IBM DB2® Universal Database™
DB2 Problem Determination Tutorial Series

Application Problem Determination

Table of Contents

Application Problem Determination

About this tutorial

Introduction

This tutorial assists you in diagnosing common problems with application development and execution involving DB2 databases. To understand concepts in this tutorial, you should have a basic knowledge of database fundamentals, SQL error messages, and application interfaces - specifically, ODBC, JDBC, and OLEDB/ADO. You will also need to understand basic concepts of SQL and data manipulation.

This tutorial focuses on problem determination for building and executing the following kinds of applications:

- ODBC Applications
- JDBC Applications
- OLEDB/ADO Applications
- Stored procedures

Setup

This tutorial shows examples installed on the Windows® operating system, but all concepts and exercises can be completed on any DB2 distributed system.

In order to work through the examples in this tutorial, you should have completed the following tasks:

- Installed DB2 and the samples
- Created an instance and two IDs to access (and create if necessary) a DB2 test database, with permissions to create objects within the database. We will use the DB2 sample database throughout the tutorial.
- Installed compilers for C/C++, Java™ and Microsoft® Visual Basic. These are necessary only if you wish to complete all of the examples given in this tutorial.

About the author

Murray Chislett has an honours Bachelor of Computer Science degree and is a senior member of the DB2 technical support team at the IBM Toronto Laboratory. A former developer in database technology at IBM, he supports customers worldwide with expertise in the DB2 UDB engine and application development. Murray is now primarily responsible for IBM Data Management Premier Services customers.

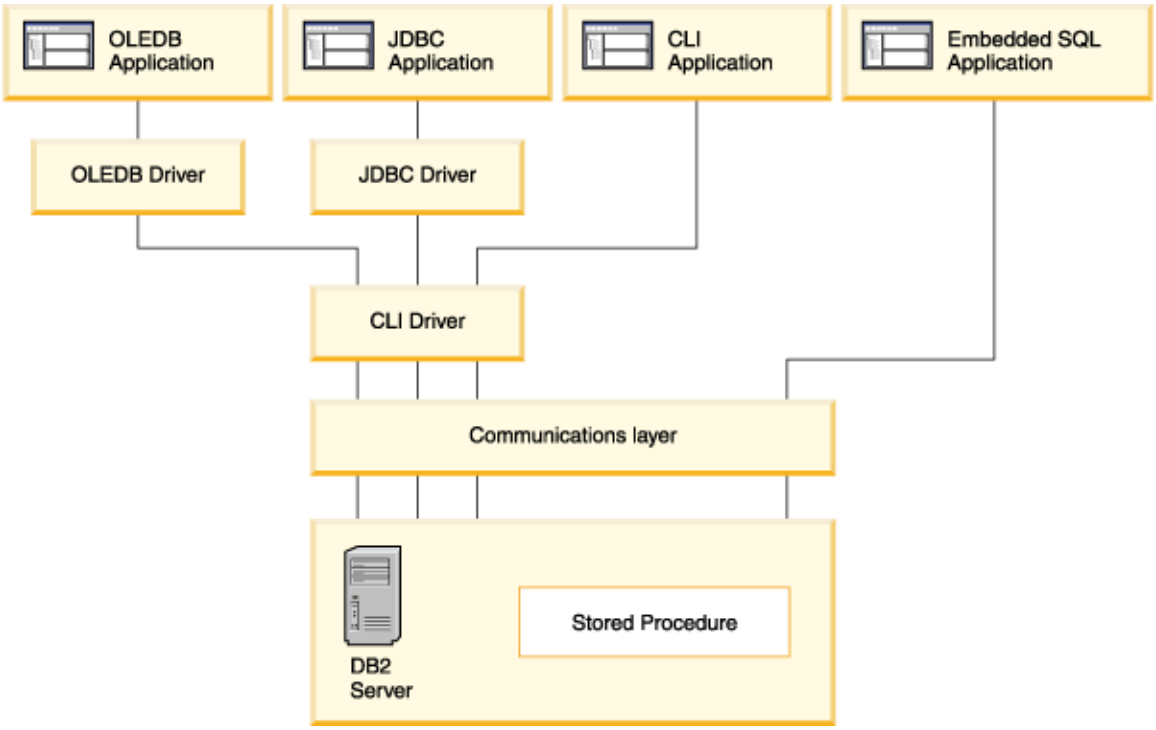
You can reach Murray by locating his e-mail address in the IBM Global Directory at http://www.ibm.com/contact/employees/us .

Building applications

How applications work

Applications or stored procedures are written in a variety of languages using a set of standard APIs to manipulate database data.

It is important to know the layers involved in DB2 applications in order to diagnose issues. The DB2 Call Level Interface (CLI), which is equivalent in functionality to ODBC, is often used underneath other drivers, so it may be necessary to debug different layers to determine problems. The figure below shows the typical application flow:



Errors that occur during compilation are often due to programming errors or environments that are not correctly set up for application creation. This tutorial will look at some examples of compilation and linking errors.

By following the *DB2 Application Development* manuals as well as the specific documentation related to your programming interface, you can avoid most errors involved in creating and executing DB2 applications. DB2 product manuals for application development can be found in both HTML and PDF formats at
http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v7pubs.d2w/en_main .

Compilation errors

In this section, we will look at an example of a compilation error in a CLI application written in C/C++. Here is the sample program that we will use:

Sample.c (ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/tutorials/Sample.c)

Compile Sample.C with this command:

```
cl sample.c -I"%DB2PATH%\include" -link"%DB2PATH%\lib\db2cli.lib"
```

(You might have to modify the command if your compiler requires different options. You could also choose to create a makefile to compile the sample.c program) If you compile the program you will get the following error:

```
sample.c(22) : error C2198: 'SQLAllocHandle' : too few actual parameters
```

Line 22 of Sample.C contains the following code:

```
sqlrc = SQLAllocHandle( SQL_HANDLE_ENV, NULL ) ;
```

Checking the CLI documentation for SQLAllocHandle, you see that the call is incorrect. You can find the SQLAllocHandle call documented at
http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/document.d2w/report?fn=db2v7l0sqll1401.htm#HDRFNAH1 .
This is a simple error common to all types of programming. Errors such as this, received at compile time, can usually be resolved by checking the related documentation and correcting the code as necessary.

Correct the error by modifying line 22 of Sample.c as follows:

```
sqlrc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, NULL ) ;
```

Once modified, the program should compile and execute successfully. If not, check the DB2 Application Development Guide and your compiler documentation to ensure that your environment is set up correctly. Success means your environment is set up correctly to run CLI applications.

Linking errors

Compiler and linker errors for DB2 applications are often the same as those you encounter when creating any type of application. The linking step sometimes results in difficulty creating the executable.

Ensure your environment is correctly installed and you are using the proper DB2 libraries contained in C:\Program Files\SQLLIB\LIB or \BIN. This is the best way to avoid compilation/linking problems. You should check the *DB2 Application Development* guides relating

to your programming interface and your compiler specific documentation. They will guide you to set up the correct environment.

Here are some of the common items that relate to having your environment setup correctly:

- Ensure makefiles contain the correct compiler information
- Set all compiler-related variables for include files and libraries
- Ensure any command line arguments point to the correct DB2 directories

Here are some of the common mistakes that can lead to linker errors:

- Improper API/function calls
- Multiple versions of linked libraries in the LIBPATH
- API/function definitions not matching the code contained in the library files

You can look at the samples included with DB2 as they also include scripts and instructions to compile and link them.

ODBC applications

Introduction

When you receive unexpected behaviour from DB2 applications, tracing is often the best way to determine the root cause. Remember that many of the supported programming interfaces for DB2 will eventually go through the CLI application layer. These include JDBC and OLEDB. So, even if you are not writing or supporting CLI or ODBC applications, it may be important to understand how it works and its tracing methods.

In this tutorial, we will look at tracing for CLI, JDBC and OLEDB applications.

Tracing

ODBC applications use the DB2 CLI interface to access DB2. The DB2 CLI library is also an ODBC library. When diagnosing ODBC applications it is often easiest to determine the problem by using an ODBC trace or DB2 CLI trace. If you are using an ODBC driver manager, it will likely provide the capability to take an ODBC trace. Consult your driver manager documentation on how to enable ODBC tracing. DB2 CLI traces are DB2-specific and will often contain more information than a generic ODBC trace. Both traces are usually quite similar, listing entry and exit points for all CLI calls from an application; including any parameters and return codes to those calls.

The CLI trace is enabled by adding the following entries to the db2cli.ini file, which is located on Windows systems in the C:\Program Files\SQLLIB\ directory:

```
TRACEFLUSH=1
TRACEPATHNAME=C:\TEMP
TRACE=ON
```

To find out what these parameters mean, read the *DB2 Call Level Interface Guide and Reference* at http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/document.d2w/report?fn=db2v7l0clikeyw.htm#HDRHDCONFG .

Note that there are lots of keywords that can be added to the db2cli.ini file that can affect application behavior. These keywords can resolve or be the cause of application problems. There are also some keywords that are not covered in the CLI documentation. Those are only available from DB2 Service and Support. If you have keywords in your db2cli.ini file that are not documented, it is likely that they were a recommendation from the DB2 support team.

Enabling CLI traces

Turn on your CLI trace, as shown in the previous page. Here is an example of the entries you might see in your db2cli.ini file:

```
; Comment lines start with a semi-colon.

[tstcli1x]
uid=userid
pwd=password
autocommit=0
TableType="'TABLE','VIEW','SYSTEM TABLE'"

[tstcli2x]
; Assuming dbalias2 is a database in DB2 for MVS.
SchemaList="'OWNER1','OWNER2',CURRENT SQLID"

[MyVeryLongDBALIASName]
dbalias=dbalias3
SysSchema=MYSCHEMA

[DWCTRLDB]
DBALIAS=DWCTRLDB

[COMMON]
trace=1
traceflush=1
Tracefilename=C:\temp\cli.trc
```

When you use the trace facility to diagnose application issues, keep in mind that it does have an impact o application performance and that it affects all applications, not only your test application. Remember to turn it off after the problem has been identified.

Application errors

Here we have a sample application that has a problem upon execution.

Sample1.c (ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/tutorials/Sample1.c)

When you run the application, it uses the current user ID and password to connect with a database named sample.

You will need to modify the program to change the database name to the database that you are actually using for these exercises.

Compile Sample1.c with the following command. (You many need to alter this command to accommoda you specific C/C++ compiler.)

```
cl sample1.c -I"%DB2PATH%\include"  -link"%DB2PATH%\lib\db2cli.lib"
```

Execute the program against a database by calling the executable created in the above compile step. This will result in the following error:

```
Error Fetching.
```

Examine Sample1.C to find where the application would return this error. You will find in order for the application to return the above error, the SQLFetch call must have returned an SQL_ERROR. Since the application didn't request the error info, we need more information to find out the cause.

You can examine the CLI trace, since it was enabled in a previous section. It will be the file you specified in the TRACEFILENAME keyword in your db2cli.ini file. Look at the portion of the CLI trace that includes the SQLFetch call:

```
SQLFetch( hStmt=1:1 )
    ---> Time elapsed - +2.200000E-004 seconds

SQLFetch( )
    <--- SQL_ERROR   Time elapsed - +1.238000E-003
    seconds
```

This shows the error being returned, but no error detail.

ODBC error detail is returned only when an application requests that information. Taking a CLI trace may reveal more information than the application returns. In the above case, you see that it does not. Next we will look at ensuring that the application returns enough information to permit problem diagnosis.

Proper error handling

The trace has not revealed any further info, so we will now add the calls necessary to request the error information.

Often applications are difficult to troubleshoot because not enough error information is provided by the application. If you were to rewrite the sample program to gather the proper error information, you would need to add SQLGetDiagRec calls to obtain the information from the server. The application has been rewritten to add a new function to gather DB2 error information should any of the calls fail. Look at Sample2.c to see how this was done.

Sample2.c (ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/tutorials/Sample2.c)

Delete the cli.trc file created from running sample1 in the previous panel.

Now, compile sample2.c as you did sample1.c above and execute the new program created.

You will see the following error returned:

```
Error Fetching.
```

```
  SQLSTATE          = HY010
  Native Error Code = -99999
[IBM][CLI Driver] CLI0125E  Function sequence error.
SQLSTATE=HY010
-------------------------

 Application Complete.
```

You can also see the error if you check the new cli.trc file created from running sample2.c. It will be in the same location as before. Examine just the portion of the file related to the SQLFetch call that returned an error:

```
SQLFetch( hStmt=1:1 )
    ---> Time elapsed - +2.130000E-004 seconds

SQLFetch( )
    <--- SQL_ERROR   Time elapsed - +2.769000E-003
    seconds

SQLGetDiagRec( fHandleType=SQL_HANDLE_STMT,
hHandle=1:1, iRecNumber=1, pszSqlState=&0012ff10,
pfNativeError=&0012fb08, pszErrorMsg=&0012fb0c,
cbErrorMsgMax=1025, pcbErrorMsg=&0012ff1c )
    ---> Time elapsed - +5.350000E-004 seconds

SQLGetDiagRec( pszSqlState="HY010",
pfNativeError=-99999, pszErrorMsg="[IBM][CLI Driver]
CLI0125E  Function sequence error. SQLSTATE=HY010" -
X"5B49424D5D5B434C492044726976657 25D20434C4930313235
45202046756E6374696F6E2073657175656E6365206572726F72
2E2053514C53544154453D4859303130", pcbErrorMsg=67 )
    <--- SQL_SUCCESS   Time elapsed - +3.274000E-003
    seconds

SQLGetDiagRec( fHandleType=SQL_HANDLE_STMT,
hHandle=1:1, iRecNumber=2, pszSqlState=&0012ff10,
pfNativeError=&0012fb08, pszErrorMsg=&0012fb0c,
cbErrorMsgMax=1025, pcbErrorMsg=&0012ff1c )
    ---> Time elapsed - +1.142700E-002 seconds

SQLGetDiagRec( )
    <--- SQL_NO_DATA_FOUND   Time elapsed -
    +2.361000E-003 seconds
```

By checking the ODBC Standard or DB2 CLI documentation, you will learn that a function sequence error is returned from an SQLFetch call if you do not call SQLExecute first. The statement needs to be executed before you can fetch any data it may return.

Attempt to correct this programming error in sample2.c by adding an SQLExecute call. Recompile and re-execute the program to ensure you have made the correct changes.

JDBC applications

Tracing

The JDBC interface accesses DB2 by making use of the DB2 CLI interface. Debugging JDBC applications will often involve using multiple trace facilities.

For solving JDBC issues, enable JDBC tracing by adding JDBC trace keywords to your db2cli.ini file. The keywords added are in bold font. If you were enabling JDBC tracing, your file would look similar to this:

```
; Comment lines start with a semi-colon.

[tstcli1x]
uid=userid
pwd=password
autocommit=0
TableType="'TABLE','VIEW','SYSTEM TABLE'"

[tstcli2x]
; Assuming dbalias2 is a database in DB2 for MVS.
SchemaList="'OWNER1','OWNER2',CURRENT SQLID"

[MyVeryLongDBALIASName]
dbalias=dbalias3
SysSchema=MYSCHEMA

[DWCTRLDB]
DBALIAS=DWCTRLDB

[COMMON]
trace=1
traceflush=1
Tracefilename=C:\temp\cli.trc

JDBCTRACE=1
JDBCFILENAME=C:\TEMP\JDBC.TRC
JDBCTRACE=FLUSH
```

You do not need to enable JDBC tracing at this time. This is an example of how to enable JDBC tracing in your environment.

Debugging

In this next exercise, executing the Java sample program and SQL statements in your environment might not return exactly the same errors as described here. You should follow through this example and try to find the cause of the errors from the information provided.

Executing sample3.java in your environment will have unkown results. Suppose you have the following JDBC program:

sample3.java (ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/tutorials/sample3.java)

Your users have described the following problems with this program. When the program is executed with USER1 logged in, it runs successfully. When it is executed with USER2 logged in, it fails with the error stating that the object USER1.TABLE is not defined.

You can see from the code that the application is attempting to execute the following SQL statement:

```
Insert into TABLE values ('sample3')
```

If you receive errors while attempting to execute SQL from within an application, it is often quite valuable to attempt the same SQL from another interface, for example the DB2 command line.

In this example, if you issue the statement exactly as it is in the application, from the command line, it is successful. It is not understood at this time why the application is looking for the object USER2.TABLE when it is not specified in the application. To continue your investigation, you would take a JDBC trace of the failing application. Below is a sample section of a JDBC trace from the failing application. Try to locate any reason that would cause the above problem:

```
jdbc.app.DB2Connection -> prepareStatement( insert
into Test values ('sample3') ) (2002-11-25 00:24:22.53)
| jdbc.app.DB2PreparedStatement -> DB2Statement( con,
1003, 1007 ) (2002-11-25 00:24:22.53)
| | jdbc.app.DB2PreparedStatement ->
checkResultSetType( 1003, 1007 )
(2002-11-25 00:24:22.53)
| | jdbc.app.DB2PreparedStatement <-
checkResultSetType() [Time Elapsed = 0.0]
(2002-11-25 00:24:22.53)
| | 10: Peak statements = 1
| | 10: Statement Handle = 1:1
| jdbc.app.DB2PreparedStatement <- DB2Statement()
[Time Elapsed = 0.01] (2002-11-25 00:24:22.54)
| jdbc.app.DB2PreparedStatement ->
DB2PreparedStatement( "insert into Test values
('sample3')", con, 1003, 1007 )
(2002-11-25 00:24:22.54)
| | jdbc.app.DB2PreparedStatement ->
getStatementType( insert into Test values ('sample3')
) (2002-11-25 00:24:22.54)
| | jdbc.app.DB2PreparedStatement <- getStatementType()
returns STMT_TYPE_OTHER (26) [Time Elapsed = 0.0]
(2002-11-25 00:24:22.54)
| | 10: maxNumParam = 0
```



```
| jdbc.app.DB2PreparedStatement <-
DB2PreparedStatement() [Time Elapsed = 0.01]
(2002-11-25 00:24:22.55)
jdbc.app.DB2Connection <- prepareStatement()
[Time Elapsed = 0.02] (2002-11-25 00:24:22.55)

jdbc.app.DB2PreparedStatement -> execute()
(2002-11-25 00:24:22.55)
| 10: Statement Handle = 1:1
| jdbc.app.DB2PreparedStatement -> execute2()
(2002-11-25 00:24:22.55)
| | 10: Statement Handle = 1:1
| | 10: SQLExecute()- returnCode = -1
| | jdbc.DB2Exception -> DB2Exception()
(2002-11-25 00:24:22.56)
| | | 10: SQLError = [IBM][CLI Driver][DB2/NT]
SQL0204N  "USER1.TEST" is an undefined name.
SQLSTATE=42704

| | |     SQLState = 42S02
| | |     SQLNativeCode = -204
| | |     LineNumber = 0
| | |     SQLerrmc = USER1.TEST
| | jdbc.DB2Exception <- DB2Exception()
[Time Elapsed = 0.0] (2002-11-25 00:24:22.56)
| | 20: executed (Exit) = false
| jdbc.app.DB2PreparedStatement <- execute2()
[Time Elapsed = 0.02] (2002-11-25 00:24:22.57)
jdbc.app.DB2PreparedStatement <- execute() returns
false [Time Elapsed = 0.02] (2002-11-25 00:24:22.57)
```

In this trace, the only information that might be relevant is the error message that the application itself returned. You still need more diagnostic information to discover the root of this problem.

As stated earlier, JDBC applications use the CLI interface as well, so we may need to examine both layers to debug problems in JDBC applications. The next step would be to execute the program while CLI tracing is enabled. Next a CLI trace of the failing sample3.java has been included.

Cli.trc (ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/tutorials/Cli.trc)

Examine the SQLPrepare and SQLExecute call from the above trace:

```
SQLPrepareW( hStmt=1:1, pszSqlStr="insert into Test values ('sample3')" -
 X"69006E0073006500720074002000690006E0074006F00200054006500730074002000760
 061006C00750065007300200028002700730061006D0070006C006500330027002900",
 cbSqlStr=35 )
    ---> Time elapsed - +5.211000E-003 seconds
( StmtOut="insert into Test values ('sample3')" )
```



```
SQLPrepareW( )
    <--- SQL_SUCCESS   Time elapsed - +2.277000E-003 seconds

SQLNumParams( hStmt=1:1, pcPar=&0006fa04 )
    ---> Time elapsed - +2.160000E-004 seconds

SQLNumParams( pcPar=0 )
    <--- SQL_SUCCESS   Time elapsed - +1.136000E-003 seconds

SQLExecute( hStmt=1:1 )
    ---> Time elapsed - +3.290000E-003 seconds
    sqlccsend( ulBytes - 256 )
    sqlccsend( Handle - 152786688 )
    sqlccsend( ) - rc - 0, time elapsed - +1.083000E-003
    sqlccrecv( )
    sqlccrecv( ulBytes - 163 ) - rc - 0, time elapsed - +2.020000E-004

SQLExecute( )
    <--- SQL_ERROR   Time elapsed - +5.108000E-003 seconds

SQLGetDiagFieldW( fHandleType=SQL_HANDLE_STMT, hHandle=1:1, iRecNumber=1,
fDiagIdentifier=Unknown value 2467, pDiagInfo=&0006f8ec, cbDiagInfoMax=140,
pcbDiagInfo=&0006f99a )
    ---> Time elapsed - +6.290000E-004 seconds

SQLGetDiagFieldW( pDiagInfo=5439573, pcbDiagInfo=20 )
    <--- SQL_SUCCESS   Time elapsed - +3.003000E-003 seconds

SQLGetDiagFieldW( fHandleType=SQL_HANDLE_STMT, hHandle=1:1, iRecNumber=1,
fDiagIdentifier=Unknown value 2461, pDiagInfo=&0006f988, cbDiagInfoMax=4,
pcbDiagInfo=&0006f99a )
    ---> Time elapsed - +2.130000E-004 seconds

SQLGetDiagFieldW( pDiagInfo=0, pcbDiagInfo=20 )
    <--- SQL_SUCCESS   Time elapsed - +2.255000E-003 seconds

SQLErrorW( hEnv=0:0, hDbc=0:0, hStmt=1:1, pszSqlState=&0006f978,
pfNativeError=&0006f984, pszErrorMsg=&0006f0e8, cbErrorMsgMax=1024,
pcbErrorMsg=&0006f98e )
    ---> Time elapsed - +2.100000E-004 seconds

SQLErrorW( pszSqlState="42S02 4ÿ  " - X"340032005300300032000000034FFFFFF0
0000000", pfNativeError=-204, pszErrorMsg="[IBM][CLI Driver][DB2/NT]
SQL0204N  "USER1.TEST" is an undefined name.  SQLSTATE=42704

" -  -
X"5B00490042004D005D005B0043004C00490020004400720069007600650072005D005B0
04400420032002F004E0054005D002000530051004C00300032003000340040E0020002000
22005500530045005200310020E00540045005300540022002000690073002000610006E002
00075006E00640065006600660069006E006500640020006E0061006D0065002E002000200053
0051004C00530054004100540045003D00340032003700300034000D000A00",
pcbErrorMsg=88 )
    <--- SQL_SUCCESS   Time elapsed - +3.066000E-003 seconds

SQLGetDiagFieldW( fHandleType=SQL_HANDLE_STMT, hHandle=1:1, iRecNumber=1,
fDiagIdentifier=Unknown value 2467, pDiagInfo=&0006f8ec, cbDiagInfoMax=140,
pcbDiagInfo=&0006f99a )
    ---> Time elapsed - +1.824000E-003 seconds

SQLGetDiagFieldW( pDiagInfo=5439573, pcbDiagInfo=20 )
```

```
    <--- SQL_SUCCESS   Time elapsed - +2.603000E-003 seconds

SQLGetDiagFieldW( fHandleType=SQL_HANDLE_STMT, hHandle=1:1, iRecNumber=1,
fDiagIdentifier=Unknown value 2461, pDiagInfo=&0006f988, cbDiagInfoMax=4,
pcbDiagInfo=&0006f99a )
    ---> Time elapsed - +2.100000E-004 seconds

SQLGetDiagFieldW( pDiagInfo=0, pcbDiagInfo=20 )
    <--- SQL_SUCCESS   Time elapsed - +2.268000E-003 seconds

SQLErrorW( hEnv=0:0, hDbc=0:0, hStmt=1:1, pszSqlState=&0006f978,
pfNativeError=&0006f984, pszErrorMsg=&0006f0e8, cbErrorMsgMax=1024,
pcbErrorMsg=&0006f98e )
    ---> Time elapsed - +2.030000E-004 seconds

SQLErrorW( )
  <--- SQL_NO_DATA_FOUND   Time elapsed - +2.143000E-003 seconds
```

You still have not found any information that shows the cause of this error. This problem requires examining the entire trace.

As previously mentioned, CLI keywords placed in the db2cli.ini file may affect applications, so we should look for any evidence of keywords being set. They will be included in the CLI trace along with the SQLDriverConnectW call. Look at the SQLDriverConnect call from the following CLI trace section:

```
SQLDriverConnectW( hDbc=0:1, hwnd=0:0, szConnStrIn=
"DSN=sample;UID=;PWD=", cbConnStrIn=20, szConnStrOut
=NULL, cbConnStrOutMax=0, pcbConnStrOut=NULL,
fDriverCompletion=SQL_DRIVER_NOPROMPT )
    ---> Time elapsed - +8.110000E-004 seconds
    sqlccsend( ulBytes - 1616 )
    sqlccsend( Handle - 152786688 )
    sqlccsend( ) - rc - 0, time elapsed -
    +8.190000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 1262 ) - rc - 0, time
    elapsed - +1.990000E-004
    sqlccsend( ulBytes - 599 )
    sqlccsend( Handle - 152786688 )
    sqlccsend( ) - rc - 0, time elapsed -
    +4.410000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 237 ) - rc - 0, time
    elapsed - +1.180900E-002
( DBMS NAME="DB2/NT", Version="07.02.0005",
Fixpack="0x23060105" )

( Application Codepage=1252, Database  Codepage=
1252, Char Send/Recv Codepage=1252, Graphic
Send/Recv Codepage=1200, Application Char
Codepage=1252, Application Graphic Codepage=1200 )

( StmtOut="SET CURRENT SCHEMA = 'USER1'" )

    sqlccsend( ulBytes - 220 )
    sqlccsend( Handle - 152786688 )
    sqlccsend( ) - rc - 0, time elapsed -
```



```
        +3.990000E-004
        sqlccrecv( )
        sqlccrecv( ulBytes - 163 ) - rc - 0, time
        elapsed - +4.920000E-004
    ( COMMIT=1 )

        sqlccsend( ulBytes - 196 )
        sqlccsend( Handle - 152786688 )
        sqlccsend( ) - rc - 0, time elapsed -
        +4.660000E-004
        sqlccrecv( )
        sqlccrecv( ulBytes - 27 ) - rc - 0, time
        elapsed - +1.920000E-004

    SQLDriverConnectW( )
        <--- SQL_SUCCESS   Time elapsed -
        +4.637400E-002 seconds
    ( DSN="SAMPLE" )

    ( UID=" " )

    ( PWD="*" )

    ( CURRENTSQLID="USER1" )
```

The problem is the CURRENTSQLID keyword included in the connect call. By examining the db2cli.ini file that is being used by USER1 we would see the following entry:

```
CURRENTSQLID=USER1
```

You can see that USER2 has the CURRENTSQLID keyword set to USER1. The CURRENTSQLID keyword keyword will add 'USER1' as the schema to unqualified SQL statements. So, even though the application used the correct ID (USER2) to connect to the database and issued an SQL statement that had been successful when executed from the command line, there are still other factors that can affect it upon execution.

Even though our application was written in Java, the solution came from examining the CLI trace. This is why it is very important to understand all aspects of how the applications work and where to look when problems arise. Potentially, you could have also found the problem with this application by examining the db2cli.ini files for the users in this scenario before taking any traces. For instance, searching through the CLI trace for 'USER1' may also have lead you to the problem.

OLEDB/ADO applications

OLEDB/ADO applications

OLEDB/ADO applications also go through the DB2 CLI layer to access a db2 database, so you should leave our DB2 CLI tracing on for this section. Before you execute the Visual Basic program used in the next exercise, remove the CLI trace (cli.trc) file from the last sample.

OLEDB applications are only available on the Windows operating system and the most common language for developing them is Visual Basic using the ADO interface. Often errors do not reveal all of the information needed to find out what is wrong at execution time. For these types of applications, if a user has only the binary executable, a DB2 CLI trace may be the best thing for debugging.

Connect to your test database with the user ID USER1 and create a table with the command below:

```
CREATE TABLE TEST (C1 CHAR(20) NOT NULL PRIMARY KEY)
```

Create a Visual Basic application and use it to execute the following section of code. (For assistance in writing Visual Basic applications, DB2 includes samples the DB2 samples directory.)

```
Sub Main()

  Dim db As Connection
  On Error GoTo DisplayError
  Set db = New Connection
  db.CursorLocation = adUseClient
  db.Open "PROVIDER=IBMDADB2;dsn=SAMPLE;uid=;pwd=;"

  db.Execute "insert into test values ('VBSample')"
  MsgBox "Success!"
  Exit Sub

DisplayError:
     MsgBox Err.Description

End Sub
```

You will need to update the above code to change the dsn from SAMPLE to the name of your test database.

If it runs successfully, the application opens a pop-up window containing the word `Success!` Run the application a second time. This time it fails with the following error:

> [DB2/NT] SQL0803N One or more values in the INSERT statement, UPDATE statement, or foreign key update caused by a DELETE statement are not valid because the primary key, unique constraint or unique index identified by "1" constrains tabl "

Sometimes it is surprising when application runs successfully at least once, and then fails. In this case, the error message describes the error sufficiently - a primary key is being violated on the table.

Read the description of an SQL0803 error in the *Messages Reference* at
http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/document.d2w/report?fn=db2v7m0sql0800.htm#HDRSQL0800

Although the application returned the error in this case, the error could also have been found using the CLI trace facility. Often there are multiple ways to determine what is at the root of application errors.

Stored procedures

Creating a stored procedure using the Stored Procedure Builder

Stored procedures are perhaps the most difficult applications to debug, because they execute within the database engine. The best technique is to write robust procedures that contain error-handling logic.

For this section, you will need two user IDs, which we will refer to as USER1 and USER2. You will need to create and test the procedure as part of the setup for this problem.

Using the ID USER1, connect to your database and execute the following DDL to create the table needed for this example:

```
Create table Table table1 (C1 char(20))
Insert into table1 values ( 'Row 1' )
Insert into table1 values ( 'Row 2' )
```

Create the following JDBC stored procedure by invoking the Stored Procedure Builder and opening a project file or creating a new one. When invoking the stored procedure, connect to the database with USER1. Name the procedure StoredProc.GetData and have it return a result set. On the SQLStatement panel enter the statement

```
Select C1 from table1
```

On the options panel give the procedure the specific name `GetData` and a JarID of `StoredProc`.

Upon completion you should be able to build successfully. Try executing the procedure from the DB2 command line with the following statement:

```
Call storedproc.getdata()
```

You should see a successful execution. The results should return the contents of Table1.

```
C1
Row 1
Row 2
```

"GETDATA" RETURN_STATUS: "0"

So far, so good - your stored procedure is now complete.

Stored procedure source files

Suppose that USER1 and USER2 are the only people you want to access this procedure you created. Using the Windows explorer, locate the STOREDPROC.jar file (in the \Program Files\sqllib\fuction\jar directory) and modify the permissions on the file so that only USER1 and USER2 have full control, and no one else has access to the file.

Now, connect to the database as USER2 and attempt to execute the procedure again with the same call statement as above.

You will now get the following error:

```
SQL4304N  Java stored procedure or user-defined
function "STOREDPROC.GETDATA", specific name
"GETDATA" could not load Java class "GetData",
reason code "1".  SQLSTATE=42724
```

Stored procedures are executed within the database engine, so access to the executable programs that run the procedures is also required by the ID under which the database engine runs. For this reason, access to your stored procedures should be controlled by GRANT and REVOKE commands, not by manipulating file permissions. (Note, however, that you may need to alter file permissions when binary files for stored procedures are copied to a database. For example, if you create a C/C++ stored procedure outside of the Stored Procedure Builder environment, the binary file would typically be copied into your stored procedure directory. Once this is done, if access to the file is restricted, you may need to add execute permission to that file.)

Another common problem with stored procedures can be finding and loading the stored procedure itself.. Ensuring the procedure is in the correct location and that file permissions are correct, you can eliminate most SQL4304 errors.

Executing a stored procedure

Now that the file permissions have been restored, execute the stored procedure again from the command line interface. You should get the following error:

```
SQL0204N  "DB2ADMIN.TABLE1"
is an undefined name.  SQLSTATE=42704
```

Even though the procedure was created with the same ID that was used to create the table, and executes under the engine process, it will still use the current ID as the default schema.

Launch the stored procedure builder again and open the GetData procedure. You will need to modify the SQL statement to include the schema USER1. Your new SQL statement should look like this:

```
Select c1 from user1.table1
```

Execute the stored procedure once again from the command line. It should now be successful.

Summary

What you should know

You should now be familiar with many different types of DB2 applications, such as CLI, Java and OLEDB, and how they work with DB2 databases. Also, you should be familiar with some of the methods you can use to debug applications and how to invoke them.

Experiment with the samples included with your DB2 installation. Try making modifications to the samples that will both succeed and fail. Make sure you work with the DB2 sample database or a suitable database that is only for testing and not a production environment.

For more information

For more information see the DB2 product manuals, specifically the *Application and Development Guide* and the *Administration Guide*.

Complete all of tutorials contained in the Certified for Support program. Other tutorials in this series include ways to find problems that will also assist in debugging application errors, such as the db2trace facility.

Also check the various articles available through http://www.ibm.com as well as courses available through IBM Education to build your application development and debugging skills.

Check the DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/udb/winos2unix/support for Technotes on DB2 Applications can also be an excellent resource for assisting in application debugging and development.