CLI/ODBC

CLI ISO/IEC 9075-3:2003 .

CLI SQL , . CLI 1990 , C COBOL , .

ODBC(Open Database Connectivity) . ODBC API 3.52 ISO X/Open .

CLI

.

-
-

.

SQLAllocConnect	SQLDisconnect	SQLGetDescField	SQLPrepare
SQLAllocEnv	SQLDriverConnect	SQLGetDescRec	SQLPrimaryKeys
SQLAllocHandle	SQLExecDirect	SQLGetDiagRec	SQLStatistics
SQLAllocStmt	SQLExecute	SQLGetEnvAttr	SQLRowCount
SQLBindCol	SQLFetch	SQLGetFunctions	SQLSetConnectAttr
SQLBindParameter+	SQLFreeConnect	SQLGetInfo	SQLSetDescField
SQLColAttribute	SQLFreeEnv	SQLGetStmtAttr	SQLSetDescRec
SQLColumns	SQLFreeHandle	SQLGetTypeInfo	SQLSetEnvAttr
SQLConnect	SQLFreeStmt	SQLNativeSQL	SQLSetStmtAttr
SQLCopyDesc	SQLGetConnectAttr	SQLNumParams	SQLStatistics
SQLDescribeCol	SQLGetData	SQLNumResultCols	SQLTables

- CLI
  -
- CLI (APPEND)
  - Appe nd
  - Appe nd
  - Appe nd
  -
  - Trace
  - APP END
    - SQLAppendC
    - SQLAppendD
    - SQLAppendD
    - SQLAppendD
    - SQLAppendD
    - SQLAppendF
    - SQLAppendC
    - SQLAppendS
    - SQLSetConn
    - SQLSetStmtA
    - Error
-

CLI , .

DSN	. ODBC , CLI IP .
DBNAME	Machbase DB .
SERVER	Machbase IP .
NLS_USE	.(, .)
UID	
PWD	
PORT_NO	
PORT_DIR	Unix domain . (, .)
CONNTYPE	. 1: TCP/IP INET 2: Unix Domain
COMPRESS	Append . 0 . 0 Append .) COMPRESS=512 512 . .
SHOW_HIDDEN_COLS	(_arrival_time) select * . 0 , 1 .
CONNECTION_TIMEOUT	. 30 . 30 , .
ALTERNATIVE_SERVERS	cluster , . , , . , < >:< > ',' . ex) ALTERNATIVE_SERVERS=192.168.0.10:20320,192.168.0.11:20320;

CLI .

```
sprintf(connStr,"SERVER=127.0.0.1;COMPRESS=512;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 )) {
   ...
}
```

CLI (APPEND)

CLI Machbase Append .

4 , , , , .

Append

Machbase Append . , . , Append , , . Append . .

Append

SQLExecute SQLExecDirect() Machbase . , SQLAppendDataV2() . , . Append . SQLAppendFlush() , .

Append

Append . , , . , , Machbase . .

1. ,
2. SQLAppendFlush()
3. SQLAppendClose()

, 3 , I/O .

, . , SQLAppendOpen() aErrorCheckCount . 0 , . , 0 SQLAppendData() . 10 10 Append . , .

Trace

Append Trace DUMP_APPEND_ERROR 1 . , mach.trc . , , Machbase .

APPEND

SQLAppendOpen

```
SQLRETURN SQLAppendOpen(SQLHSTMT   aStatementHandle,
                        SQLCHAR   *aTableName,
                        SQLINTEGER aErrorCheckCount );
```

. .

1024 Statement . Statement SQLAppendOpen .

1. aStatementHandle : Append Statement .
2. aTableName : Append .
3. aErrorCheckCount : . 0 .

SQLAppendData (deprecated)

```
SQLRETURN  SQLAppendData(SQLHSTMT StatementHandle, void *aData[]);
```

.

- aData . Open .
- SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR .
  , SQL_SUCCESS_WITH_INFO .

- float, double, short, int, long long, char * .

- ipv4 5 (unsigned char) .
- 4 , 4 .
- , 127.0.0.1 5 **0x04, 0x7f, 0x00, 0x00, 0x01** .

```
// 4      (short(16), int(32), long(64), varchar)

testAppendIPFunc()
{
   short val1 = 0;
   int   val2 = 1;
   long long  val3 = 2;
   char *val4 = "my string";
   void *valueArray[4];

   valueArray[0] = (void *)&val1;
   valueArray[1] = (void *)&val2;
   valueArray[2] = (void *)&val3;
   valueArray[3] = (void *)val4;

   SQLAppendData(aStmt, valueArray);
}
```

datetime

- Machbase , 64 .
  mktime .
  Machbase = (1970 1 1 ()) * 1,000,000,000 + mili-second * 1,000,000 + micro-second * 1000 + nano-second;

```
// Date String "-- :: ::"

testAppendDateStrFunc(char *aDateString)
{
    int yy, int mm, int dd, int hh, int mi, int ss;
    unsigned long t1;
    void *valueArray[5];
    sscanf(aDateString, "%d-%d-%d %d:%d:%d %d:%d:%d",
        &yy, &mm, &dd, &hh, &mi, &ss, &mmm, &uuu, &nnn);
    sTm.tm_year = yy - 1900;
    sTm.tm_mon = mm - 1;
    sTm.tm_mday = dd;
    sTm.tm_hour = hh;
    sTm.tm_min = mi;
    sTm.tm_sec = ss;
    t1 = mktime(&sTm);
    t1 = t1 * 1000000000L;
    t1 = t1 + (mmm*1000000L) + (uuu*1000) + nnn;

    valueArray[4] = &t1;
    SQLAppendData(aStmt, valueArray);
}
```

SQLAppendDataByTime(deprecated)

```
SQLRETURN  SQLAppendDataByTime(SQLHSTMT StatementHandle, SQLBIGINT aTime, void *aData[]);
```

, DB _arrival_time .

, 1 .

- aTime _arrival_time time .
- aData .
- Open .

SQLAppendData() .

```
// 4       (short(16), int(32), long(64), varchar)

testAppendFuncWithTime()
{
   long long sTime = 1;
   short val1 = 0;
   int   val2 = 1;
   long long  val3 = 2;
   char *val4 = "my string";
   void *valueArray[4];

   valueArray[0] = (void *)&val1;
   valueArray[1] = (void *)&val2;
   valueArray[2] = (void *)&val3;
   valueArray[3] = (void *)val4;

   SQLAppendDataByTime(aStmt, sTime, valueArray);
}
```

SQLAppendDataV2

```
SQLRETURN  SQLAppendDataV2(SQLHSTMT StatementHandle, SQL_APPEND_PARAM *aData);
```

Machbase 2.0 Append , .

, 2.0 TEXT BINARY SQLAppendDataV2() .

- NULL
- VARCHAR
- IPv4, IPv6
- TEXT, BINARY

.

- aData SQL_APPEND_PARAM . Open .
- SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR . , SQL_SUCCESS_WITH_INFO .

V2 `SQL_APPEND_PARAM` , `machbase_sqlcli.h` .

```
typedef struct machAppendVarStruct
{
    unsigned int mLength;
    void *mData;
} machAppendVarStruct;

/* for IPv4, IPv6 as bin or string representation */
typedef struct machAppendIPSttypedef struct machbaseAppendVarStruct
{
    unsigned int mLength;
    void        *mData;
} machbaseAppendVarStruct;

/* for IPv4, IPv6 as bin or string representation */
typedef struct machbaseAppendIPStruct
{
    unsigned char   mLength; /* 0:null, 4:ipv4, 6:ipv6, 255:string representation */
    unsigned char   mAddr[16];
    char           *mAddrString;
} machbaseAppendIPStruct;

/* Date time*/
typedef struct machbaseAppendDateTimeStruct
{
    long long       mTime;
#if defined(SUPPORT_STRUCT_TM)
    struct tm       mTM;
#endif
    char           *mDateStr;
    char           *mFormatStr;
} machbaseAppendDateTimeStruct;

typedef union machbaseAppendParam
{
    short                       mShort;
    unsigned short              mUShort;
    int                         mInteger;
    unsigned int                mUInteger;
    long long                   mLong;
    unsigned long long          mULong;
    float                       mFloat;
    double                      mDouble;
    machbaseAppendIPStruct      mIP;
    machbaseAppendVarStruct     mVar;     /* for all varying type */
    machbaseAppendVarStruct     mVarchar; /* alias */
    machbaseAppendVarStruct     mText;    /* alias */
    machbaseAppendVarStruct     mBinary;  /* binary */
    machbaseAppendVarStruct     mBlob;    /* reserved alias */
    machbaseAppendVarStruct     mClob;    /* reserved alias */
    machbaseAppendDateTimeStruct mDateTime;
} machbaseAppendParam;

#define SQL_APPEND_PARAM machbaseAppendParam
```

machbaseAppendParam . . .

short, ushort, integer, uinteger, long, ulong, float, double . SQL_APPEND_PARAM .

	NULL	SQL_APPEND_PARAM
SHORT	SQL_APPEND_SHORT_NULL	mShort
USHORT	SQL_APPEND_USHORT_NULL	mUShort
INTEGER	SQL_APPEND_INTEGER_NULL	mInteger
UINTEGER	SQL_APPEND_UINTEGER_NULL	mUInteger
LONG	SQL_APPEND_LONG_NULL	mLong
ULONG	SQL_APPEND_ULONG_NULL	mULong
FLOAT	SQL_APPEND_FLOAT_NULL	mFloat
DOUBLE	SQL_APPEND_DOUBLE_NULL	mDouble

.

```
// Table Schema 8 ,  SHORT, USHORT, INTEGER, UINTEGER, LONG, ULONG, FLOAT, DOUBLE   .

void testAppendExampleFunc()
{
    SQL_APPEND_PARAM sParam[8];

    /* fixed column */
    sParam[0].mShort = SQL_APPEND_SHORT_NULL;
    sParam[1].mUShort = SQL_APPEND_USHORT_NULL;
    sParam[2].mInteger = SQL_APPEND_INTEGER_NULL;
    sParam[3].mUInteger = SQL_APPEND_UINTEGER_NULL;
    sParam[4].mLong = SQL_APPEND_LONG_NULL;
    sParam[5].mULong = SQL_APPEND_ULONG_NULL;
    sParam[6].mFloat = SQL_APPEND_FLOAT_NULL;
    sParam[7].mDouble = SQL_APPEND_DOUBLE_NULL;

    SQLAppendDataV2(Stmt, sParam);

    /* FIXED COLUMN Value */
    sParam[0].mShort = 2;
    sParam[1].mUShort = 3;
    sParam[2].mInteger = 4;
    sParam[3].mUInteger = 5;
    sParam[4].mLong = 6;
    sParam[5].mULong = 7;
    sParam[6].mFloat = 8.4;
    sParam[7].mDouble = 10.9;

    SQLAppendDataV2(Stmt, sParam);
}
```

DATETIME . .

SQL_APPEND_PARAM mDateTime . mDateTime mTime 64 .

```
typedef struct machbaseAppendDateTimeStruct
{
    long long       mTime;
#if defined(SUPPORT_STRUCT_TM)
    struct tm       mTM;
#endif
    char           *mDateStr;
    char           *mFormatStr;
} machbaseAppendDateTimeStruct;
```

SQL_APPEND_DATETIME_NOW	.
SQL_APPEND_DATETIME_STRUCT_TM	mDateTime struct tm mTM , .
SQL_APPEND_DATETIME_STRING	mDateTime , . mDateStr : mFormatStr :
SQL_APPEND_DATETIME_NULL	NULL .
64	datetime . 1970 1 1 . , 10 (1,000,000,000) , 1970 1 1 0 0 1 .(GMT)

```
//        .  DATETIME   .
void testAppendDateTimeFunc()
{
    SQL_mach_PARAM sParam[1];
    /* NULL  */
    sParam[0].mDateTime.mTime   = SQL_APPEND_DATETIME_NULL;
    SQLAppendDataV2(Stmt, sParam);

    /*    */
    sParam[0].mDateTime.mTime      = SQL_APPEND_DATETIME_NOW;
    SQLAppendDataV2(Stmt, sParam);

    /*    :1970.1.1     */
    sParam[0].mDateTime.mTime      = 1234;
    SQLAppendDataV2(Stmt, sParam);

    /*      */
    sParam[0].mDateTime.mTime      = SQL_APPEND_DATETIME_STRING;
    sParam[0].mDateTime.mDateStr   = "23/May/2014:17:41:28";
    sParam[0].mDateTime.mFormatStr = "DD/MON/YYYY:HH24:MI:SS";
    SQLAppendDataV2(Stmt, sParam);

    /*  struct tm    */
    sParam[0].mDateTime.mTime      = SQL_APPEND_DATETIME_STRUCT_TM;
    sParam[0].mDateTime.mTM.tm_year = 2000 - 1900;
    sParam[0].mDateTime.mTM.tm_mon  =  11;
    sParam[0].mDateTime.mTM.tm_mday  = 31;
    SQLAppendDataV2(Stmt, sParam);
}
```

IPv4 IPv6 . . SQL_APPEND_PARAM mLength .

```
/* for IPv4, IPv6 as bin or string representation */
typedef struct machbaseAppendIPStruct
{
    unsigned char   mLength; /* 0:null, 4:ipv4, 6:ipv6, 255:string representation */
    unsigned char   mAddr[16];
    char           *mAddrString;
} machbaseAppendIPStruct;
```

(mLength)	
SQL_APPEND_IP_NULL	NULL
SQL_APPEND_IP_IPV4	mAddr IPv4
SQL_APPEND_IP_IPV6	mAddr IPv6
SQL_APPEND_IP_STRING	mAddrString .

.

```
void testAppendIPFunc()
{
    SQL_APPEND_PARAM sParam[1];
    /* NULL */
    sParam[0].mIP.mLength  = SQL_APPEND_IP_NULL;
    SQLAppendDataV2(Stmt, sParam);

    /*    */
    sParam[0].mIP.mLength  = SQL_APPEND_IP_IPV4;
    sParam[0].mIP.mAddr[0] = 127;
    sParam[0].mIP.mAddr[1] = 0;
    sParam[0].mIP.mAddr[2] = 0;
    sParam[0].mIP.mAddr[3] = 1;
    SQLAppendDataV2(Stmt, sParam);

    /* IPv4 from binary */
    sParam[0].mIP.mLength  = SQL_APPEND_IP_IPV4;
    *(in_addr_t *)(sParam[0].mIP.mAddr) = inet_addr("192.168.0.1");
    SQLAppendDataV2(Stmt, sParam);

    /* IPv4 : ipv4 from string */
    sParam[0].mIP.mLength     = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = "203.212.222.111";
    SQLAppendDataV2(Stmt, sParam);

    /* IPv4 : ipv4 from invalid string */
    sParam[0].mIP.mLength     = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = "ip address is not valid";
    SQLAppendDataV2(Stmt, sParam);                          // invalid IP value

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

    sParam[0].mIP.mLength     = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = "::127.0.0.1";
    SQLAppendDataV2(Stmt, sParam);

    sParam[0].mIP.mLength     = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = "FFFF:FFFF:1111:2222:3333:4444:7733:2123";
    SQLAppendDataV2(Stmt, sParam);
}
```

IP (STRING) SQLAppendDataV2 mLength 4 6 .
SQLAppendDataV2() , mLength SQL_APPEND_IP_STRING .

( )

VARCHAR TEXT , BLOB CLOB . VARCHAR , . strlen() , V2 . , , . , .

```
typedef struct machAppendVarStruct
{
    unsigned int mLength;
    void *mData;
} machAppendVarStruct;
```

mLength , mData . mLength . SQLAppendDataV2() **SQL_SUCCESS_WITH_INFO** , . SQLError() .

	NULL	**SQL_APPEND_PARAM (mVar)**
VARCHAR	SQL_APPEND_VARCHAR_NULL	mVarchar
TEXT	SQL_APPEND_TEXT_NULL	mText
BINARY	SQL_APPEND_BINARY_NULL	mBinary
BLOB	SQL_APPEND_BLOB_NULL	mBlob
CLOB	SQL_APPEND_CLOB_NULL	mClob

. VARCHAR .

```
CREATE TABLE ttt (name VARCHAR(10));
```

```
void testAppendVarcharFunc()
{
    SQL_mach_PARAM sParam[1];

    /*  VARCHAR : NULL */
    sParam[0].mVarchar.mLength = SQL_APPEND_VARCHAR_NULL
    SQLAppendDataV2(Stmt, sParam); /* OK */

    /*  VARCHAR : string */
    strcpy(sVarchar, "MY VARCHAR");
    sParam[0].mVarchar.mLength = strlen(sVarchar);
    sParam[0].mVarchar.mData   = sVarchar;
    SQLAppendDataV2(Stmt, sParam); /* OK */

    /*  VARCHAR : Truncation! */
    strcpy(sVarchar, "MY VARCHAR9"); /* Truncation! */
    sParam[0].mVarchar.mLength = strlen(sVarchar);
    sParam[0].mVarchar.mData   = sVarchar;
    SQLAppendDataV2(Stmt, sParam);  /* SQL_SUCCESS_WITH_INFO */
}
```

Text .

```
CREATE TABLE ttt (doc TEXT);
```

```
void testAppendFunc()
{
    SQL_mach_PARAM sParam[1];

    /*  VARCHAR : NULL */
    sParam[0].mText.mLength = SQL_APPEND_TEXT_NULL
    SQLAppendDataV2(Stmt, sParam); /* OK */

    /*  VARCHAR : string */
    strcpy(sText, "This is the sample document for tutorial.");
    sParam[0].mVar.mLength = strlen(sText);
    sParam[0].mVar.mData   = sText;
    SQLAppendDataV2(Stmt, sParam); /* OK */
}
```

SQLAppendDataByTimeV2

```
SQLRETURN  SQLAppendDataByTimeV2(SQLHSTMT StatementHandle, SQLBIGINT aTime, SQL_APPEND_PARAM  *aData);
```

, DB _arrival_time . , 1 .

- aTime _arrival_time time. *1970 1 1 nano second* . .
- aData . Open .

SQLAppendDataV2() .

SQLAppendFlush

```
SQLRETURN SQLAppendFlush(SQLHSTMT StatementHandle);
```

Machbase .

SQLAppendClose

```
SQLRETURN SQLAppendClose(SQLHSTMT   aStmtHandle,
                         SQLBIGINT* aSuccessCount,
                         SQLBIGINT* aFailureCount);
```

. .

- aSuccessCount : Append .
- aFailureCount : Append .

SQLAppendSetErrorCallback

```
SQLRETURN SQLAppendSetErrorCallback(SQLHSTMT aStmtHandle, SQLAppendErrorCallback aFunc);
```

Append . , .

- aStmtHandle : Statement .
- aFunc : Append .

SQLAppendErrorCallback .

```
typedef void (*SQLAppendErrorCallback)(SQLHSTMT aStmtHandle,
                                       SQLINTEGER aErrorCode,
                                       SQLPOINTER aErrorMessage,
                                           SQLLEN aErrorBufLen,
                                       SQLPOINTER aRowBuf,
                                           SQLLEN aRowBufLen);
```

- aStatementHandle : Statement
- aErrorCode : 32
- aErrorMessage :
- aErrorBufLen : aErrorMessage
- aRowBuf :
- aRowBufLen : aRowBuf

(dumpError)

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

    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);
}


......

    if( SQLAppendOpen(m_IStmt, TableName, aErrorCheckCount) != SQL_SUCCESS )
    {
        fprintf(stdout, "SQLAppendOpen error\n");
        exit(-1);
    }
    //  .
    assert(SQLAppendSetErrorCallback(m_IStmt, dumpError) == SQL_SUCCESS);

    doAppend(sMaxAppend);

    if( SQLAppendClose(m_IStmt, &sSuccessCount, &sFailureCount) != SQL_SUCCESS )
    {
        fprintf(stdout, "SQLAppendClose error\n");
        exit(-1);
    }
}
```

SQLSetConnectAppendFlush

```
SQLRETURN SQL_API SQLSetConnectAppendFlush(SQLHDBC hdbc, SQLINTEGER option)
```

Append SQLAppendFlush . Append . 100ms (1) .

.

- hdbc : DB connection handle.
- option : 0 auto flush off, 0 auto flush on .

hdbc .

SQLSetStmtAppendInterval

```
SQLRETURN SQL_API SQLSetStmtAppendInterval(SQLHSTMT hstmt, SQLINTEGER fValue)
```

SQLSetConnectAppendFlush flush , statement flush flush .

.

- hstmt : flush statement handle.
- fValue : flush . **0 flush ms**. 100ms flush 100 . flush . **1000** .

flush .

Error

Append . CLI return SQL_SUCCESS .

```
    SQLINTEGER errNo;
    int msgLength;
    char sqlState[6];
    char errMsg[1024];

    if (SQL_SUCCESS == SQLError ( env, con, stmt, (SQLCHAR *)sqlState, &errNo,
                                  (SQLCHAR *)errMsg, 1024, &msgLength ))
    {
        //error code 5  .
        printf("ERROR-%05d: %s\n", errNo, errMsg);
    }
```

Append .

	message	
SQLAppendOpen	statement is already opened.	SQLAppendOpen .
	Failed to close stream protocol.	.
	Failed to read protocol.	.
	cannot read column meta.	column meta
	cannot allocate memory.	.
	cannot allocate compress memory.	.
	invalid return after reading column meta.	return .
SQLAppendData	statement is not opened.	AppendOpen AppendData call.
	column() truncated :	varchar .
	Failed to add binary.	.
SQLAppendClose	statement is not opened.	AppendOpen .
	Failed to close stream protocol.	.
	Failed to close buffer protocol.	.
	cannot read column meta.	column meta .
	invalid return after reading column meta.	return .

SQLAppendFlush	statement is not opened.	AppendOpen
	Failed to close stream protocol.	.
SQLSetErrorCallback	statement is not opened.	AppendOpen .
	Protocol Error (not APPEND_DATA_PROTOCOL)	APPEND_DATA_PROTOCOL .
SQLAppendDataV2	Invalid date format or date string.	datetime .
	statement is not opened.	AppendOpen
	column() truncated :	binary .
	column() truncated :	varchar, text .
	Failed to add stream.	.
	IP address length is invalid.	IPv4, IPv6 mLength .
	IP string is invalid.	IPv4, IPv6 .
	Unknown data type has been specified.	Machbase data type .

Machbase SQLAppend Log/Tag , Lookup volatile update SQLAppend .

Machbase 5.5 . (.)

SQLSetStmtAttr() Attribute SQL_ATTR_PARAM_BIND_TYPE param SQL_PARAM_BIND_BY_COLUMN . , . SQLBindParameter() .

.

Column A (parameter A)		**Column B (parameter B)**		**Column C (parameter C)**	
Value_Array	Indicator/ length array	Value_Array	Indicator/ length array	Value_Array	Indicator/ length array

.

```
#define DESC_LEN 51
#define ARRAY_SIZE 10
SQLCHAR * Statement = "INSERT INTO Parts (PartID, Description, Price) VALUES (?, ?, ?)";

/*    */
SQLUINTEGER PartIDArray[ARRAY_SIZE];
SQLCHAR DescArray[ARRAY_SIZE][DESC_LEN];
SQLREAL PriceArray[ARRAY_SIZE];
/*     */
SQLINTEGER PartIDIndArray[ARRAY_SIZE], DescLenOrIndArray[ARRAY_SIZE], PriceIndArray[ARRAY_SIZE];
SQLUSMALLINT i, ParamStatusArray[ARRAY_SIZE];
SQLUINTEGER ParamsProcessed;

// Set the SQL_ATTR_PARAM_BIND_TYPE statement attribute to use
// column-wise binding.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_BIND_TYPE, SQL_PARAM_BIND_BY_COLUMN, 0);
// Specify the number of elements in each parameter array.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMSET_SIZE, ARRAY_SIZE, 0);
// Specify an array in which to return the status of each set of
// parameters.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_STATUS_PTR, ParamStatusArray, 0);
// Specify an SQLUINTEGER value in which to return the number of sets of
// parameters processed.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &ParamsProcessed, 0);
// Bind the parameters in column-wise fashion.
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG, SQL_INTEGER, 5, 0,
    PartIDArray, 0, PartIDIndArray);
SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, DESC_LEN - 1, 0,
    DescArray, DESC_LEN, DescLenOrIndArray);
SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_REAL, 7, 0,
    PriceArray, 0, PriceIndArray);
```