT, &sPercentage, 0, &sLen);
    SQLBindCol(sStmt, 5, SQL_C_DOUBLE, &sRatio, 0, &sLen);
    SQLBindCol(sStmt, 6, SQL_C_CHAR, sId, sizeof(sId), &sLen);
    SQLBindCol(sStmt, 7, SQL_C_CHAR, sSrcIp, sizeof(sSrcIp), &sLen);
    SQLBindCol(sStmt, 8, SQL_C_CHAR, sDstIp, sizeof(sDstIp), &sLen);
    SQLBindCol(sStmt, 9, SQL_C_TYPE_TIMESTAMP, &sRegDate, 0, &sLen);
    SQLBindCol(sStmt, 10, SQL_C_CHAR, sLog, sizeof(sLog), &sLen);
    SQLBindCol(sStmt, 11, SQL_C_CHAR, sImage, sizeof(sImage), &sLen);

    while (SQLFetch(sStmt) == SQL_SUCCESS)
    {
        printf("===== %d ========\n", i++);
        printf("seq = %d", sSeq);
        printf(", score = %d", sScore);
        printf(", total = %ld", sTotal);
        printf(", percentage = %.2f", sPercentage);
        printf(", ratio = %g", sRatio);
        printf(", id = %s", sId);
        printf(", srcip = %s", sSrcIp);
        printf(", dstip = %s", sDstIp);
        printf(", regdate = %d-%02d-%02d %02d:%02d:%02d",
               sRegDate.year, sRegDate.month, sRegDate.day,
               sRegDate.hour, sRegDate.minute, sRegDate.second);
        printf(", log = %s", sLog);
        printf(", image = %s\n", sImage);
    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP))
    {
        outError("FreeStmt eror", sStmt);
    }
}

void prepareInsert()
{
    SQLHSTMT sStmt;
    int i;
    short sSeq;
    int sScore;
    long sTotal;
    float sPercentage;
    double sRatio;
    char sId [20];
    char sSrcIp [20];
    char sDstIp [50];
    long reg_date;
    char sLog [100];
    char sImage [100];
    int sLength[5];

    const char *sSQL = "INSERT INTO CLI_SAMPLE VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ? )";

    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR)
    {
        outError("AllocStmt error", sStmt);
    }

    if (SQLPrepare(sStmt, (SQLCHAR *)sSQL, SQL_NTS) == SQL_ERROR)
    {
        printf("Prepare error[%s]\n", sSQL);
```

```
    outError("Prepare error", sStmt);
}

for(i=1; i<10; i++)
{
    sSeq = i;
    sScore = i+i;
    sTotal = (sSeq + sScore) * 10000;
    sPercentage = (float)(sScore+2)/sScore;
    sRatio = (double)(sSeq+1)/sTotal;
    sprintf(sId, "id-%d", i);
    sprintf(sSrcIp, "192.168.0.%d", i);
    sprintf(sDstIp, "2001:0DB8:0000:0000:0000:0000:1428:%04x", i);
    reg_date = i*10000;
    sprintf(sLog, "log-%d", i);
    sprintf(sImage, "image-%d", i);

    if (SQLBindParameter(sStmt,
                         1,
                         SQL_PARAM_INPUT,
                         SQL_C_SSHORT,
                         SQL_SMALLINT,
                         0,
                         0,
                         &sSeq,
                         0,
                         NULL) == SQL_ERROR)
    {
        outError("BindParameter error 1", sStmt);
    }

    if (SQLBindParameter(sStmt,
                         2,
                         SQL_PARAM_INPUT,
                         SQL_C_SLONG,
                         SQL_INTEGER,
                         0,
                         0,
                         &sScore,
                         0,
                         NULL) == SQL_ERROR)
    {
        outError("BindParameter error 2", sStmt);
    }

    if (SQLBindParameter(sStmt,
                         3,
                         SQL_PARAM_INPUT,
                         SQL_C_SBIGINT,
                         SQL_BIGINT,
                         0,
                         0,
                         &sTotal,
                         0,
                         NULL) == SQL_ERROR)
    {
        outError("BindParameter error 3", sStmt);
    }

    if (SQLBindParameter(sStmt,
                         4,
                         SQL_PARAM_INPUT,
                         SQL_C_FLOAT,
                         SQL_FLOAT,
                         0,
                         0,
                         &sPercentage,
                         0,
                         NULL) == SQL_ERROR)
    {
        outError("BindParameter error 4", sStmt);
```

```
    }

    if (SQLBindParameter(sStmt,
                         5,
                         SQL_PARAM_INPUT,
                         SQL_C_DOUBLE,
                         SQL_DOUBLE,
                         0,
                         0,
                         &sRatio,
                         0,
                         NULL) == SQL_ERROR)
    {
        outError("BindParameter error 5", sStmt);
    }

    sLength[0] = strlen(sId);
    if (SQLBindParameter(sStmt,
                         6,
                         SQL_PARAM_INPUT,
                         SQL_C_CHAR,
                         SQL_VARCHAR,
                         0,
                         0,
                         sId,
                         0,
                         (SQLLEN *)&sLength[0]) == SQL_ERROR)
    {
        outError("BindParameter error 6", sStmt);
    }

    sLength[1] = strlen(sSrcIp);
    if (SQLBindParameter(sStmt,
                         7,
                         SQL_PARAM_INPUT,
                         SQL_C_CHAR,
                         SQL_IPV4,
                         0,
                         0,
                         sSrcIp,
                         0,
                         (SQLLEN *)&sLength[1]) == SQL_ERROR)
    {
        outError("BindParameter error 7", sStmt);
    }

    sLength[2] = strlen(sDstIp);
    if (SQLBindParameter(sStmt,
                         8,
                         SQL_PARAM_INPUT,
                         SQL_C_CHAR,
                         SQL_IPV6,
                         0,
                         0,
                         sDstIp,
                         0,
                         (SQLLEN *)&sLength[2]) == SQL_ERROR)
    {
        outError("BindParameter error 8", sStmt);
    }

    if (SQLBindParameter(sStmt,
                         9,
                         SQL_PARAM_INPUT,
                         SQL_C_SBIGINT,
                         SQL_DATE,
                         0,
                         0,
                         &reg_date,
                         0,
                         NULL) == SQL_ERROR)
```

```
        {
            outError("BindParameter error 9", sStmt);
        }

        sLength[3] = strlen(sLog);
        if (SQLBindParameter(sStmt,
                             10,
                             SQL_PARAM_INPUT,
                             SQL_C_CHAR,
                             SQL_VARCHAR,
                             0,
                             0,
                             sLog,
                             0,
                             (SQLLEN *)&sLength[3]) == SQL_ERROR)
        {
            outError("BindParameter error 10", sStmt);
        }

        sLength[4] = strlen(sImage);
        if (SQLBindParameter(sStmt,
                             11,
                             SQL_PARAM_INPUT,
                             SQL_C_CHAR,
                             SQL_BINARY,
                             0,
                             0,
                             sImage,
                             0,
                             (SQLLEN *)&sLength[4]) == SQL_ERROR)
        {
            outError("BindParameter error 11", sStmt);
        }

        if( SQLExecute(sStmt) == SQL_ERROR) {
            outError("prepare execute error", sStmt);
        }

        printf("%d prepared record inserted\n", i);

    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP)) {
        outError("FreeStmt", sStmt);
    }
}

int main()
{
    connectDB();
    createTable();
    prepareInsert();
    selectTable();
    disconnectDB();

    return 0;
}
```

If you register sample3_prepare.c in the Makefile, compile and run it, it will appear as follows.

```
[mach@localhost cli]$ make

[mach@localhost cli]$ ./sample3_prepare

connected ...
1 prepared record inserted
2 prepared record inserted
3 prepared record inserted
4 prepared record inserted
5 prepared record inserted
6 prepared record inserted
7 prepared record inserted
8 prepared record inserted
9 prepared record inserted
===== 0 ========
seq = 9, score = 18, total = 270000, percentage = 1.11, ratio = 3.7037e-05, id = id-9, srcip = 192.168.0.9,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0009, regdate = 1970-01-01 09:00:00, log = log-9, image =
696D6167652D39
===== 1 ========
seq = 8, score = 16, total = 240000, percentage = 1.12, ratio = 3.75e-05, id = id-8, srcip = 192.168.0.8,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0008, regdate = 1970-01-01 09:00:00, log = log-8, image =
696D6167652D38
===== 2 ========
seq = 7, score = 14, total = 210000, percentage = 1.14, ratio = 3.80952e-05, id = id-7, srcip = 192.168.0.7,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0007, regdate = 1970-01-01 09:00:00, log = log-7, image =
696D6167652D37
===== 3 ========
seq = 6, score = 12, total = 180000, percentage = 1.17, ratio = 3.88889e-05, id = id-6, srcip = 192.168.0.6,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0006, regdate = 1970-01-01 09:00:00, log = log-6, image =
696D6167652D36
===== 4 ========
seq = 5, score = 10, total = 150000, percentage = 1.20, ratio = 4e-05, id = id-5, srcip = 192.168.0.5, dstip
= 2001:0DB8:0000:0000:0000:0000:1428:0005, regdate = 1970-01-01 09:00:00, log = log-5, image = 696D6167652D35
===== 5 ========
seq = 4, score = 8, total = 120000, percentage = 1.25, ratio = 4.16667e-05, id = id-4, srcip = 192.168.0.4,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0004, regdate = 1970-01-01 09:00:00, log = log-4, image =
696D6167652D34
===== 6 ========
seq = 3, score = 6, total = 90000, percentage = 1.33, ratio = 4.44444e-05, id = id-3, srcip = 192.168.0.3,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0003, regdate = 1970-01-01 09:00:00, log = log-3, image =
696D6167652D33
===== 7 ========
seq = 2, score = 4, total = 60000, percentage = 1.50, ratio = 5e-05, id = id-2, srcip = 192.168.0.2, dstip =
2001:0DB8:0000:0000:0000:0000:1428:0002, regdate = 1970-01-01 09:00:00, log = log-2, image = 696D6167652D32
===== 8 ========
seq = 1, score = 2, total = 30000, percentage = 2.00, ratio = 6.66667e-05, id = id-1, srcip = 192.168.0.1,
dstip = 2001:0DB8:0000:0000:0000:0000:1428:0001, regdate = 1970-01-01 09:00:00, log = log-1, image =
696D6167652D31
```

Extension Function Append Example

In Machbase, the append protocol is provided by reading a large amount of data from a file and inputting it at a high speed. Let's write an example program that uses this Append protocol.
First, let's look at an example of how to append to the various types provided by Machbase. The Append method has its own settings for each type. Th erefore, if you know how to use every method, you will be able to write programs more efficiently. All the methods are listed in the example code at the bottom.
The file name is sample4_append1.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <machbase_sqlcli.h>
#include <arpa/inet.h>

#if __linux__
#include <sys/time.h>
```

```
#endif

#if defined(SUPPORT_STRUCT_TM)
# include <time.h>
#endif

#define MACHBASE_PORT_NO 5656
#define MAX_APPEND_COUNT 0xFFFFFFFF
#define ERROR_CHECK_COUNT 100

#define ERROR -1
#define SUCCESS 0

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB();
void disconnectDB();
void outError(const char *aMsg);
void executeDirectSQL(const char *aSQL, int aErrIgnore);
void createTable();
void appendOpen();
void appendData();
int appendClose();
time_t getTimeStamp();

int main()
{
    unsigned int sCount=0;
    time_t sStartTime, sEndTime;

    connectDB();
    createTable();

    appendOpen();
    sStartTime = getTimeStamp();
    appendData();
    sEndTime = getTimeStamp();
    appendClose();

    printf("timegap = %ld microseconds for %d records\n", sEndTime - sStartTime, sCount);
    printf("%.2f records/second\n", ((double)sCount/(double)(sEndTime - sStartTime))*1000000);

    disconnectDB();
    return SUCCESS;
}

void connectDB()
{
    char sConnStr[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        outError("SQLAllocEnv error!!");
    }

    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        outError("SQLAllocConnect error!!");
    }

    sprintf(sConnStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)sConnStr, SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        outError("connection error\n");
    }
```

```
    if (SQL_ERROR == SQLAllocStmt(gCon, &gStmt) )
    {
        outError("AllocStmt error");
    }

    printf("connected ... \n");
}

void disconnectDB()
{
    if( SQL_ERROR == SQLFreeStmt(gStmt, SQL_DROP) )
    {
        outError("SQLFreeStmt error");
    }

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        outError("disconnect error");
    }
    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg)
{
    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    printf("ERROR : (%s)\n", aMsg);
    if (SQL_SUCCESS == SQLError( gEnv, gCon, gStmt, NULL, &sErrorNo,
                                 sErrorMsg, 1024, &sMsgLength ))
    {
        printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
    }

    if( gStmt )
    {
        SQLFreeStmt(gStmt, SQL_DROP);
    }

    if( gCon )
    {
        SQLFreeConnect( gCon );
    }

    if( gEnv )
    {
        SQLFreeEnv( gEnv );
    }
    exit(ERROR);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT sStmt;

    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error");
    }

    if (SQLExecDirect(sStmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error");
    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP))
    {
```

```
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error");
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(short1 short, integer1 integer, long1 long, float1 float,
double1 double, datetime1 datetime, varchar1 varchar(10), ip ipv4, ip2 ipv6, text1 text, bin1 binary)",
0);
}

void appendOpen()
{
    const char *sTableName = "CLI_SAMPLE";

    if( SQLAppendOpen(gStmt, (SQLCHAR *)sTableName, ERROR_CHECK_COUNT) != SQL_SUCCESS )
    {
        outError("SQLAppendOpen error");
    }

    printf("append open ok\n");
}

void appendData()
{
    SQL_APPEND_PARAM sParam[11];
    char sVarchar[10] = {0, };
    char sText[100] = {0, };
    char sBinary[100] = {0, };

    memset(sParam, 0, sizeof(sParam));

    /* NULL FOR ALL*/
    /* fixed column */
    sParam[0].mShort = SQL_APPEND_SHORT_NULL;
    sParam[1].mInteger = SQL_APPEND_INTEGER_NULL;
    sParam[2].mLong = SQL_APPEND_LONG_NULL;
    sParam[3].mFloat = SQL_APPEND_FLOAT_NULL;
    sParam[4].mDouble = SQL_APPEND_DOUBLE_NULL;
    /* datetime */
    sParam[5].mDateTime.mTime = SQL_APPEND_DATETIME_NULL;
    /* varchar */
    sParam[6].mVarchar.mLength = SQL_APPEND_VARCHAR_NULL;
    /* ipv4 */
    sParam[7].mIP.mLength = SQL_APPEND_IP_NULL;
    /* ipv6 */
    sParam[8].mIP.mLength = SQL_APPEND_IP_NULL;
    /* text */
    sParam[9].mText.mLength = SQL_APPEND_TEXT_NULL;
    /* binary */
    sParam[10].mBinary.mLength = SQL_APPEND_BINARY_NULL;
    SQLAppendDataV2(gStmt, sParam);

    /* FIXED COLUMN Value */
    sParam[0].mShort = 2;
    sParam[1].mInteger = 4;
    sParam[2].mLong = 6;
    sParam[3].mFloat = 8.4;
    sParam[4].mDouble = 10.9;
    SQLAppendDataV2(gStmt, sParam);

    /* DATETIME : absolute value */
    sParam[5].mDateTime.mTime = MACH_UINT64_LITERAL(1000000000);
    SQLAppendDataV2(gStmt, sParam);

    /* DATETIME : current */
    sParam[5].mDateTime.mTime = SQL_APPEND_DATETIME_NOW;
    SQLAppendDataV2(gStmt, sParam);
```

```
    /* DATETIME : string format*/
    sParam[5].mDateTime.mTime = SQL_APPEND_DATETIME_STRING;
    sParam[5].mDateTime.mDateStr = "23/May/2014:17:41:28";
    sParam[5].mDateTime.mFormatStr = "DD/MON/YYYY:HH24:MI:SS";
    SQLAppendDataV2(gStmt, sParam);

    /* DATETIME : struct tm format*/
    sParam[5].mDateTime.mTime = SQL_APPEND_DATETIME_STRUCT_TM;
    sParam[5].mDateTime.mTM.tm_year = 2000 - 1900;
    sParam[5].mDateTime.mTM.tm_mon = 11;
    sParam[5].mDateTime.mTM.tm_mday = 31;
    SQLAppendDataV2(gStmt, sParam);

    /* VARCHAR : string */
    strcpy(sVarchar, "MY VARCHAR");
    sParam[6].mVar.mLength = strlen(sVarchar);
    sParam[6].mVar.mData = sVarchar;
    SQLAppendDataV2(gStmt, sParam);

    /* IPv4 : ipv4 from binary bytes */
    sParam[7].mIP.mLength = SQL_APPEND_IP_IPV4;
    sParam[7].mIP.mAddr[0] = 127;
    sParam[7].mIP.mAddr[1] = 0;
    sParam[7].mIP.mAddr[2] = 0;
    sParam[7].mIP.mAddr[3] = 1;
    SQLAppendDataV2(gStmt, sParam);

    /* IPv4 : ipv4 from binary */
    sParam[7].mIP.mLength = SQL_APPEND_IP_IPV4;
    *(in_addr_t *)(sParam[7].mIP.mAddr) = inet_addr("192.168.0.1");
    SQLAppendDataV2(gStmt, sParam);

    /* IPv4 : ipv4 from string */
    sParam[7].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[7].mIP.mAddrString = "203.212.222.111";
    SQLAppendDataV2(gStmt, sParam);

    /* IPv6 : ipv6 from binary bytes */
    sParam[8].mIP.mLength = SQL_APPEND_IP_IPV6;
    sParam[8].mIP.mAddr[0] = 127;
    sParam[8].mIP.mAddr[1] = 127;
    sParam[8].mIP.mAddr[2] = 127;
    sParam[8].mIP.mAddr[3] = 127;
    sParam[8].mIP.mAddr[4] = 127;
    sParam[8].mIP.mAddr[5] = 127;
    sParam[8].mIP.mAddr[6] = 127;
    sParam[8].mIP.mAddr[7] = 127;
    sParam[8].mIP.mAddr[8] = 127;
    sParam[8].mIP.mAddr[9] = 127;
    sParam[8].mIP.mAddr[10] = 127;
    sParam[8].mIP.mAddr[11] = 127;
    sParam[8].mIP.mAddr[12] = 127;
    sParam[8].mIP.mAddr[13] = 127;
    sParam[8].mIP.mAddr[14] = 127;
    sParam[8].mIP.mAddr[15] = 127;
    SQLAppendDataV2(gStmt, sParam);
    sParam[8].mIP.mLength = SQL_APPEND_IP_NULL; /* recover */

    /* TEXT : string */
    memset(sText, 'X', sizeof(sText));
    sParam[9].mVar.mLength = 100;
    sParam[9].mVar.mData = sText;
    SQLAppendDataV2(gStmt, sParam);

    /* BINARY : datas */
    memset(sBinary, 0xFA, sizeof(sBinary));
    sParam[10].mVar.mLength = 100;
    sParam[10].mVar.mData = sBinary;
    SQLAppendDataV2(gStmt, sParam);
}
```

```
int appendClose()
{
    int sSuccessCount = 0;
    int sFailureCount = 0;

    if( SQLAppendClose(gStmt, &sSuccessCount, &sFailureCount) != SQL_SUCCESS )
    {
        outError("SQLAppendClose error");
    }

    printf("append close ok\n");
    printf("success : %d, failure : %d\n", sSuccessCount, sFailureCount);
    return sSuccessCount;
}

time_t getTimeStamp()
{
#if _WIN32 || _WIN64

#if defined(_MSC_VER) || defined(_MSC_EXTENSIONS)
#define DELTA_EPOCH_IN_MICROSECS 11644473600000000Ui64
#else
#define DELTA_EPOCH_IN_MICROSECS 11644473600000000ULL
#endif
    FILETIME sFT;
    unsigned __int64 sTempResult = 0;

    GetSystemTimeAsFileTime(&sFT);

    sTempResult |= sFT.dwHighDateTime;
    sTempResult <<= 32;
    sTempResult |= sFT.dwLowDateTime;

    sTempResult -= DELTA_EPOCH_IN_MICROSECS;
    sTempResult /= 10;

    return sTempResult;
#else
    struct timeval sTimeVal;
    int sRet;

    sRet = gettimeofday(&sTimeVal, NULL);

    if (sRet == 0)
    {
        return (time_t)(sTimeVal.tv_sec * 1000000 + sTimeVal.tv_usec);
    }
    else
    {
        return 0;
    }
#endif
}
```

If you register sample4_append1.c in the Makefile, compile and run it, it will appear as follows.

```
[mach@localhost cli]$ make sample4_append1
gcc -c -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -I
/home/mach/machbase_home/include -I. -L//home/mach/machbase_home/include -osample4_append1.o sample4_append1.
c
gcc -m64 -mtune=k8 -L/home/mach/machbase_home/lib -osample4_append1 sample4_append1.o -lmachcli -L/home/mach
/machbase_home/lib -lm -lpthread -ldl -lrt -rdynamic
[mach@localhost cli]$ ./sample4_append1
connected ...
append open ok
append close ok
success : 13, failure : 0
```

```
timegap = 48 microseconds for 13 records
270833.33 records/second
[mach@localhost cli]$

You can check what is inserted after MACH_SQL.

Mach> select * from CLI_SAMPLE;
SHORT1 INTEGER1 LONG1 FLOAT1 DOUBLE1
-------------------------------------------------------------------------------------------------------------------
DATETIME1 VARCHAR1 IP IP2
-------------------------------------------------------------------------------------------------------------------
-----------------
TEXT1
--------------------------------------------------------------------------------
BIN1
--------------------------------------------------------------------------------
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 203.212.222.111 NULL
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
FAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFA
FAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFA
FAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFA
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 203.212.222.111 NULL
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 203.212.222.111 7F7F:7F7F:7F7F:7F7F:7F7F:7F7F:7F7F:7F7F:7F7F:7F7F
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 203.212.222.111 NULL
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 192.168.0.1 NULL
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 127.0.0.1 NULL
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR NULL NULL
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 NULL NULL NULL
NULL
NULL
2 4 6 8.4 10.9
2014-05-23 17:41:28 000:000:000 NULL NULL NULL
NULL
NULL
2 4 6 8.4 10.9
2015-04-09 16:44:11 134:256:000 NULL NULL NULL
NULL
NULL
2 4 6 8.4 10.9
1970-01-01 09:00:01 000:000:000 NULL NULL NULL
NULL
NULL
2 4 6 8.4 10.9
1970-01-01 09:00:00 000:000:000 NULL NULL NULL
NULL
NULL
[12] row(s) selected.
```

Now let's use a fast append method using a file. This is an example useful for fast input of large amounts of logs, packets, etc. used in business. The file name is sample4_append2.c.
You have to save the data to be entered in advance in data.txt.

```
./make_data
```

Modifying the given make_data.c gives you the opportunity to create a data.txt file for your situation.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <machbase_sqlcli.h>

#define MACHBASE_PORT_NO 5656
#define MAX_APPEND_COUNT 0xFFFFFFFF
#define ERROR_CHECK_COUNT 100

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB();
void disconnectDB();
void outError(const char *aMsg);
void executeDirectSQL(const char *aSQL, int aErrIgnore);
void createTable();
void appendOpen();
int appendData();
void appendClose();
time_t getTimeStamp();

int main()
{
    unsigned int sCount=0;
    time_t sStartTime, sEndTime;

    connectDB();
    createTable();

    appendOpen();
    sStartTime = getTimeStamp();
    sCount = appendData();
    sEndTime = getTimeStamp();

    appendClose();

    printf("timegap = %ld microseconds for %d records\n", sEndTime - sStartTime, sCount);
    printf("%.2f records/second\n", ((double)sCount/(double)(sEndTime - sStartTime))*1000000);

    disconnectDB();

    return 0;
}

void connectDB()
{
    char sConnStr[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        outError("SQLAllocEnv error!!");
    }

    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
```

```
        outError("SQLAllocConnect error!!");
    }

    sprintf(sConnStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)sConnStr, SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        outError("connection error!!");
    }

    if( SQL_ERROR == SQLAllocStmt(gCon, &gStmt) )
    {
        outError("SQLAllocStmt error!!");
    }

    printf("connected ... \n");
}

void disconnectDB()
{
    if( SQL_ERROR == SQLFreeStmt(gStmt, SQL_DROP) )
    {
        outError("SQLFreeStmt error");
    }

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        outError("disconnect error");
    }

    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg)
{
    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    printf("ERROR : (%s)\n", aMsg);

    if (SQL_SUCCESS == SQLError( gEnv, gCon, gStmt, NULL, &sErrorNo,
                                 sErrorMsg, 1024, &sMsgLength ))
    {
        printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
    }

    if( gStmt )
    {
        SQLFreeStmt( gStmt, SQL_DROP );
    }
    if( gCon )
    {
        SQLFreeConnect( gCon );
    }
    if( gEnv )
    {
        SQLFreeEnv( gEnv );
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT sStmt;

    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR)
    {
```

```
        if (aErrIgnore != 0) return;
        outError("AllocStmt error");
    }

    if (SQLExecDirect(sStmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("sql_exec_direct error");
    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error");
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(seq short, score integer, total long, percentage float,
ratio double, id varchar(10), srcip ipv4, dstip ipv6, reg_date datetime, tlog text, image binary)", 0);

    printf("table created\n");
}

void appendOpen()
{
    const char *sTableName = "CLI_SAMPLE";

    if( SQLAppendOpen(gStmt, (SQLCHAR *)sTableName, ERROR_CHECK_COUNT) != SQL_SUCCESS )
    {
        outError("SQLAppendOpen error!!");
    }

    printf("append open ok\n");
}

int appendData()
{
    FILE *sFp;
    char sBuf[1024];
    int j;
    char *sToken;
    unsigned int sCount=0;
    SQL_APPEND_PARAM sParam[11];

    sFp = fopen("data.txt", "r");
    if( !sFp )
    {
        printf("file open error\n");
        exit(-1);
    }

    printf("append data start\n");

    memset(sBuf, 0, sizeof(sBuf));

    while( fgets(sBuf, 1024, sFp ) != NULL )
    {
        if( strlen(sBuf) < 1)
        {
            break;
        }

        j=0;
        sToken = strtok(sBuf,",");

        while( sToken != NULL )
        {
            memset(sParam+j, 0, sizeof(sParam));
```

```
            switch(j){
                case 0 : sParam[j].mShort = atoi(sToken); break; //short
                case 1 : sParam[j].mInteger = atoi(sToken); break; //int
                case 2 : sParam[j].mLong = atol(sToken); break; //long
                case 3 : sParam[j].mFloat = atof(sToken); break; //float
                case 4 : sParam[j].mDouble = atof(sToken); break; //double
                case 5 : //string
                case 9 : //text
                case 10 : //binary
                         sParam[j].mVar.mLength = strlen(sToken);
                         strcpy(sParam[j].mVar.mData, sToken);
                         break;
                case 6 : //ipv4
                case 7 : //ipv6
                         sParam[j].mIP.mLength = SQL_APPEND_IP_STRING;
                         strcpy(sParam[j].mIP.mAddrString, sToken);
                         break;
                case 8 : //datetime
                         sParam[j].mDateTime.mTime = SQL_APPEND_DATETIME_STRING;
                         strcpy(sParam[j].mDateTime.mDateStr, sToken);
                         sParam[j].mDateTime.mFormatStr = "DD/MON/YYYY:HH24:MI:SS";
                         break;
            }

            sToken = strtok(NULL, ",");

            j++;
        }
        if( SQLAppendDataV2(gStmt, sParam) != SQL_SUCCESS )
        {
            printf("SQLAppendData error\n");
            return 0;
        }
        if ( ((sCount++) % 10000) == 0)
        {
            printf(".");
        }

        if( ((sCount) % 100) == 0 )
        {
            if( SQLAppendFlush( gStmt ) != SQL_SUCCESS )
            {
                outError("SQLAppendFlush error");
            }
        }
        if (sCount == MAX_APPEND_COUNT)
        {
            break;
        }
    }

    printf("\nappend data end\n");

    fclose(sFp);

    return sCount;
}

void appendClose()
{
    int sSuccessCount = 0;
    int sFailureCount = 0;

    if( SQLAppendClose(gStmt, &sSuccessCount, &sFailureCount) != SQL_SUCCESS )
    {
        outError("SQLAppendClose error");
    }

    printf("append close ok\n");
    printf("success : %d, failure : %d\n", sSuccessCount, sFailureCount);
}
```

```
time_t getTimeStamp()
{
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec*1000000+tv.tv_usec;
}
```

If you register sample4_append2.c in the Makefile, compile and run it, it will appear as follows.

```
[mach@localhost cli]$ make
gcc -c -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -I
/home/mach/machbase_home/include -I. -L//home/mach/machbase_home/include -osingle_append2.o single_append2.c
gcc -m64 -mtune=k8 -L/home/mach/machbase_home/lib -osingle_append2 single_append2.o -lmachcli -L/home/mach
/machbase_home/lib -lm -lpthread -ldl -lrt -rdynamic
[mach@localhost cli]$ ./single_append2
connected ...
table created
append open ok
append data start
.....................................................................................................
append data end
append close ok
success : 1000000, failure : 0
timegap = 1641503 microseconds for 1000000 records
609197.79 records/second
```

Acquiring Table Column Information Example

There are a number of ways to obtain table column information, but we will look at how to use SQLDescribeCol and SQLColumns.

SQLDescribeCol

SQLDescribeCol is a function that retrieves the number, name, buffer size, length, and type of table columns.

The sample file name is sample5_describe.c.

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

#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB()
{
    char connStr[1024];

    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }
```

```
    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }

    sprintf(connStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)connStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");

        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &errNo,
                                      errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }

    if (SQLAllocStmt(gCon, &gStmt) == SQL_ERROR)
    {
        outError("AllocStmt error", gStmt);
    }

    printf("connected ... \n");

}

void disconnectDB()
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        printf("disconnect error\n");

        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &errNo,
                                     errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
    }

    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg, SQLHSTMT stmt)
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    printf("ERROR : (%s)\n", aMsg);

    if (SQL_SUCCESS == SQLError( gEnv, gCon, stmt, NULL, &errNo,
                                 errMsg, 1024, &msgLength ))
    {
        printf(" mach-%d : %s\n", errNo, errMsg);
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
```

```
{
    SQLHSTMT stmt;

    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error", stmt);
    }

    if (SQLExecDirect(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error", stmt);
    }

    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error", stmt);
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(seq short, score integer, total long, percentage float,
ratio double, id varchar(10), srcip ipv4, dstip ipv6, reg_date datetime, tlog text, image binary)", 0);

}

int main()
{
    char sSqlStr[] = "select * from cli_sample";
    SQLCHAR sColName[32];
    SQLSMALLINT sColType;
    SQLSMALLINT sColNameLen;
    SQLSMALLINT sNullable;
    SQLULEN sColLen;
    SQLSMALLINT sDecimalDigits;
    SQLLEN sOutlen;
    SQLCHAR* sData;
    SQLLEN sDisplaySize;
    int i;

    SQLSMALLINT sColumns;

    connectDB();

    createTable();

    if(SQLPrepare(gStmt, (SQLCHAR*)sSqlStr, SQL_NTS))
    {
        outError("sql prepare fail", gStmt);
        return -1;
    }

    if(SQLNumResultCols(gStmt, &sColumns) != SQL_SUCCESS )
    {
        printf("get col length error \n");
        return -1;
    }

    printf("----------------------------------------------------------------\n");
    printf("%32s%16s%10s\n","Name","Type","Length");
    printf("----------------------------------------------------------------\n");

    for(i = 0; i < sColumns; i++)
    {
        SQLDescribeCol(gStmt,
                       (SQLUSMALLINT)(i + 1),
```

```
                        sColName,
                        sizeof(sColName),
                        &sColNameLen,
                        &sColType,
                        (SQLULEN *)&sColLen,
                        &sDecimalDigits,
                        (SQLSMALLINT *)&sNullable);

        printf("%32s%16d%10d\n",sColName, sColType, sColLen);
    }

    printf("----------------------------------------------------------------\n");

    disconnectDB();

    return 0;
}
```

If you add the above file and run make, you can see the contents of the column as shown below.

```
[mach@localhost cli]$ make

[mach@localhost cli]$ ./sample5_describe
connected ...
----------------------------------------------------------------
Name Type Length
----------------------------------------------------------------
SEQ 5 5
SCORE 4 10
TOTAL -5 19
PERCENTAGE 6 27
RATIO 8 27
ID 12 10
SRCIP 2104 15
DSTIP 2106 60
REG_DATE 9 31
TLOG 2100 67108864
IMAGE -2 67108864
----------------------------------------------------------------
[mach@localhost cli]$
```

SQLColumns

SQLColumns is a function that can find the information of the columns existing in the current table. Machbase also supports the above functions and can be used to find out the information of each column.

The file name is sample6_columns.c.

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

#include <time.h>

#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB()
{
```

```
    char connStr[1024];

    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }

    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }

    sprintf(connStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)connStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");

        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &errNo,
                                      errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }

    if (SQLAllocStmt(gCon, &gStmt) == SQL_ERROR)
    {
        outError("AllocStmt error", gStmt);
    }

    printf("connected ... \n");

}

void disconnectDB()
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        printf("disconnect error\n");

        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &errNo,
                                     errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
    }

    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg, SQLHSTMT stmt)
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];
```

```
    printf("ERROR : (%s)\n", aMsg);

    if (SQL_SUCCESS == SQLError( gEnv, gCon, stmt, NULL, &errNo,
                                 errMsg, 1024, &msgLength ))
    {
        printf(" mach-%d : %s\n", errNo, errMsg);
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT stmt;

    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error", stmt);
    }

    if (SQLExecDirect(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error", stmt);
    }

    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error", stmt);
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(seq short, score integer, total long, percentage float,
ratio double, id varchar(10), srcip ipv4, dstip ipv6, reg_date datetime, tlog text, image binary)", 0);
}

int main()
{
    SQLCHAR sColName[32];
    SQLSMALLINT sColType;
    SQLCHAR sColTypeName[16];
    SQLSMALLINT sColNameLen;
    SQLSMALLINT sColTypeLen;
    SQLSMALLINT sNullable;
    SQLULEN sColLen;
    SQLSMALLINT sDecimalDigits;
    SQLLEN sOutlen;
    SQLCHAR* sData;
    SQLLEN sDisplaySize;
    int i;

    SQLSMALLINT sColumns;

    connectDB();

    createTable();

    if(SQLColumns(gStmt, NULL, 0, NULL, 0, "cli_sample", SQL_NTS, NULL, 0) != SQL_SUCCESS)
    {
        printf("sql columns error!\n");
        return -1;
    }

    SQLBindCol(gStmt, 4, SQL_C_CHAR, sColName, sizeof(sColName), &sColNameLen);
    SQLBindCol(gStmt, 5, SQL_C_SSHORT, &sColType, 0, &sColTypeLen);
    SQLBindCol(gStmt, 6, SQL_C_CHAR, sColTypeName, sizeof(sColTypeName), NULL);
```

```
    SQLBindCol(gStmt, 7, SQL_C_SLONG, &sColLen, 0, NULL);

    printf("--------------------------------------------------------------------------------\n");
    printf("%32s%16s%16s%10s\n","Name","Type","TypeName","Length");
    printf("--------------------------------------------------------------------------------\n");

    while( SQLFetch(gStmt) != SQL_NO_DATA )
    {
        printf("%32s%16d%16s%10d\n",sColName, sColType, sColTypeName, sColLen);
    }
    printf("--------------------------------------------------------------------------------\n");

    disconnectDB();

    return 0;
}
```

Add the above file and run make. The results are as follows.

```
[mach@localhost cli]$ make

[mach@localhost cli]$ ./sample6_columns
connected ...
--------------------------------------------------------------------------------
Name Type TypeName Length
--------------------------------------------------------------------------------
_ARRIVAL_TIME 93 DATE 31
SEQ 5 SMALLINT 5
SCORE 4 INTEGER 10
TOTAL -5 BIGINT 19
PERCENTAGE 6 FLOAT 27
RATIO 8 DOUBLE 27
ID 12 VARCHAR 10
SRCIP 2104 IPV4 15
DSTIP 2106 IPV6 60
REG_DATE 93 DATE 31
TLOG 2100 TEXT 67108864
IMAGE -2 BINARY 67108864
--------------------------------------------------------------------------------
```

Multi-Thread Append Example

An example of using multiple threads in one program to append to multiple tables.

The file name is sample8_multi_session_multi_table.c.

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

#define MACHBASE_PORT_NO       5656
#define ERROR_CHECK_COUNT   100

#define LOG_FILE_CNT        3
#define MAX_THREAD_NUM      LOG_FILE_CNT

#define RC_FAILURE          -1
#define RC_SUCCESS          0

#define UNUSED(aVar) do { (void)(aVar); } while(0)

char *gTableName[LOG_FILE_CNT] = {"table_f1", "table_f2", "table_event"};
char *gFileName[LOG_FILE_CNT] =  {"suffle_data1.txt","suffle_data2.txt","suffle_data3.txt"};
```

```
void printError(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, char *aMsg);
int connectDB(SQLHENV *aEnv, SQLHDBC *aCon);
void disconnectDB(SQLHENV aEnv, SQLHDBC aCon);
int executeDirectSQL(SQLHENV aEnv, SQLHDBC aCon, const char *aSQL, int aErrIgnore);
int appendOpen(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, char* aTableName);
int appendClose(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt);
int createTables(SQLHENV aEnv, SQLHDBC aCon);

/*
 * error code returned from CLI lib
 */
void printError(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, char *aMsg)
{
    SQLINTEGER      sNativeError;
    SQLCHAR         sErrorMsg[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR         sSqlState[SQL_SQLSTATE_SIZE + 1];
    SQLSMALLINT     sMsgLength;

    if( aMsg != NULL )
    {
        printf("%s\n", aMsg);
    }

    if( SQLError(aEnv, aCon, aStmt, sSqlState, &sNativeError,
                 sErrorMsg, SQL_MAX_MESSAGE_LENGTH, &sMsgLength) == SQL_SUCCESS )
    {
        printf("SQLSTATE-[%s], Machbase-[%d][%s]\n", sSqlState, sNativeError, sErrorMsg);
    }
}

/*
 * error code returned from Machbase server
 */

void appendDumpError(SQLHSTMT    aStmt,
                     SQLINTEGER  aErrorCode,
                     SQLPOINTER  aErrorMessage,
                     SQLLEN      aErrorBufLen,
                     SQLPOINTER  aRowBuf,
                     SQLLEN      aRowBufLen)
{
    char       sErrMsg[1024] = {0, };
    char       sRowMsg[32 * 1024] = {0, };

    UNUSED(aStmt);

    if (aErrorMessage != NULL)
    {
        strncpy(sErrMsg, (char *)aErrorMessage, aErrorBufLen);
    }

    if (aRowBuf != NULL)
    {
        strncpy(sRowMsg, (char *)aRowBuf, aRowBufLen);
    }

    fprintf(stdout, "Append Error : [%d][%s]\n[%s]\n\n", aErrorCode, sErrMsg, sRowMsg);
}


int connectDB(SQLHENV *aEnv, SQLHDBC *aCon)
{
    char sConnStr[1024];

    if( SQLAllocEnv(aEnv) != SQL_SUCCESS )
    {
        printf("SQLAllocEnv error\n");
        return RC_FAILURE;
    }

    if( SQLAllocConnect(*aEnv, aCon) != SQL_SUCCESS )
```

```
    {
        printf("SQLAllocConnect error\n");

        SQLFreeEnv(*aEnv);
        *aEnv = SQL_NULL_HENV;

        return RC_FAILURE;
    }

    sprintf(sConnStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if( SQLDriverConnect( *aCon, NULL,
                          (SQLCHAR *)sConnStr,
                          SQL_NTS,
                          NULL, 0, NULL,
                          SQL_DRIVER_NOPROMPT ) != SQL_SUCCESS
      )
    {

        printError(*aEnv, *aCon, NULL, "SQLDriverConnect error");

        SQLFreeConnect(*aCon);
        *aCon = SQL_NULL_HDBC;

        SQLFreeEnv(*aEnv);
        *aEnv = SQL_NULL_HENV;

        return RC_FAILURE;
    }

    return RC_SUCCESS;
}


void disconnectDB(SQLHENV aEnv, SQLHDBC aCon)
{
    if( SQLDisconnect(aCon) != SQL_SUCCESS )
    {
        printError(aEnv, aCon, NULL, "SQLDisconnect error");
    }

    SQLFreeConnect(aCon);
    aCon = SQL_NULL_HDBC;

    SQLFreeEnv(aEnv);
    aEnv = SQL_NULL_HENV;
}


int executeDirectSQL(SQLHENV aEnv, SQLHDBC aCon, const char *aSQL, int aErrIgnore)
{
    SQLHSTMT sStmt = SQL_NULL_HSTMT;

    if( SQLAllocStmt(aCon, &sStmt) != SQL_SUCCESS )
    {
        if( aErrIgnore == 0 )
        {
            printError(aEnv, aCon, sStmt, "SQLAllocStmt Error");
            return RC_FAILURE;
        }
    }

    if( SQLExecDirect(sStmt, (SQLCHAR *)aSQL, SQL_NTS) != SQL_SUCCESS )
    {

        if( aErrIgnore == 0 )
        {
            printError(aEnv, aCon, sStmt, "SQLExecDirect Error");

            SQLFreeStmt(sStmt,SQL_DROP);
            sStmt = SQL_NULL_HSTMT;
```

```
            return RC_FAILURE;
        }
    }

    if( SQLFreeStmt(sStmt, SQL_DROP) != SQL_SUCCESS )
    {
        if (aErrIgnore == 0)
        {
            printError(aEnv, aCon, sStmt, "SQLFreeStmt Error");
            sStmt = SQL_NULL_HSTMT;
            return RC_FAILURE;
        }
    }
    sStmt = SQL_NULL_HSTMT;

    return RC_SUCCESS;
}


int appendOpen(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, char* aTableName)
{
    if( aTableName == NULL )
    {
        printf("append open wrong table name");
        return RC_FAILURE;
    }

    if( SQLAppendOpen(aStmt, (SQLCHAR *)aTableName, ERROR_CHECK_COUNT) != SQL_SUCCESS )
    {
        printError(aEnv, aCon, aStmt, "SQLAppendOpen error");
        return RC_FAILURE;
    }
    return RC_SUCCESS;
}


int appendClose(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt)
{
    int sSuccessCount = 0;
    int sFailureCount = 0;

    if( SQLAppendClose(aStmt, &sSuccessCount, &sFailureCount) != SQL_SUCCESS )
    {
        printError(aEnv, aCon, aStmt, "SQLAppendClose error");
        return RC_FAILURE;
    }

    printf("append result success : %d, failure : %d\n", sSuccessCount, sFailureCount);

    return RC_SUCCESS;
}


int createTables(SQLHENV aEnv, SQLHDBC aCon)
{
    int      i;
    char    *sSchema[] = { "srcip1 ipv4, srcip2 ipv6, srcport short, dstip1 ipv4, dstip2 ipv6, dstport
short, data1 long, data2 long",
        "srcip1 ipv4, srcip2 ipv6, srcport short, dstip1 ipv4, dstip2 ipv6, dstport short, data1 long,
data2 long",
        "machine ipv4, err integer, msg varchar(30)"
    };

    char sDropQuery[256];
    char sCreateQuery[256];

    for(i = 0; i < LOG_FILE_CNT; i++)
    {
        snprintf(sDropQuery, 256, "DROP TABLE %s", gTableName[i]);
        snprintf(sCreateQuery, 256, "CREATE TABLE %s ( %s )", gTableName[i], sSchema[i]);
```

```
        executeDirectSQL(aEnv, aCon, sDropQuery, 1);
        executeDirectSQL(aEnv, aCon, sCreateQuery, 0);
    }

    return RC_SUCCESS;
}


int appendF1(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, FILE *aFp)
{
    SQL_APPEND_PARAM sParam[8];
    SQLRETURN        sRC;

    SQLINTEGER      sNativeError;
    SQLCHAR         sErrorMsg[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR         sSqlState[SQL_SQLSTATE_SIZE + 1];
    SQLSMALLINT     sMsgLength;

    char             sData[4][64];

    memset(sParam, 0, sizeof(sParam));

    fscanf(aFp, "%s %s %hd %s %s %hd %lld %lld\n",
           sData[0], sData[1], &sParam[2].mShort,
           sData[2], sData[3], &sParam[5].mShort,
           &sParam[6].mLong, &sParam[7].mLong);

    sParam[0].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = sData[0];

    sParam[1].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[1].mIP.mAddrString = sData[1];

    sParam[3].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[3].mIP.mAddrString = sData[2];

    sParam[4].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[4].mIP.mAddrString = sData[3];

    sRC = SQLAppendDataV2(aStmt, sParam);
    if( !SQL_SUCCEEDED(sRC) )
    {
        if( SQLError(aEnv, aCon, aStmt, sSqlState, &sNativeError,
                     sErrorMsg, SQL_MAX_MESSAGE_LENGTH, &sMsgLength) != SQL_SUCCESS )
        {
            return RC_FAILURE;
        }

        printf("SQLSTATE-[%s], Machbase-[%d][%s]\n", sSqlState, sNativeError, sErrorMsg);

        if( sNativeError != 9604 &&
            sNativeError != 9605 &&
            sNativeError != 9606 )
        {
            return RC_FAILURE;
        }
        else
        {
            //data value error in one record, so return success to keep attending
        }
    }
    return RC_SUCCESS;
}


int appendF2(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, FILE* aFp)
{
    SQL_APPEND_PARAM sParam[8];
    SQLRETURN        sRC;

    SQLINTEGER      sNativeError;
```

```
    SQLCHAR         sErrorMsg[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR         sSqlState[SQL_SQLSTATE_SIZE + 1];
    SQLSMALLINT     sMsgLength;

    char             sData[4][64];

    memset(sParam, 0, sizeof(sParam));

    fscanf(aFp, "%s %s %hd %s %s %hd %lld %lld\n",
           sData[0], sData[1], &sParam[2].mShort,
           sData[2], sData[3], &sParam[5].mShort,
           &sParam[6].mLong, &sParam[7].mLong);

    sParam[0].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = sData[0];

    sParam[1].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[1].mIP.mAddrString = sData[1];

    sParam[3].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[3].mIP.mAddrString = sData[2];

    sParam[4].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[4].mIP.mAddrString = sData[3];

    sRC = SQLAppendDataV2(aStmt, sParam);
    if( !SQL_SUCCEEDED(sRC) )
    {
        if( SQLError(aEnv, aCon, aStmt, sSqlState, &sNativeError,
                     sErrorMsg, SQL_MAX_MESSAGE_LENGTH, &sMsgLength) != SQL_SUCCESS )
        {
            return RC_FAILURE;
        }

        printf("SQLSTATE-[%s], Machbase-[%d][%s]\n", sSqlState, sNativeError, sErrorMsg);

        if( sNativeError != 9604 &&
            sNativeError != 9605 &&
            sNativeError != 9606 )
        {
            return RC_FAILURE;
        }
        else
        {
            //data value error in one record, so return success to keep attending
        }
    }
    return RC_SUCCESS;
}


int appendEvent(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, FILE* aFp)
{
    SQL_APPEND_PARAM sParam[3];
    SQLRETURN        sRC;

    SQLINTEGER      sNativeError;
    SQLCHAR         sErrorMsg[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR         sSqlState[SQL_SQLSTATE_SIZE + 1];
    SQLSMALLINT     sMsgLength;

    char             sData[2][20];

    memset(sParam, 0, sizeof(sParam));

    fscanf(aFp, "%s %d %s\n",sData[0], &sParam[1].mInteger, sData[1]);

    sParam[0].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = sData[0];

    sParam[2].mVarchar.mLength = strlen(sData[1]);
```

```
    sParam[2].mVarchar.mData = sData[1];

    sRC = SQLAppendDataV2(aStmt, sParam);
    if( !SQL_SUCCEEDED(sRC) )
    {
        if( SQLError(aEnv, aCon, aStmt, sSqlState, &sNativeError,
                     sErrorMsg, SQL_MAX_MESSAGE_LENGTH, &sMsgLength) != SQL_SUCCESS )
        {
            return RC_FAILURE;
        }

        printf("SQLSTATE-[%s], Machbase-[%d][%s]\n", sSqlState, sNativeError, sErrorMsg);

        if( sNativeError != 9604 &&
            sNativeError != 9605 &&
            sNativeError != 9606 )
        {
            return RC_FAILURE;
        }
        else
        {
            //data value error in one record, so return success to keep attending
        }
    }
    return RC_SUCCESS;
}


void *eachThread(void *aIdx)
{
    SQLHENV    sEnv = SQL_NULL_HENV;
    SQLHDBC    sCon = SQL_NULL_HDBC;
    SQLHSTMT   sStmt[LOG_FILE_CNT] = {SQL_NULL_HSTMT,};

    FILE*      sFp;
    int        i;
    int        sLogType;

    int        sThrNo = *(int *)aIdx;

    // Alloc ENV and DBC
    if( connectDB(&sEnv, &sCon) == RC_SUCCESS )
    {
        printf("[%d]connectDB success.\n", sThrNo);
    }
    else
    {
        printf("[%d]connectDB failure.\n", sThrNo);
        goto error;
    }

    // set timed flush true
    if( SQLSetConnectAppendFlush(sCon, 1) != SQL_SUCCESS )
    {
        printError(sEnv, sCon, NULL, "SQLSetConnectAppendFlush Error");
        goto error;
    }

    for( i = 0; i < LOG_FILE_CNT; i++ )
    {
        // Alloc stmt
        if( SQLAllocStmt(sCon,&sStmt[i]) != SQL_SUCCESS )
        {
            printError(sEnv, sCon, sStmt[i], "SQLAllocStmt Error");
            goto error;
        }

        if( appendOpen(sEnv, sCon, sStmt[i], gTableName[i]) == RC_FAILURE )
        {
            printError(sEnv, sCon, sStmt[i], "SQLAppendOpen Error");
            goto error;
```

```
    }
    else
    {
        printf("[%d-%d]appendOpen success.\n", sThrNo, i);
    }

    if( SQLAppendSetErrorCallback(sStmt[i], appendDumpError) != SQL_SUCCESS )
    {
        printError(sEnv, sCon, sStmt[i], "SQLAppendSetErrorCallback Error");
        goto error;
    }

    // set timed flush interval as 2 seconds
    if( SQLSetStmtAppendInterval(sStmt[i], 2000) != SQL_SUCCESS )
    {
        printError(sEnv, sCon, sStmt[i], "SQLSetStmtAppendInterval Error");
        goto error;
    }
}

sFp = fopen((char*)gFileName[sThrNo], "rt");
if( sFp == NULL )
{
    printf("file open error - [%d][%s]\n", sThrNo, gFileName[sThrNo]);
}
else
{
    printf("file open success - [%d][%s]\n", sThrNo, gFileName[sThrNo]);

    for( i = 0; !feof(sFp); i++ )
    {
        fscanf(sFp, "%d ", &sLogType);
        switch(sLogType)
        {
            case 1://f1
                if( appendF1(sEnv, sCon, sStmt[0], sFp) == RC_FAILURE )
                {
                    goto error;
                }
                break;
            case 2://f2
                if( appendF2(sEnv, sCon, sStmt[1],sFp) == RC_FAILURE )
                {
                    goto error;
                }
                break;
            case 3://event
                if(appendEvent(sEnv, sCon, sStmt[2], sFp) == RC_FAILURE )
                {
                    goto error;
                }
                break;
            default:
                printf("unknown type error\n");
                break;
        }

        if( (i%10000) == 0 )
        {
            fprintf(stdout, ".");
            fflush(stdout);
        }
    }
    printf("\n");

    fclose(sFp);
}

for( i = 0; i < LOG_FILE_CNT; i++)
{
    printf("[%d-%d]appendClose start...\n", sThrNo, i);
```

```
        if( appendClose(sEnv, sCon, sStmt[i]) == RC_FAILURE )
        {
            printf("[%d-%d]appendClose failure\n", sThrNo, i);
        }
        else
        {
            printf("[%d-%d]appendClose success\n", sThrNo, i);
        }

        if( SQLFreeStmt(sStmt[i], SQL_DROP) != SQL_SUCCESS )
        {
            printError(sEnv, sCon, sStmt[i], "SQLFreeStmt Error");
        }
        sStmt[i] = SQL_NULL_HSTMT;
    }

    disconnectDB(sEnv, sCon);

    printf("[%d]disconnected.\n", sThrNo);

    pthread_exit(NULL);

error:
    for( i = 0; i < LOG_FILE_CNT; i++)
    {
        if( sStmt[i] != SQL_NULL_HSTMT )
        {
            appendClose(sEnv, sCon, sStmt[i]);

            if( SQLFreeStmt(sStmt[i], SQL_DROP) != SQL_SUCCESS )
            {
                printError(sEnv, sCon, sStmt[i], "SQLFreeStmt Error");
            }
            sStmt[i] = SQL_NULL_HSTMT;
        }
    }

    if( sCon != SQL_NULL_HDBC )
    {
        disconnectDB(sEnv, sCon);
    }

    pthread_exit(NULL);
}


int initTables()
{
    SQLHENV     sEnv  = SQL_NULL_HENV;
    SQLHDBC     sCon  = SQL_NULL_HDBC;

    if( connectDB(&sEnv, &sCon) == RC_SUCCESS )
    {
        printf("connectDB success.\n");
    }
    else
    {
        printf("connectDB failure.\n");
        goto error;
    }

    if( createTables(sEnv, sCon) == RC_SUCCESS )
    {
        printf("createTables success.\n");
    }
    else
    {
        printf("createTables failure.\n");
        goto error;
    }
```

```
    disconnectDB(sEnv, sCon);

    return RC_SUCCESS;

error:

    if( sCon != SQL_NULL_HDBC )
    {
        disconnectDB(sEnv, sCon);
    }

    return RC_FAILURE;
}


int main()
{
    pthread_t sThread[MAX_THREAD_NUM];
    int       sNum[MAX_THREAD_NUM];
    int       sRC;
    int       i;

    initTables();

    //
    //eachThread has own ENV,DBC and STMT
    //
    for(i = 0; i < MAX_THREAD_NUM; i++)
    {
        sNum[i] = i;

        sRC = pthread_create(&sThread[i], NULL, (void *)eachThread, (void*)&sNum[i]);
        if ( sRC != RC_SUCCESS )
        {
            printf("Error in Thread create[%d] : %d\n", i, sRC);
            return RC_FAILURE;
        }
    }

    for(i = 0; i < MAX_THREAD_NUM; i++)
    {
        sRC = pthread_join(sThread[i], NULL);
        if( sRC != RC_SUCCESS )
        {
            printf("Error in Thread[%d] : %d\n", i, sRC);
            return RC_FAILURE;
        }
        printf("%d thread join\n", i+1);
    }

    return RC_SUCCESS;
}
```

Add the make code and run the executable file. Because the threads are used, the output order may be different. The results are as follows.

```
[mach@localhost cli]$ make sample8_multi_session_multi_table
gcc -c -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -I
/home/mach/machbase_home/include -I. -L//home/mach/machbase_home/include -osample8_multi_session_multi_table.
o sample8_multi_session_multi_table.c
gcc -m64 -mtune=k8 -L/home/mach/machbase_home/lib -osample8_multi_session_multi_table
sample8_multi_session_multi_table.o -lmachcli  -L/home/mach/machbase_home/lib -lm -lpthread -ldl -lrt -
rdynamic
[mach@localhost cli]$ ./sample8_multi_session_multi_table
connectDB success.
createTables success.
[0]connectDB success.
[1]connectDB success.
[2]connectDB success.
[1-0]appendOpen success.
[0-0]appendOpen success.
[2-0]appendOpen success.
[1-1]appendOpen success.
[2-1]appendOpen success.
[0-1]appendOpen success.
[1-2]appendOpen success.
[2-2]appendOpen success.
file open success - [1][suffle_data2.txt]
file open success - [2][suffle_data3.txt]
[0-2]appendOpen success.
file open success - [0][suffle_data1.txt]
.....................................................................................

[1-0]appendClose start...
..
[0-0]appendClose start...
append result success : 100000, failure : 0
[1-0]appendClose success
[1-1]appendClose start...
append result success : 100000, failure : 0
[1-1]appendClose success
[1-2]appendClose start...
append result success : 100000, failure : 0
[1-2]appendClose success
append result success : 100000, failure : 0
[0-0]appendClose success
[0-1]appendClose start...
.append result success : 100000, failure : 0
[0-1]appendClose success
[0-2]appendClose start...
append result success : 100000, failure : 0
[0-2]appendClose success

[2-0]appendClose start...
append result success : 100000, failure : 0
[2-0]appendClose success
[2-1]appendClose start...
append result success : 100000, failure : 0
[2-1]appendClose success
[2-2]appendClose start...
append result success : 100000, failure : 0
[2-2]appendClose success
[1]disconnected.
[2]disconnected.
[0]disconnected.
1 thread join
2 thread join
3 thread join
```

You can see the result through machsql as below.

```
[mach@localhost cli]$ machsql


================================================================
     Machbase Client Query Utility
     Release Version 3.5.0
     Copyright 2014, Machbase Inc. or its subsidiaries.
     All Rights Reserved.
================================================================
Machbase Server Addr (Default:127.0.0.1) :
Machbase User ID  (Default:SYS)
Machbase User Password : manager
MACH_CONNECT_MODE=INET, PORT=5656
Mach> select count(*) from table_f1;
count(*)
----------------------
300000
[1] Row Selected.
Mach> select count(*) from table_f2;
count(*)
----------------------
300000
[1] row(s) selected.
Mach> select count(*) from table_event;
count(*)
----------------------
300000
[1] row(s) selected.
```