Tibero

Driver 연결 가이드

안내서 정보

안내서 제목: Tibero Driver 연결 가이드

발행일: 2013-09-10

소프트웨어 버전: Tibero 5

안내서 버전: 2.1.2

내용 목차

그림 목차

안내서에 대하여

안내서의 대상

ODBC, OLE DB, JDBC, ESQL와 같은 다양한 인터페이스를 통해 Tibero로 연결하기 위한 각 Driver 연동 방법을 소개한다.

안내서의 전제 조건

본 안내서는 Driver 연결 과정을 설명한 안내서이다. 따라서 본 안내서를 원활히 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- 데이터베이스의 이해
- RDBMS의 이해
- 운영체제 및 시스템 환경의 이해
- UNIX 계열(Linux 포함)의 기본 지식

안내서의 제한 조건

본 안내서는 Tibero를 실무에 적용하거나 운용하는 데 필요한 모든 사항을 포함하지 않는다.

안내서 규약

표기	의미
<AaBbCc123>	프로그램 소스 코드의 파일명, 디렉터리
<Ctrl>+C	Ctrl과 C를 동시에 누름
[Button]	GUI의 버튼 또는 메뉴 이름
진하게	강조
" "(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
'입력항목'	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일계정, 웹 사이트
>	메뉴의 진행 순서
+----	하위 디렉터리 또는 파일 있음
|----	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
[그림 1.1]	그림 이름
[표 1.1]	표 이름
`AaBbCc123`	명령어, 명령어 수행 후 화면에 출력된 결과물, 예제코드
{ }	필수 인수 값
[ ]	옵션 인수 값

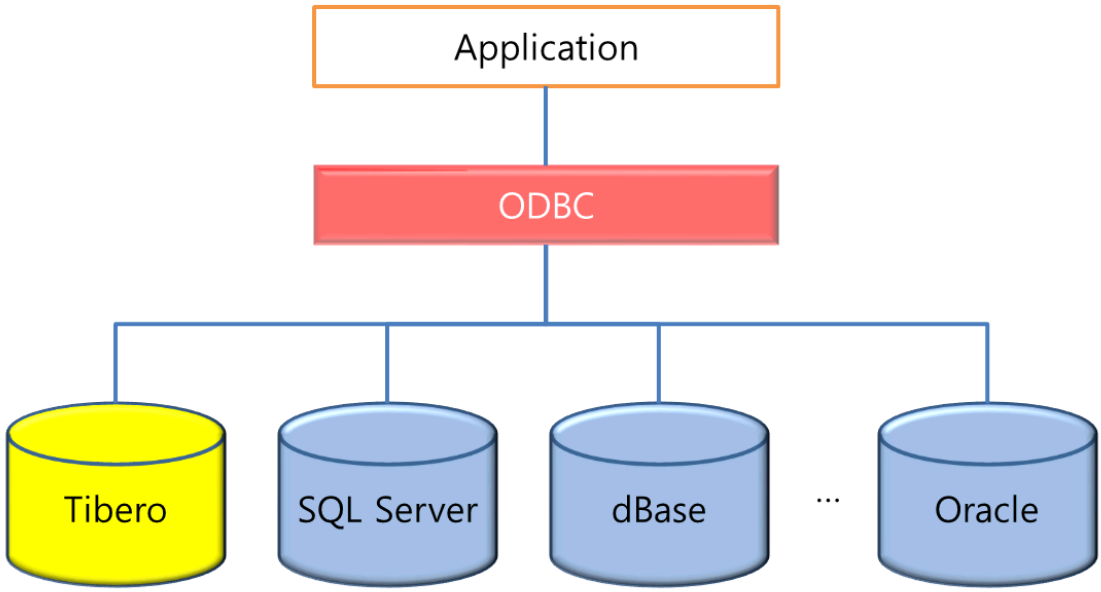
제1장 ODBC 연결

본 장에서는 ODBC에 대한 개념과 Tibero ODBC의 설치 및 구성에 대해서 설명한다.

1.1. ODBC 개념

ODBC(Open DataBase Connectivity)는 모든 DBMS에 독립적인 데이터베이스 애플리케이션을 작성하기 위한 API의 집합으로 특정 DBMS 사용자가 ODBC 드라이버를 통해 다른 DBMS를 사용할 수 있게 한다. 따라서 DBMS에 연결하기 위해 ODBC 드라이버 관리자를 호출하여 사용하려는 드라이버를 호출하고 그 드라이버는 SQL을 사용하여 DBMS와 교신한다. 즉, ODBC는 사용자와 각 데이터베이스 엔진 사이를 연결해 사용자가 공통된 인터페이스로 각각의 다른 데이터베이스 엔진에 접근하게 하여 원하는 데이터를 참조할 수 있도록 한다.

[그림 1.1] ODBC



- Tibero ODBC

  Tibero ODBC는 2.x, 3.x 버전을 모두 지원한다. 단, ODBC 표준 61개 함수 중 아래 2개 함수를 지원하지 않으며 사용을 할 경우 not implemented 에러가 발생한다.

함수	설명
SQLBrowseConnect	연결 문자열을 찾아내기 위해 iterative한 방법을 제공하는 API이다.
SQLSetScrollOptions	3.x에서 SQLSetStmtAttr로 대체한다.

- ODBC 표준 관련 링크

  API에 대한 관련 정보는 http://msdn.microsoft.com/en-us/library/ms712628(VS.85).aspx를 참고한다.

- tbCLI

  Tibero가 제공하는 Call Level Interface(CLI)로 사용자의 애플리케이션 프로그램과 Tibero간의 SQL 인터페이스 역할을 수행한다. tbCLI는 ODBC 및 X/Open Call Level Interface Standard를 기초로 개발되었으며 Tibero ODBC=tbCLI로 인식해도 무방하다.

1.2. Tibero ODBC 설치 및 구성

본 절에서는 ODBC의 설치 파일과 드라이버 등록 및 연결하는 방법에 대해서 설명한다. 기본적으로 32bit 클라이언트 OS 환경에서는 32bit ODBC 설치, 64bit 클라이언트 OS 환경에서는 64bit ODBC 설치 방법을 설명하며 64bit 클라이언트 OS 환경에서 32bit ODBC를 설치하는 방법은 “1.2.4. Windows 64bit에 32bit ODBC 설치”를 참고한다.

1.2.1. ODBC 설치 파일

클라이언트 인스톨러나 ODBC 인스톨러의 경우 GUI 환경에서 설치 및 등록을 자동으로 진행할 수 있다. 만약, 수동으로 진행하려면 Tibero 서버가 설치된 환경에서 설치하고자 하는 클라이언트 컴퓨터 환경에 맞게 설치 파일을 가져온다.

다음은 서버의 운영체제별 바이너리 위치에 대한 설명이다.

- UNIX 계열 서버

구분	바이너리 위치
Windows 32bit	$TB_HOME/client/win32
Windows 64bit	$TB_HOME/client/win64

- Windows 계열 서버

  – 32bit 바이너리

Tibero 버전	구분	위치
Tibero 4 SP1 및 이전 버전	Windows 32bit	%TB_HOME%\client\lib
	Windows 64bit	%TB_HOME%\client\win64
Tibero 5 이상	Windows 32bit	%TB_HOME%\bin
	Windows 64bit	%TB_HOME%\client\win64

– 64bit 바이너리

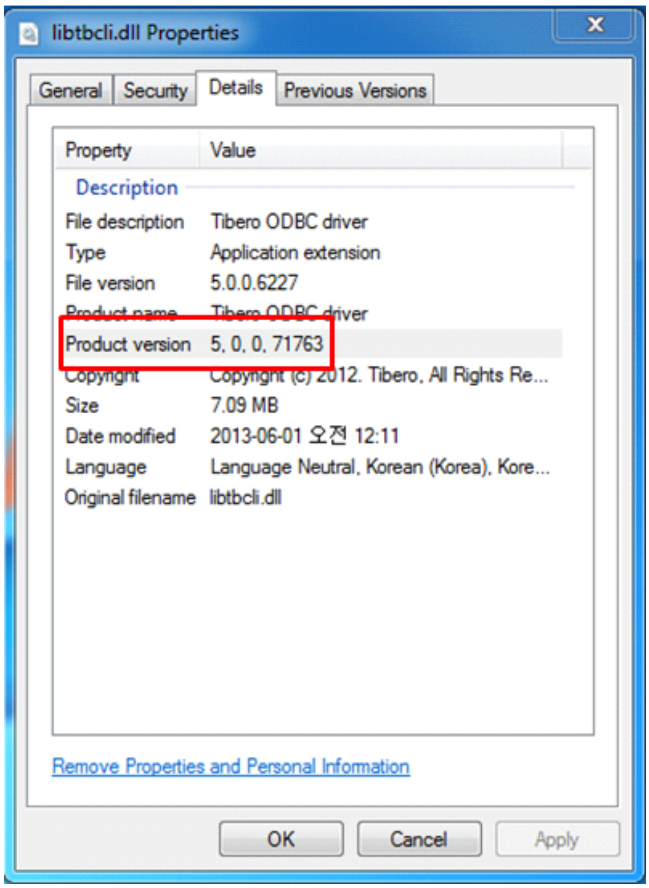
Tibero 버전	구분	위치
Tibero 4 SP1 및 이전 버전	Windows 32bit	%TB_HOME%\client\win32
	Windows 64bit	%TB_HOME%\client\lib
Tibero 5 이상	Windows 32bit	%TB_HOME%\client\win32
	Windows 64bit	%TB_HOME%\bin

다음은 배포되는 바이러리 파일명이다. 해당 파일은 Tibero 5 기준이며 다른 버전의 경우 exe 파일명이 일부 다를 수 있다.

구분	바이너리 파일명
Windows 32bit	libtbcli.dll, libtbcli.lib, tbodbc_driver_installer_5_32.exe
Windows 64bit	libtbcli.dll, libtbcli.lib, tbodbc_driver_installer_5_64.exe

서버와 버전이 일치하는지 확인하기 위해 libtbcli.dll 파일에서 오른쪽 마우스 버튼을 클릭한 뒤 **[속성]** > **[자세히]**의 Product version을 확인한다.

[그림 1.2] ODBC 버전 확인



1.2.2. ODBC 드라이버 등록

ODBC 드라이버 등록 순서는 다음과 같다.

1. Tibero ODBC의 bit 선택

   Tibero ODBC의 32bit 또는 64bit 선택은 설치하는 클라이언트 OS 환경에 맞추기 보다는 Tibero ODBC를 사용하는 실제 어플리케이션의 bit에 맞춘다.

2. 바이너리 복사

   Tibero 버전에 따라 ODBC 바이너리를 복사하는 위치가 다르다. 버전에 맞게 ODBC 바이너리를 위치한다.

   - Tibero 4 SP1 및 이전 버전

     %WINDIR%\system32(예 : c:\windows\system32)

   - Tibero 5 이상

     임의의 위치로 가능하다. 단, 드라이버를 등록할 경우 해당 경로로 지정이 필요하다. (예 : c:\tibero\odbc)

3. 드라이버 등록

   command 창을 열어 ODBC 바이너리가 위치한 곳으로 이동한 후 명령어를 실행한다. 만약 Windows 7 이상일 경우 command 창을 관리자 권한으로 실행한다.

   - 등록 방법

     ```
     tbodbc_driver_installer_5_xx.exe -i <driver path>
     ```

항목	**설명**
<driver path>	ODBC 바이너리 디렉터리이다.

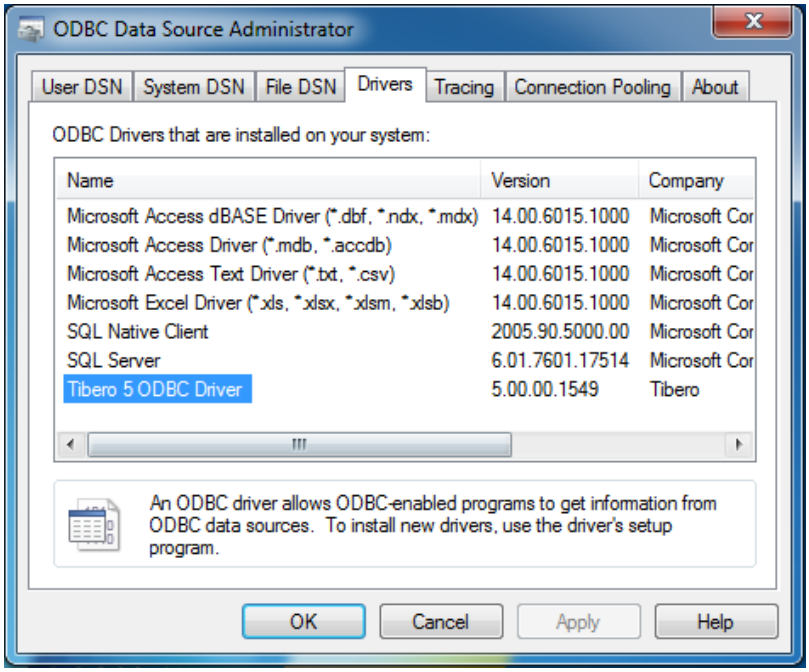
   - 등록 예

     ```
     C:\TmaxSoft\win64>tbodbc_driver_installer_5_64.exe -i c:\TmaxSoft\win64
     ```

4. 드라이버 등록 확인

   [시작] > **[제어판]** > **[관리도구]** > **[데이터 원본(ODBC)]** > **[드라이버]**에서 Tibero 드라이버가 등록된 것을 확인한다.

[그림 1.3] ODBC 등록 확인



1.2.3. ODBC 연결

Tibero ODBC를 사용하는 애플리케이션에서 연결 문자열(Connection String)을 사용하는 방식과 ODBC 관리자에 DSN(Data Source Name)을 등록하는 방식에 대해서 설명한다.

연결 문자열을 사용하는 방식

버전에 따라서 일부 문자열이 다르다. ODBC 함수 중에서 SQLDriverConnect를 사용할 경우 아래와 같은 정보가 필요하다. 실제 사용할 때는 한줄로 입력하며 Tibero 5에서 DB 항목의 경우는 DB NAME에 해당하는 정보를 입력한다.

- Tibero 4 SP1

```
DRIVER={Tibero 4 ODBC Driver};SERVER=192.168.70.185;PORT=8629;
       UID=dbtech;PWD=dbtech;
```

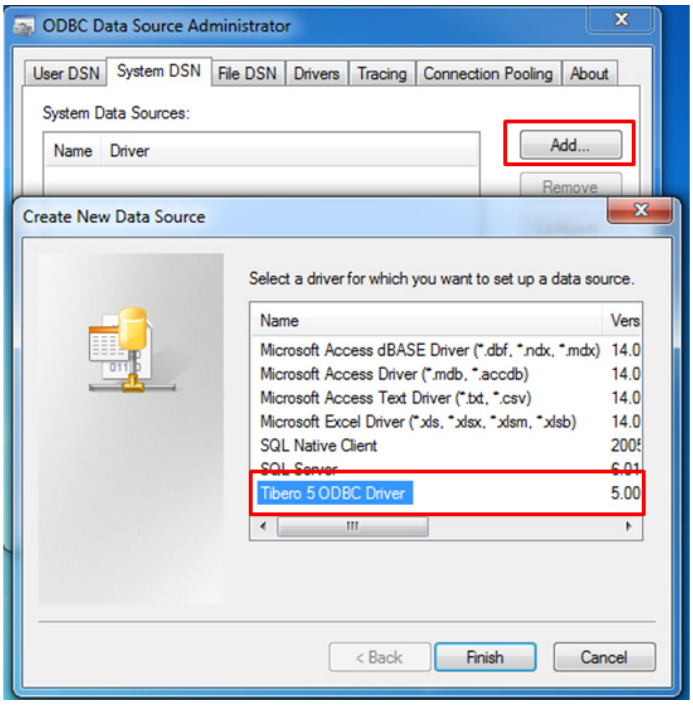
- Tibero 5

```
DRIVER={Tibero 5 ODBC Driver};SERVER=192.168.70.185;PORT=8629;DB=t5;
       UID=dbtech;PWD=dbtech;
```

DSN을 등록하는 방식

ODBC 함수 중에서 SQLConnect를 사용할 때 DSN 정보가 필요하므로 **[제어판]** > **[관리도구]** > **[데이터 원본(ODBC)]** > **[시스템 DSN]**에 데이터소스를 추가한다.

[그림 1.4] ODBC 데이스소스 추가

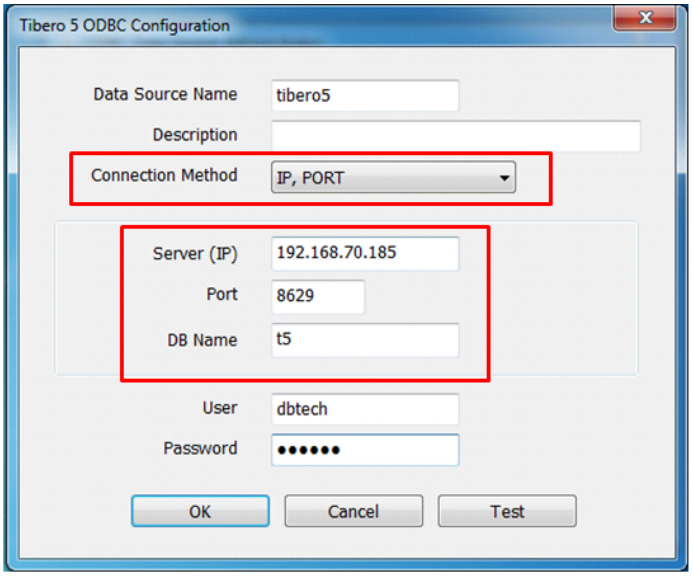


접속방법은 IP, PORT 방식 또는 SID 방식이 있다.

- IP, PORT 방식을 이용한 접속

  Tibero 클라이언트 또는 서버의 설치 없이 ODBC Driver를 등록한 이후에 바로 사용할 수 있다.

[그림 1.5] IP, PORT 방식



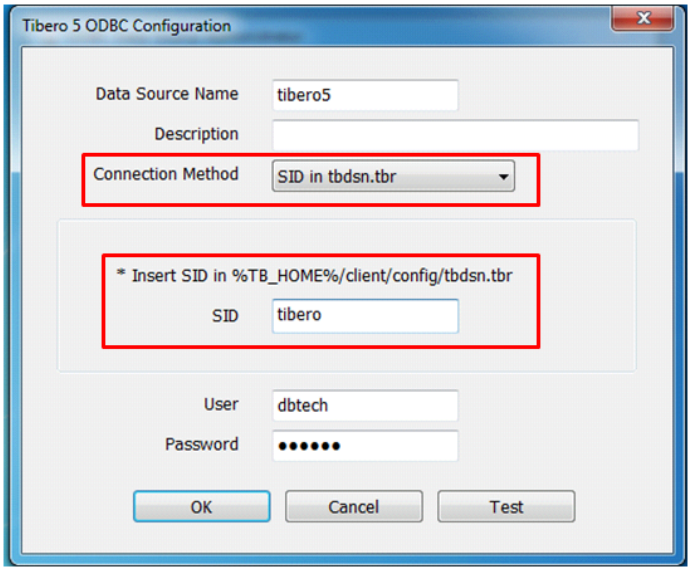
- SID 방식을 이용한 접속

  Tibero 클라이언트 또는 서버가 설치된 경우 사용할 수 있으며 클라이언트 설정 파일인 tbdsn.tbr 파일(Tibero 4 이전 버전의 경우 tbnet_alias.tbr 파일)에 SID 이름이 등록되어 있어야 한다.

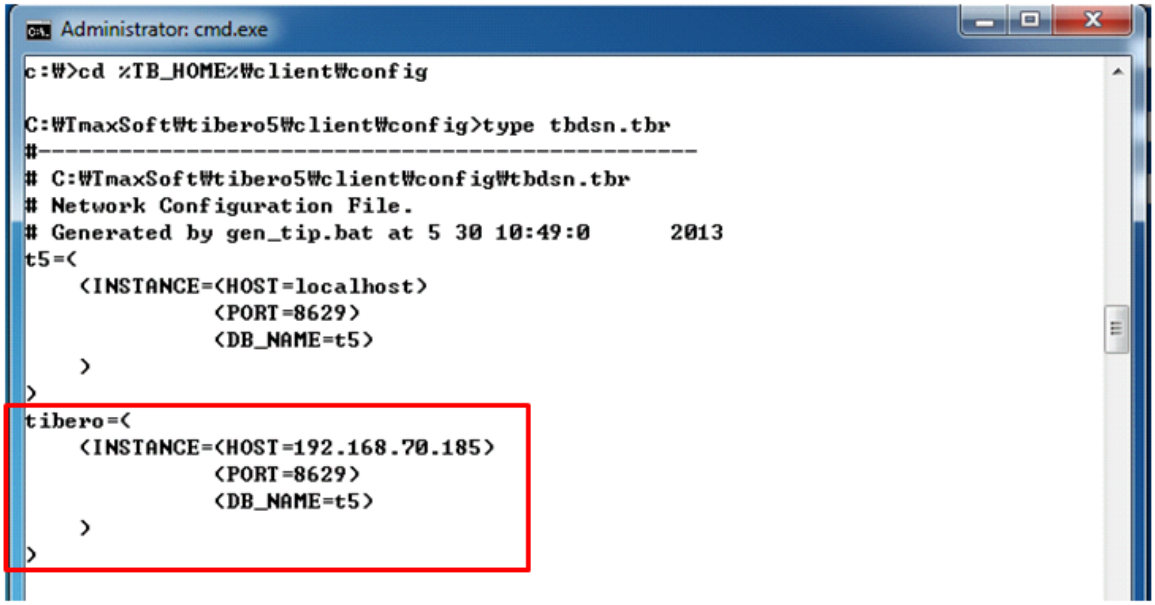
참고

Tibero 클라이언트 설치 방법에 대한 자세한 내용은 "Tibero 클라이언트 설치 가이드"를 참고한다.

[그림 1.6] SID 방식

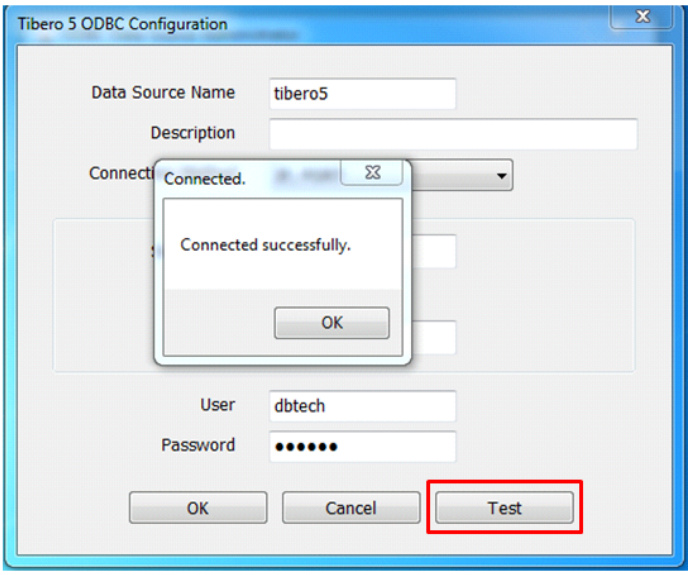


[그림 1.7] tbdsn.tbr



DSN 등록 후 **[Test]** 버튼을 클릭하여 정상적으로 접속이 되는지 확인한다. 만약 실패할 경우 Tibero 서버의 기동, 방화벽 차단, 접속 정보 등을 확인한다.

[그림 1.8] ODBC 접속 테스트



1.2.4. Windows 64bit에 32bit ODBC 설치

Windows 64bit 환경에 32bit Tibero ODBC를 설치할 경우 "%WINDIR%\SysWOW64" 폴더의 32bit용 명령어를 이용한다.

다음은 Windows 64bit에 32bit ODBC 설치하는 과정이다.

1. 바이너리 복사

   Tibero 버전에 따라 ODBC 바이너리를 복사하는 위치가 다르다. 버전에 맞게 ODBC 바이너리를 위치한다.

   - Tibero 4 SP1 및 이전 버전

     %WINDIR%\SysWOW64(예 : c:\windows\SysWOW64)

   - Tibero 5 이상

     임의의 위치로 가능하다. 단, 드라이버를 등록할 경우 해당 경로로 지정이 필요하다.

     (예 : c:\tibero\win32)

2. 드라이버 등록

   command 창을 열어 ODBC 바이너리가 위치한 곳으로 이동한 후 명령어를 실행한다. 만약 Windows 7 이상일 경우 command 창을 관리자 권한으로 실행한다.

- 등록 방법

```
tbodbc_driver_installer_5_32.exe -i <driver path>
```

항목	설명
<driver path>	32bit ODBC 바이너리 디렉터리이다.

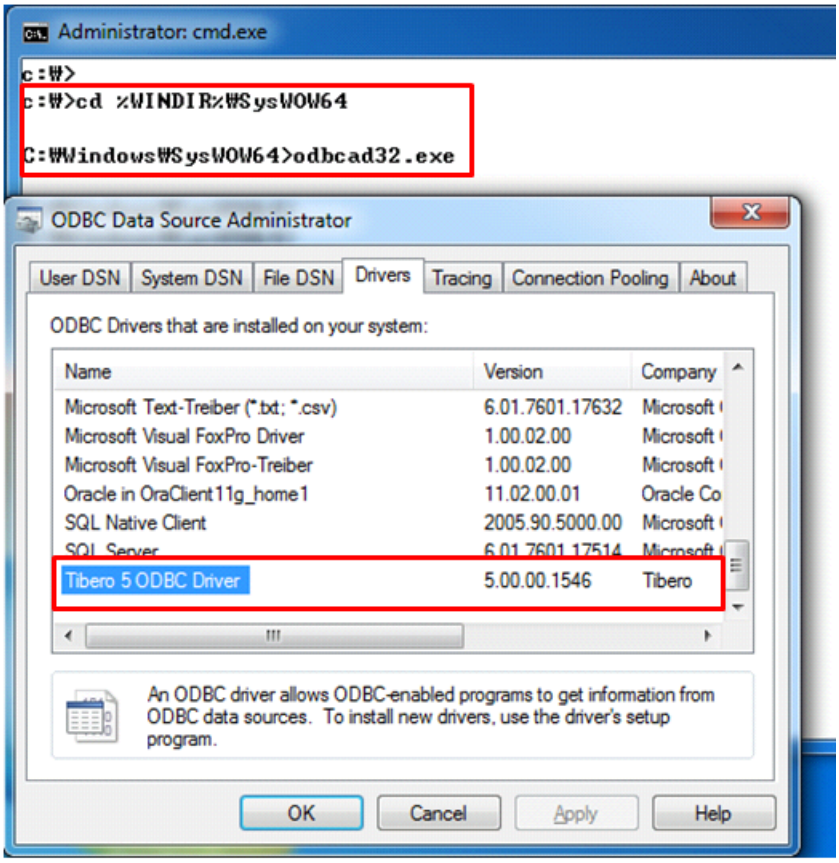
- 등록 예

```
C:\TmaxSoft\win32>tbodbc_driver_installer_5_32.exe -i c:\TmaxSoft\win32
```

3. 드라이버 등록 확인

command 창을 열어 "%WINDIR%\SysWOW64" 폴더로 이동한 후 32bit용 **odbcad32.exe** 명령어를 실행한다.

[그림 1.9] 32bit용 ODBC 데이터 원본 관리자 실행 및 확인



참고

Windows 64bit 환경일 경우 **[시작]** > **[제어판]** > **[관리도구]** > **[데이터 원본(ODBC)]**의 64bit용 odbcad32.exe를 이용한다.

4. ODBC 연결

   ODBC 연결에 대한 자세한 내용은 “1.2.3. ODBC 연결”을 참고한다.

주의

Windows 64bit 환경에 32bit Tibero ODBC를 설치할 경우 ODBC 연결은 32bit용 ODBC 데이터 원본 관리자를 이용해야 한다.

1.3. ODBC Manager 설치

본 절에서는 UNIX 계열(Linux 포함)에서 iodbc 설치 및 연동하는 과정을 설명한다. UNIX 계열의 경우 ODBC Manager가 존재하지 않으므로 iodbc 또는 UNIX ODBC를 별도로 설치해야 한다.

다음은 ODBC Manager 설치 과정에 대한 설명이다.

1. Driver Manager 설치
2. profile 설정
3. ODBC 환경 파일 설정 및 확인

각 과정에 대한 상세한 설명은 해당 절의 내용을 참고한다.

Driver Manager 설치

다음은 Driver Manager를 설치하는 과정에 대한 설명이다.

1. 다운로드

   http://iodbc.org에서 설치 파일 다운로드 후 설치를 원하는 서버에 업로드한다.

   다음은 libiodbc-3.52.7.tar.gz 파일을 다운받아 서버에 설치한 결과이다. 만약, tar 옵션의 xvzf가 적용되지 않을 경우 먼저 gunzip으로 압축을 해제한 후 tar -xvf 옵션으로 해제한다.

```
$ tar -xvzf libiodbc-3.52.7.tar.gz
libiodbc-3.52.7/
libiodbc-3.52.7/admin/
libiodbc-3.52.7/admin/gtk-2.0.m4
libiodbc-3.52.7/admin/gtk.m4
libiodbc-3.52.7/admin/libtool.m4
libiodbc-3.52.7/admin/ltoptions.m4
libiodbc-3.52.7/admin/ltsugar.m4
libiodbc-3.52.7/admin/ltversion.m4
libiodbc-3.52.7/admin/lt~obsolete.m4
```

```
libiodbc-3.52.7/admin/Makefile.am
libiodbc-3.52.7/admin/Makefile.in
libiodbc-3.52.7/admin/libiodbc.pc.in
libiodbc-3.52.7/admin/libiodbc.spec.in
libiodbc-3.52.7/admin/config.guess
libiodbc-3.52.7/admin/config.sub
libiodbc-3.52.7/admin/depcomp
libiodbc-3.52.7/admin/install-sh
libiodbc-3.52.7/admin/ltmain.sh
libiodbc-3.52.7/admin/missing
libiodbc-3.52.7/admin/mkinstalldirs
libiodbc-3.52.7/debian/
libiodbc-3.52.7/debian/changelog
```

2. iodbc 설치

   tar.gz 파일을 가지고 설치를 진행할 경우 아래와 같은 순서를 따른다.

   설치 전 사전환경 점검에서 prefix를 설정하지 않으면 기본적으로 /usr/local 아래에 설치되기 때문에 특정 디렉터리를 지정하여 설치하고, 컴파일 및 설치 작업을 다시 수행할 경우 **make clean** 이후에 진행한다.

   a. '$HOME/iodbc' 디렉터리에 설치 전 사전환경 점검을 한다.

   ```
   ./configure --prefix=$HOME/iodbc --disable-gui
   ```

   b. 점검한 결과 문제가 없으면 컴파일한다.

   ```
   ./make
   ```

   c. 컴파일된 파일을 설치한다.

   ```
   ./make install
   ```

3. 설치 확인

   설치된 서버에 원하는 bit로 설치가 되었는지 확인한다. 특정 OS에 따라 64bit 서버에 32bit로 설치되는 경우가 있으므로 **file** 명령어를 이용하여 확인이 필요하다.

   만약, ODBC Manager가 64bit로 설치됐다면 내부적으로 사용하는 Tibero ODBC 역시 64bit여야 하며 재설치가 필요할 경우 **make clean** 이후에 진행한다.

   ```
   $ cd $HOME/iodbc/bin
   $ file iodbctest
   iodbctest: ELF 64-bit LSB executable, AMD x86-64, version 1 (SYSV), for
   GNU/Linux 2.6.9, dynamically linked (uses shared libs), not stripped
   ```

[참고]

다음은 OS별로 iodbc 64bit 설치와 설치 확인 방법에 대한 설명이다.

- AIX

1. 사전 환경 점검

```
export CFLAGS=-maix64
export LDFLAGS="-maix64 -brtl"
./configure --prefix=$HOME/iodbc --disable-gui
```

2. 컴파일

```
export OBJECT_MODE=64
./make
```

3. 설치

```
./make install
```

4. 설치 확인

```
$ file iodbctest
iodbctest: 64-bit XCOFF executable or object module not stripped
```

- HP(IA64)

1. 사전 환경 점검

```
export CFLAGS=+DD64
./configure --prefix=$HOME/iodbc --disable-gui
```

2. 컴파일

```
./make
```

3. 설치

```
./make install
```

4. 설치 확인

```
$ file iodbctest
iodbctest: ELF-64 executable object file - IA64
```

- SunOS

1. 사전 환경 점검

```
export CFLAGS=-m64
./configure --prefix=$HOME/iodbc --disable-gui
```

2. 컴파일

```
./make
```

3. 설치

```
./make install
```

4. 설치 확인

```
$ file iodbctest
iodbctest: ELF 64-bit MSB executable SPARCV9 Version 1,
dynamically linked, not stripped, no debugging information available
```

- Linux

1. 사전 환경 점검

```
export CFLAGS=-m64
./configure --prefix=$HOME/iodbc --disable-gui
```

2. 컴파일

```
./make
```

3. 설치

```
./make install
```

4. 설치 확인

```
$ file iodbctest
iodbctest: ELF 64-bit LSB executable, AMD x86-64, version 1(SYSV),
for GNU/Linux 2.6.9, dynamically linked(uses shared libs), not stripped
```

profile 설정

profile 내에 아래와 같은 내용을 추가한다. IODBC_HOME의 경우 iodbc를 설치한 위치로 설정한다.

```
# iodbc setting
export IODBC_HOME=$HOME/iodbc
export LD_LIBRARY_PATH=$IODBC_HOME/lib:$LD_LIBRARY_PATH
export PATH=$IODBC_HOME/bin:$PATHh"
```

주의

OS에 맞게 환경변수 LD_LIBRARY_PATH(Linux), LIBPATH(AIX), SHLIB_PATH(HP)를 설정한다.

ODBC 환경 파일 설정 및 확인

ODBC Driver Manager의 환경 파일에 Tibero ODBC Driver를 등록하는 방법이다. 연결 테스트 전에 Tibero 클라이언트 또는 서버의 설치 및 관련 환경설정이 되어야 한다.

1. 환경 파일 위치 및 이름 설정

   $HOME/.odbc.ini(개인설정) 또는 /etc/odbc.ini(공통설정)로 설정 가능하다. 우선순위는 $HOME/.odbc.ini 가 높다.

   - ODBC 환경 파일 설정 방법

```
[ODBC Data Sources]
<ODBC Data Sources> = Tibero5 ODBC driver

[ODBC]
Trace = 1
TraceFile = /home/tibero/iodbc/tmp/odbc.trace

[<ODBC Data Sources 세부설정>]
Driver          = <Tibero ODBC Driver 파일>
Description     = Tibero5 ODBC Datasource
SID             = <tbdsn.tbr 파일에 설정한 alias 정보>
User            = dbtech
Password        = dbtech
```

항목	설명
<*ODBC Data Sources*>	Datasource 이름으로 Oracle Gateway 설정 파일에 해당 내용이 들어간다.
<*ODBC Data Sources 세부설정*>	ODBC Data Sources에서 설정한 이름으로 대소문자까지 일치해야 한다.
Driver	ODBC Manager에서 로드하는 Tibero ODBC Driver 파일이다. 해당 파일 존재 여부 및 권한에 대해서 확인이 필요하다.
SID	Tibero 클라이언트 또는 서버의 tbdsn.tbr 파일에 설정한 Alias 정보이다.
User	사용자를 의미한다. 테스트 및 링크 생성의 경우 별도로 사용자를 가져가므로 크게 의미는 없다.
Password	사용자 패스워드를 의미한다. 테스트 및 링크 생성의 경우 별도로 사용자를 가져가므로 크게 의미는 없다.

   - ODBC 환경 파일 설정 예

```
[ODBC Data Sources]
tibero5 = Tibero5 ODBC driver

[ODBC]
```

```
Trace = 1
TraceFile = /home/tibero/iodbc/tmp/odbc.trace

[tibero5]
Driver         = /home/tibero/tibero5/client/lib/libtbodbc.so
Description    = Tibero5 ODBC Datasource
SID            = tibero
User           = dbtech
Password       = dbtech
```

2. 연결 테스트

'$IODBC_HOME/bin' 폴더에 있는 iodbctest를 이용하여 연결 테스트를 수행한다. 테스트에 문제가 발생하는 경우 ODBC 환경 파일 이름 및 위치와 설정을 확인한다.

다음은 iodbctest를 이용한 연결 테스트 설정 방법이다.

```
iodbctest "DSN=<dsn>;UID=<user>;PWD=<pwd>"
```

항목	설명
DSN	ODBC 환경 파일에서 설정한 ODBC Datasources 이름이다.
UID, PWD	테스트 할 접속 계정 및 패스워드이다.

다음은 iodbctest를 이용한 연결 테스트 실행 예이다.

```
$ iodbctest "DSN=tibero5;UID=dbtech;PWD=dbtech"
iODBC Demonstration program
This program shows an interactive SQL processor
Driver Manager: 03.52.0709.0909
Driver: 05.00.0204 (libtbodbc.so)

SQL>select * from dept;

DEPTNO |DEPTNAME                      |MGRNO    |LOCNO
-------+------------------------------+---------+-------
100    |Administration                |2000     |2700
200    |Marketing                     |2010     |2800
300    |Purchasing                    |1140     |2700
400    |Human Resources               |2030     |3400
```

1.4. 문제 해결

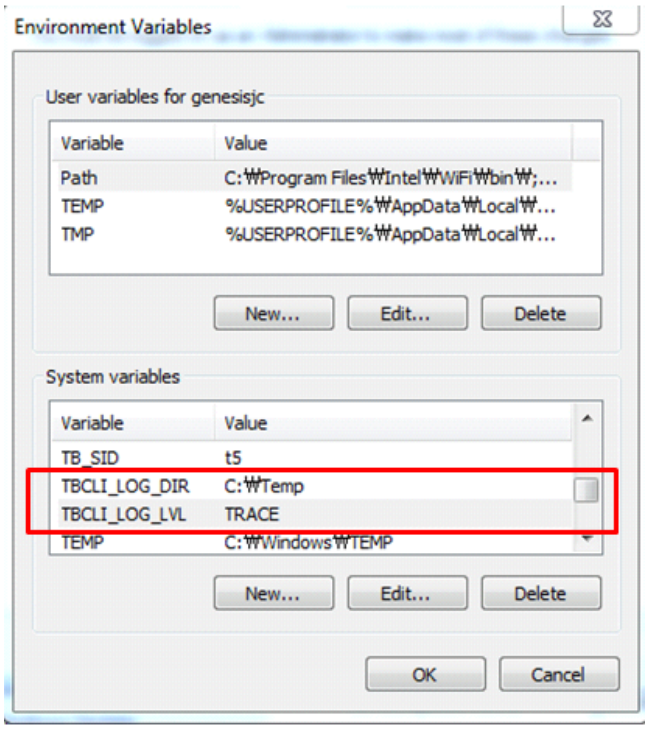
본 절에서는 특정 문제가 발생했을 때 해결하는 방법을 설명한다.

1.4.1. 로그 발생

Tibero ODBC(=tbCLI)에서는 특정 환경변수를 적용하여 로그를 발생시키거나 글자 깨짐 등의 문제를 해결할 수 있다. 로그를 발생하는 환경변수의 경우 일부 성능이 느려질 수 있으므로 문제가 있을 때 일시적으로만 사용한다.

아래처럼 환경변수를 적용한 후에 애플리케이션을 재기동한다.

[그림 1.10] ODBC(=tbCLI) 환경변수 적용



다음은 자주 사용하는 환경변수에 대한 설명이다.

환경변수명	설명
TBCLI_LOG_LVL	로그를 출력하게 하는 환경변수이다. trace 값을 설정하면 로그가 출력된다.
TBCLI_LOG_DIR	로그를 생성하는 디렉터리를 설정한다. 설정하지 않을 경우 아래와 같은 경로에 로그가 생성된다. – Windows 계열 : C:\tbcli_날짜시간.log – UNIX 계열 : /tmp/tbcli_날짜시간.log

환경변수명	설명
	Windows 7은 C:\ 디렉터리에 파일을 생성할 때 관리자 권한이 필요하므로 해당 경로에 로그를 생성하려면 애플리케이션을 관리자 권한으로 실행한다.
TB_NLS_LANG	클라이언트의 캐릭터 셋을 설정하는 부분으로 기본은 MSWIN949(한글)로 설정되어 있다. 보통은 DB 캐릭터 셋과 일치시키거나 부분집합으로 설정하며 설정할 수 있는 값은 DB 캐릭터 셋과 동일하다.

참고

다른 환경변수에 대해서는 제품 매뉴얼 **"Tibero 관리자 안내서"**의 **"Appendix D. 클라이언트 환경변수"**를 참고한다.

1.5. 예제

본 절에서는 ODBC를 이용하여 연결하는 기본적인 형태의 예제와 특수한 타입의 예제를 설명한다.

1.5.1. 연결 예제

다음은 ODBC를 이용한 Windows 환경의 연결 예제이다.

define _DIRECT_를 주석 여부에 따라 SQLDriverConnect와 SQLConnect 중 하나를 사용해서 접속한다.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
/* #include "stdafx.h" */
#define ROWSET_SIZE 20

#define _DIRECT_

int main(int argc, char* argv[])
{
    SQLRETURN rc = SQL_SUCCESS;
    SQLUINTEGER len;
    SQLHANDLE henv, hdbc, hstmt;
    SQLCHAR *sql = (SQLCHAR *)"SELECT TO_CHAR(SYSDATE,'YYYYMMDD') FROM DUAL";
    char buf[128];

/* Env Handle  */
    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
```

```
    SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3, 0);
    SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);



/* Tibero Connect */
#ifdef _DIRECT_

 // IP, PORT
 rc = SQLDriverConnect(hdbc,
        (SQLHWND)NULL,
        //(SQLCHAR *) "DRIVER={Tibero 4 ODBC Driver};\
                        SERVER=192.168.70.185;PORT=9629;UID=tibero;PWD=tmax;",
        (SQLCHAR *) "DRIVER={Tibero 5 ODBC Driver};\
            SERVER=192.168.70.185;PORT=9629;DB=t5;UID=dbtech;PWD=dbtech;",
        SQL_NTS,
        NULL,
        0,
        NULL,
        SQL_DRIVER_NOPROMPT);


#else

 // Data Source Name
 rc = SQLConnect(hdbc,
                    (SQLCHAR *)"t5", SQL_NTS,      /* Data Source Name or DB NAME */
                    (SQLCHAR *)"dbtech", SQL_NTS,  /* User */
                    (SQLCHAR *)"dbtech", SQL_NTS); /* Password */
#endif

 if (rc != SQL_SUCCESS) {
        fprintf(stderr, "Connection failed!!!");
        exit(1);
    }


/* Statements Handle */
    SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    printf("Query: %s\n", sql);

/* Execute Query */
    rc = SQLExecDirect(hstmt, sql, SQL_NTS);
    if (rc != SQL_SUCCESS) {
fprintf(stderr, "SQLExecDirect failed!!!");
        exit(1);
    }
```

```
/* Bind Result */
    SQLBindCol(hstmt, 1, SQL_C_CHAR, (SQLCHAR *)buf, 128, (long *)&len);

    printf("Result: ", buf);

/* Fetch Result */
    while(SQLFetch(hstmt) != SQL_NO_DATA) {
        printf("%s\n", buf);
    }

/* Release Handle and Close Connection */
 SQLFreeStmt(hstmt, SQL_DROP);
 SQLDisconnect(hdbc);
 SQLFreeConnect(hdbc);
 SQLFreeEnv(henv);
 return 0;
}
```

UNIX 계열에서 테스트 하고 싶다면 Tibero 클라이언트 또는 서버가 설치된 환경에서 위의 샘플 소스를 아래와 같이 삭제, 추가, 컴파일 및 링크한다.

아래의 경우는 Linux 64bit일 때 예제이다.

- 삭제

```
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
```

- 추가

```
#include <sqlcli.h>
```

- 컴파일 및 링크

```
cc -m64 -O -I$TB_HOME/client/include -L$TB_HOME/client/lib -c test.c
cc -m64 -O -I$TB_HOME/client/include -o test.bin -L$TB_HOME/client/lib -ltbcli \
-lclialloc test.o
```

참고

컴파일 및 링크 옵션의 경우 OS에 따라 일부 다를 수 있다. 강조된 부분의 경우는 동일하고 나머지 부분은 원래 사용하던 컴파일 및 링크 옵션을 사용한다.

Autocommit 예제

ODBC의 Autocommit 관련 기본설정은 ON 이다. 애플리케이션에서 OFF로 처리하기 위해서는 아래와 같이 처리한다.

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sql.h>
#include <sqlext.h>
//#include <sqlcli.h>

int main()
{
    SQLRETURN rc;
 SQLHANDLE henv, hdbc, hstmt;
    SQLCHAR sRegNo[10], sRegName[25];
    SQLINTEGER sqlnts=SQL_NTS;

 /* Env Handle  */
    SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
    SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(void*)SQL_OV_ODBC3,0);
    SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);

 // Set Connection Attribute
    //SQLSetConnectAttr(hdbc,5,(void*)SQL_LOGIN_TIMEOUT,0);


 /* Tibero Connect */
 rc = SQLConnect(hdbc,
                (SQLCHAR *)"t5", SQL_NTS,       /* Data Source Name or DB NAME */
                (SQLCHAR *)"dbtech", SQL_NTS,   /* User */
                (SQLCHAR *)"dbtech", SQL_NTS);  /* Password */

 if (rc != SQL_SUCCESS) {
  printf("Connection failed!!! : %d", rc);
        exit(1);
    }

 /* Set AutoCommit ON/OFF */
rc = SQLSetConnectAttr(hdbc,SQL_ATTR_AUTOCOMMIT,(SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);
//rc = SQLSetConnectAttr(hdbc,SQL_ATTR_AUTOCOMMIT,(SQLPOINTER)SQL_AUTOCOMMIT_ON, 0);

    if (rc != 0) return NULL;
```

```
    /* Statements Handle */
    SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);

 /* Execute Query - INSERT */
rc = SQLExecDirect(hstmt,(SQLCHAR *)"INSERT INTO reg VALUES(100,'Seoul')",SQL_NTS);
    if (rc != SQL_SUCCESS) {
  printf("Insert Failed!!!");
        exit(1);
    }

 /* Commit */
 // rc = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);

    /* Execute Query - SELECT */
    rc = SQLExecDirect(hstmt,(SQLCHAR*)"SELECT * FROM reg",SQL_NTS);
    if (rc != SQL_SUCCESS) {
  printf(stderr, "Select Failed!!!");
        exit(1);
    }

 // Fetch
 printf("%10s%25s\n","REGNO","REGNAME");
 printf("%10s%25s\n","=====","========================");

 while( SQLFetch(hstmt) != SQL_NO_DATA ) {
  SQLGetData(hstmt,1,SQL_C_CHAR,sRegNo,10,&sqlnts);
  SQLGetData(hstmt,2,SQL_C_CHAR,sRegName,25,&sqlnts);
  printf("%10s%25s\n", sRegNo,sRegName);
 } // while end

 printf("%10s%25s\n","=====","========================");


 /* Release Handle and Close Connection */
 SQLFreeStmt(hstmt, SQL_DROP);
 SQLDisconnect(hdbc);
 SQLFreeConnect(hdbc);
 SQLFreeEnv(henv);

    return 0;

}
```

1.5.2. 타입 관련 예제

다음은 CLOB을 이용한 특수한 타입의 예제이다.

```
// $gcc test_clob.c -I$TB_HOME/client/include -L$TB_HOME/client/lib -ltbcli
// a.out DSN USERNAME PASSWORD


#include <stdio.h>
#include <assert.h>
#include "sqlcli.h"

SQLRETURN _rc = SQL_SUCCESS;
SQLHANDLE _henv = SQL_NULL_HANDLE;
SQLHANDLE _hdbc = SQL_NULL_HANDLE;
SQLHANDLE _hstmt = SQL_NULL_HANDLE;
SQLCHAR *DSN = NULL;
SQLCHAR *UID = NULL;
SQLCHAR *PWD = NULL;

int
main(int argc, char **argv)
{
    SQLINTEGER lob_loc = 0;
    SQLLEN ind = 0;
    SQLLEN lob_ind = 0;
    SQLINTEGER byte_len = 0;
    SQLBIGINT lob_len = 0;
    SQLCHAR lob_char[10+1];        // 버퍼크기(10)+널텀(1)

    // 읽어들일 글자 수, 버퍼크기가 10이므로 5로 설정
    SQLINTEGER read_char_len = 5;
    SQLINTEGER sqlnts=SQL_NTS;

    int offset = 1; // 여러번 나누어 가져올때의 offset
    int lob_length;

    _rc = SQLAllocEnv(&_henv);
    _rc = SQLAllocConnect(_henv, &_hdbc);

    DSN = (SQLCHAR *)argv[1];
    UID = (SQLCHAR *)argv[2];
    PWD = (SQLCHAR *)argv[3];

    _rc = SQLConnect(_hdbc, DSN, SQL_NTS, UID, SQL_NTS, PWD, SQL_NTS);

    _rc = SQLAllocStmt(_hdbc, &_hstmt);
```

```
_rc = SQLExecDirect(_hstmt, (SQLCHAR *)"DROP TABLE T", SQL_NTS);
_rc = SQLExecDirect(_hstmt, (SQLCHAR *)"CREATE TABLE T(C1 CLOB)", SQL_NTS);
_rc = SQLExecDirect(_hstmt, (SQLCHAR *)"DELETE FROM T", SQL_NTS);
_rc = SQLExecDirect(_hstmt, (SQLCHAR *)"INSERT INTO T VALUES (NULL)",
                    SQL_NTS);
_rc = SQLEndTran(SQL_HANDLE_DBC, _hdbc, SQL_COMMIT);
_rc = SQLFreeStmt(_hstmt, SQL_RESET_PARAMS);

/* Case 1 */
_rc = SQLExecDirect(_hstmt, (SQLCHAR *)"SELECT * FROM T", SQL_NTS);
_rc = SQLFetch(_hstmt);

/* must be return SQL_SUCCESS with SQL_NULL_DATA indicator value
 * at first invocation */
_rc = SQLGetData(_hstmt, 1, SQL_C_CHAR, NULL, 0, &ind);

/* c1컬럼(clob)이 null인지 확인. null이면 indicator가 SQL_NULL_DATA */
if (ind == SQL_NULL_DATA) {
    printf("CASE1: lob : null\n");
}

/* must be return SQL_NO_DATA at second invocation*/
_rc = SQLGetData(_hstmt, 1, SQL_C_CHAR, NULL, 0, &ind);
assert(_rc == SQL_NO_DATA);

_rc = SQLCloseCursor(_hstmt);
_rc = SQLFreeStmt(_hstmt, SQL_DROP);

/* Case 2 : lob locator사용 */
ind = 0;
_rc = SQLAllocStmt(_hdbc, &_hstmt);

_rc = SQLExecDirect(_hstmt, (SQLCHAR *)"DELETE FROM T", SQL_NTS);
_rc = SQLExecDirect(_hstmt, (SQLCHAR *)"INSERT INTO T \
                    VALUES ('티맥스소프트abc티베로입니다.')", SQL_NTS);

_rc = SQLExecDirect(_hstmt, (SQLCHAR *)"SELECT * FROM T", SQL_NTS);
_rc = SQLBindCol(_hstmt, 1, SQL_C_CLOB_LOCATOR, \
                           (SQLPOINTER)&lob_loc, 0, &ind);
_rc = SQLFetch(_hstmt); // c1이 null이 아니면 lob_loc이 여기서 채워짐

/* 해당 lob 데이터가 NULL인지 확인 */
if (ind == SQL_NULL_DATA) {
    printf("CASE2 : lob : null\n");
}
else {
    _rc = SQLLobGetLength(_hstmt, lob_loc, &lob_len, NULL);
```

```
        printf("CASE2(lob_len) : lob len : %lld\n", lob_len);

        /*
        SQLRETURN SQL_API
        SQLLobGetData2(SQLHSTMT         StatementHandle,
               SQLINTEGER       Locator,
// 1부터 시작, 문자 수 의미의 offset, 여러번 호출 시 값 변경 필요
               SQLBIGINT        Offset,
               SQLINTEGER      *ReadLength, // 읽어들일 문자 수
               SQLSMALLINT      TargetCType,
               SQLPOINTER       DataPtr,
               SQLINTEGER       BufferLength, // 버퍼크기
               SQLLEN          *Indicator)
        */

        while (1) {
                if( read_char_len > lob_len )
                        read_char_len = lob_len;

                _rc = SQLLobGetData2(_hstmt, lob_loc, offset, &read_char_len \
                , SQL_C_CHAR,lob_char, sizeof(lob_char),&lob_ind);
                if (!SQL_SUCCEEDED(_rc) || lob_ind <= 0)
                        break;
                offset += read_char_len;
                lob_len -= read_char_len;

                printf("length of writing buffer : %lld\n", lob_ind);
                printf("lob data : %s\n", lob_char);
        }// while end

   }


    //_rc = SQLExecDirect(_hstmt, (SQLCHAR *)"DROP TABLE T", SQL_NTS);
    _rc = SQLDisconnect(_hdbc);
    _rc = SQLFreeConnect(_hdbc);
    _rc = SQLFreeEnv(_henv);

    return 0;
}
```

제2장 OLE DB 연결

본 장에서는 OLE DB에 대한 개념과 기능, Tibero OLE DB Provider 설치 및 구성에 대해서 설명한다.

2.1. OLE DB 개념

ODBC는 응용 프로그램이 데이터베이스를 일정한 방식으로 액세스하는 방법을 제공하였다. 언어, 테이블 구조, 내부 정보 등에 관계없이 데이터베이스를 액세스하는 공통의 추상적인 API를 제공했지만 IT기술이 발전함에 따라 새로운 방식으로 DB기반 응용 프로그램을 설계하고 구축하는 상황에 ODBC는 부적합하게 되어 OLE DB라는 새로운 개방형 DB 연결방식이 생겨났다.

OLE DB는 Microsoft UDA(Universal Data Access)의 개념을 구체화한 프로그래밍 인터페이스 모델이다. UDA는 단일 COM 기반 프로그래밍 인터페이스를 사용하여 관계형, 비관계형, 계층형 등과 같은 모든 유형의 데이터를 액세스할 수 있는 기능을 제공한다.

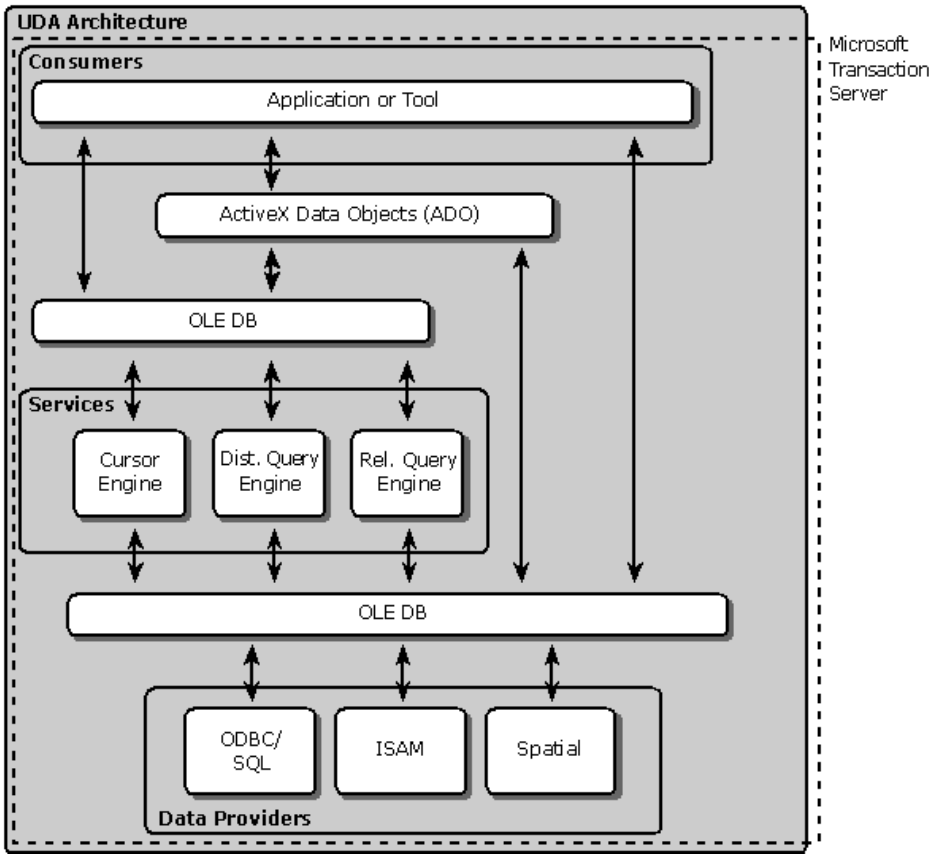
Tibero OLE DB Provider는 현재 Windows 운영체제만을 지원한다. Tibero OLE DB Provider는 ADO 또는 OLE DB 기반 애플리케이션이 Tibero 데이터베이스에 접근하는 환경의 성능 및 안정성을 보장한다. Tibero OLE DB Provider가 최신 OLE DB 및 ADO 스펙과 호환하므로 ADO, OLE DB 개발자는 Tibero 환경으로의 애플리케이션 마이그레이션 작업을 쉽고 간단하게 수행할 수 있다. 또 Tibero OLE DB Provider는 PSM 저장 프로시저, LOB 등 Tibero 환경이 제공하는 기능을 활용하는 것을 가능하게 하며 Microsoft OLE DB .NET data provider를 통해 .NET 환경을 완벽하게 지원한다. OLE DB .NET을 사용하는 경우 모든 종류의 .NET 프로그래밍 언어를 사용하여 Tibero 데이터베이스에 접근할 수 있다.

2.1.1. OLE DB의 역할

기본적으로 분산되어 있는 데이터들에 대한 통합된 view를 제공하는 역할을 한다.

아래 그램처럼 UDA 기술은 ADO와 OLEDB를 사용하여 SQL 데이터 뿐만 아니라 비 SQL적인 데이터(메일, 텍스트, 디렉터리 서비스 등)에도 접근 가능하고 OLE DB가 기존의 ODBC를 이용한 데이터 연결도 사용할 수 있게 되어있다. OLE DB를 통한 접근은 기존의 ODBC가 부족했던 점을 보완하여 접근 속도 또한 많은 향상을 가져왔다.

[그림 2.1] UDA Architecture



(출처)마이크로소프트

2.1.2. OLE DB 내부 구조

OLE DB 내부 구조 및 각 항목에 해당하는 설명은 아래와 같다.

- Data Source

  OLE DB provider 초기화 및 환경구축을 하고 접속 정보를 받아 한 개 이상의 연결(Session)을 생성한다.

- Session

  하나의 연결 단위로 일반적으로 한 개 이상의 명령문(Command)을 생성하고, 직접적으로 한 개 이상의 결과 테이블(Rowset)을 생성한다.

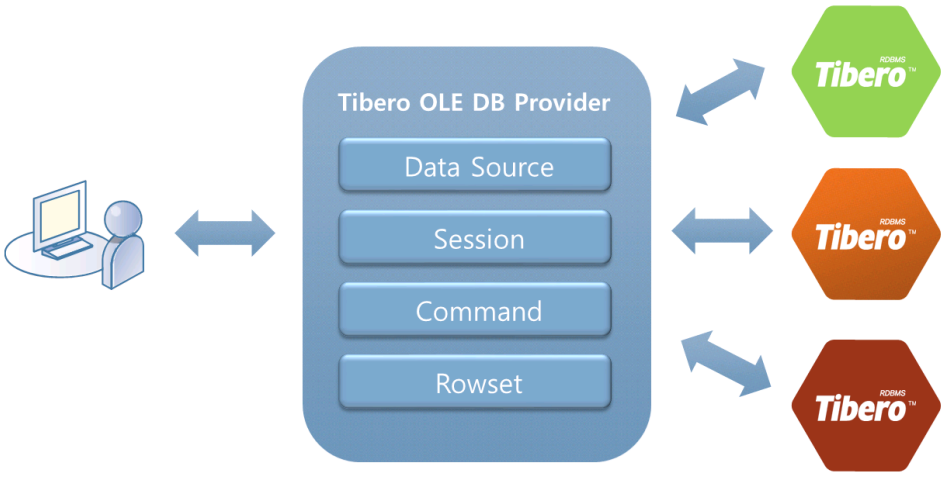
- Command

  SQL 문장 실행의 단위로 SELECT 문일 경우 한 개 이상의 Rowset을 생성한다.

- Rowset

  쿼리에 의해 서버로부터 가져온 데이터들의 집합으로 순방향, 역방향, 특정위치 row 접근을 지원한다.

[그림 2.2] OLE DB 내부 구조



2.1.3. OLE DB 기능

OLE DB의 몇가지 특징적인 기능들에 대해서 살펴본다.

Updatable Cursor

쿼리의 결과 테이블의 데이터를 직접 수정하여 서버에 반영시키는 기능이다. 기본적으로 rowid를 추가로 쿼리에 첨부하여 보내고 rowid를 붙일 수 없는 쿼리는 Updatable Cursor가 될 수 없다. 추가, 삭제, 수정 모두 내부적으로 rowid를 이용한 DML로 작성되어 서버로 보내진다.

Updatable Cursor의 기능들은 다음과 같다.

- 새로운 Row 추가 : 추가된 row는 결과 테이블에 저장되지 않는다.
- 기존 Row를 삭제 : 삭제된 row는 삭제 되었다고 표시만 해놓는다.
- 기존 Row의 데이터를 수정 : 수정된 데이터는 바로 조회가 가능하다.

다음은 Updatable Cursor의 사용 예제(PHP)이다.

```
$conn = new COM("ADODB.Connection");
$conn->Open("Provider=tbprov.Tbprov.5; Data Source=t5; User ID=dbtecj;
Password=dbtech;Updatable Cursor=True;“);
$rs = new COM("ADODB.Recordset");
$rs->CursorLocation = adUseServer;

$rs->Open("for_member", $conn, adOpenKeyset, adLockPessimistic, adCmdTable);
$rs->MoveNext();
$rs->Fields("EMAIL")->value = "test@tibero.com";
$rs->Fields("PASSWORD")->value = "tmax123";
```

```
$rs->Fields("NAME")->value = “TESTUSER”;
$rs->Update();
$rs->Close();
```

Schema Rowset

DB의 스키마 정보(테이블, 프로시저 등)를 테이블 형태로 알려주는 기능으로 대부분 내부적으로 DB의 static view를 조회하여 해당 정보들을 추출한다. DB접근이 필요없는 경우 OLE DB단에서 해당 스키마 정보를 저장하고 있다가 추출하고 Power Builder, Report Designer 등의 툴로 Schema Rowset을 통하여 각종 테이블, 뷰, 인덱스 등의 정보를 가져와 활용한다.

Schema Rowset이 지원하는 항목들은 다음과 같다.

- TABLES
- VIEWS
- COLUMNS
- INDEXES
- PRIMARY_KEYS
- PROCEDURES
- PROCEDURE_PARAMETERS
- PROVIDER_TYPES
- CATALOGS(빈 결과물 리턴)

지원되지 않지만 필요한 Schema 종류일 경우 요청에 의해 지속적으로 추가한다.

다음은 Schema Rowset의 사용 예제(PHP)이다.

```
$conn = new COM("ADODB.Connection");
$conn->Open(“Provider=tbprov.Tbprov.5; …”);
$arr[0] = "";
$arr[1] = "dbtech";
$arr[2] = "EMP";
$rs = new COM("ADODB.Recordset");
$rs->CursorLocation = adUseClient;
$rs = $conn->OpenSchema(adSchemaPrimaryKeys, $arr);
```

Connection Pooling

동일한 계정으로 반복 연결을 할 경우 기존의 연결을 닫지 않고 저장해 두었다가 재사용하는 기능이다. 연결하는 데 드는 시간을 절약하는 효과가 있다.

Connection Pooling의 설정 방법은 다음과 같다.

- 연결 문자열에 OLE DB Services 항목을 추가한다.
  - OLE DB Services=-1 : 풀링을 사용한다.
  - OLE DB Services=-2 : 풀링을 사용하지 않는다.
- PHP의 경우 무조건 사용하도록 설정된다.

2.2. Tibero OLE DB Provider 설치 및 구성

Tibero OLE DB의 경우 내부적으로 Tibero ODBC를 사용하므로 OLE DB를 설치한다면 앞장의 ODBC 내용을 참조하여 미리 설치를 한다. 만약 32bit OLE DB를 사용한다면 32bit ODBC를 설치하고 64bit OLE DB를 사용한다면 64bit ODBC를 설치한다.

본 절에서는 OLE DB의 설치 파일과 OLE DB Provider 등록 및 연결하는 방법에 대해서 설명한다. 기본적으로 32bit 클라이언트 OS 환경에서는 32bit OLE DB 설치, 64bit 클라이언트 OS 환경에서는 64bit OLE DB 설치 방법을 설명하며 64bit 클라이언트 OS환경에서 32bit OLE DB를 설치하는 방법은 “2.2.4. Windows 64bit에 32bit OLE DB 설치”를 참고한다.

2.2.1. OLE DB 설치 파일

클라이언트 인스톨러나 OLEDB 인스톨러의 경우 GUI 환경으로 설치 및 등록을 자동으로 진행할 수 있다. 만약 수동으로 진행하려면 Tibero 서버가 설치된 환경에서 설치하고자 하는 클라이언트에 맞게 설치 파일을 가져온다. 바이너리 위치에 관한 자세한 내용은 “1.2.1. ODBC 설치 파일””를 참고한다.

다음은 바이너리 파일에 대한 설명이다.

파일명	설명
tbprov5.dll	OraOLEDB의 데이터 타입 spec을 맞춘 바이너리로 Provider를 등록할 때 사용한다.
msdtb5.dll	MSDAORA의 데이터 타입 spec을 맞춘 바이너리로 Provider를 등록할 때 사용한다.
Tibero.DbAccess.dll	.Net 환경에서의 지원을 위해 추가된 바이너리로 .NET을 연동할 때 사용한다.
EntLibContrib.Data.Tibero.dll	MS Enterprise Library 지원을 위해 추가된 바이너리로 MS Enterprise Library를 연동할 때 사용한다.

참고

1. Tibero 5 r67771 이전 버전의 경우 tbprov.dll, msdtb.dll 파일을 사용한다.

2. 확장자가 pdb(Program Database)인 경우는 디버깅 작업을 위해서 사용하는 파일이다. 릴리즈 버전에 따라 포함되어 있지 않는 경우도 있다.

2.2.2. OLE DB Provider 등록

OLE DB Provider 등록 순서는 다음과 같다.

1. **바이너리 복사**

   Tibero 버전에 따라 OLE DB 바이너리를 복사하는 위치가 다르다. 버전에 맞게 OLE DB 바이너리를 위치한다.

   - Tibero 4 SP1 및 이전 버전

     %WINDIR%\system32(예 : c:\windows\system32)

   - Tibero 5 이상

     임의의 위치로 가능하다. 단, 미리 설치된 ODBC 바이너리가 있는 디렉터리에 위치해야 한다.

2. **Provider 등록 및 해제**

   - 등록

     command 창을 열어 OLE DB 바이너리를 위치한 곳으로 이동한 후 명령어를 실행한다. 만약 Windows 7 이상일 경우 command 창을 관리자 권한으로 실행한다.

     – 등록 방법

     ```
     regsvr32 <dllname>
     ```

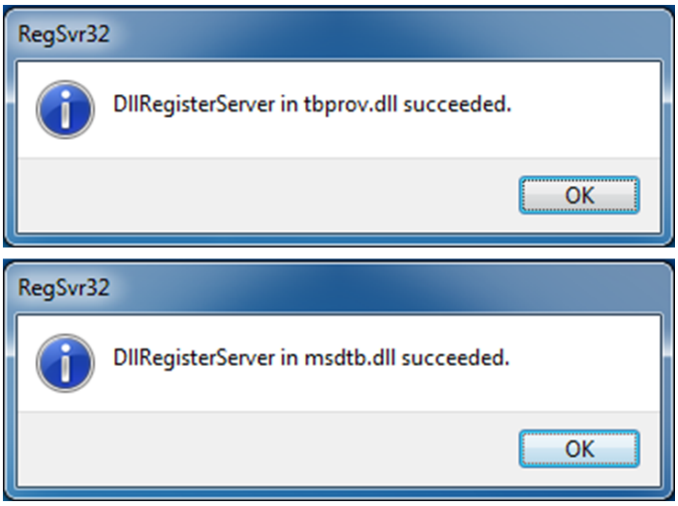
항목	설명
<dllname>	등록할 dll 파일명을 입력한다. – Tibero 4 SP1 또는 Tibero 5 r67770 이전 • tbprov.dll : OraOLEDB의 데이터 타입 spec • msdtb.dll : MSDAORA의 데이터 타입 spec – Tibero 5 r67770 이후 • tbprov5.dll : OraOLEDB의 데이터 타입 spec • msdtb5.dll : MSDAORA의 데이터 타입 spec

     – 등록 예

     ```
     c:\TmaxSoft\win64>regsvr32 tbprov5.dll
     c:\TmaxSoft\win64>regsvr32 msdtb5.dll
     ```

등록을 하면 아래와 같은 팝업창이 나타난다.

[그림 2.3] OLE DB 등록팝업



- 해제

Provider 해제가 필요한 경우 아래와 같이 수행한다. 만약 Windows 7 이상일 경우 command 창을 관리자 권한으로 실행한다.

– 해제 방법

```
regsvr32 /u <dllname>
```

항목	설명
<dllname>	기존에 이미 등록이 된 dll 파일 중에서 해제할 dll 파일명을 입력한다. – Tibero 4 SP1 또는 Tibero 5 r67770 이전 • tbprov.dll : OraOLEDB의 데이터 타입 spec • msdtb.dll : MSDAORA의 데이터 타입 spec – Tibero 5 r67770 이후 • tbprov5.dll : OraOLEDB의 데이터 타입 spec • msdtb5.dll : MSDAORA의 데이터 타입 spec

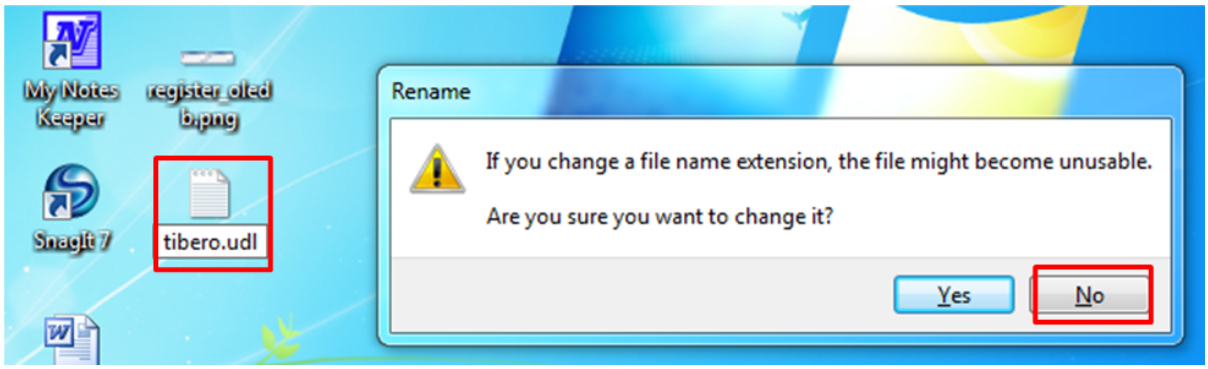
– 해제 예

```
c:\TmaxSoft\win64>regsvr32 /u tbprov5.dll
c:\TmaxSoft\win64>regsvr32 /u msdtb5.dll
```

3. **Provider 등록 확인**

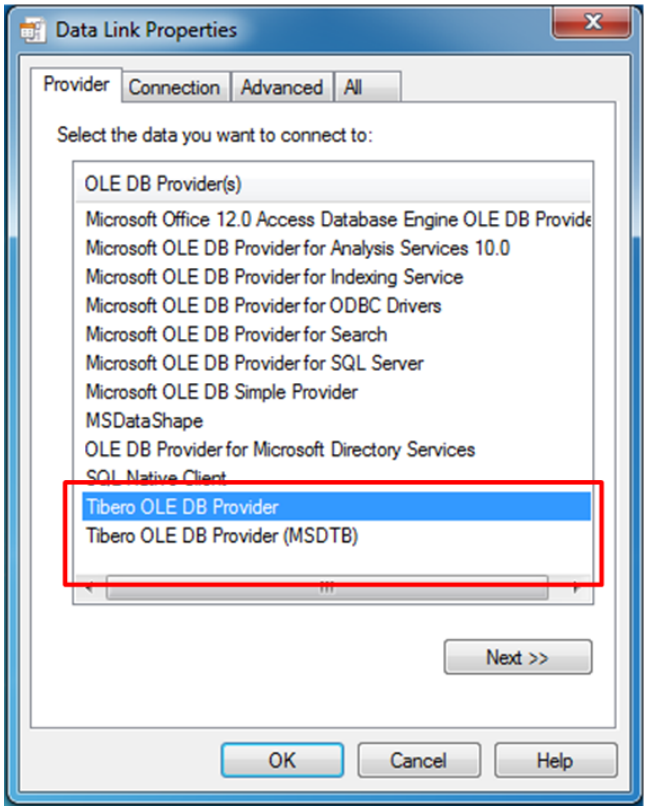
바탕화면에 tibero.udl 파일을 새로 생성한다.

[그림 2.4] udl 파일 생성



생성한 tibero.udl 파일을 더블클릭한 후 **[Provider]** 탭을 선택하면 Tibero OLE DB Provider를 확인할 수 있다.

[그림 2.5] OLE DB 등록 확인



2.2.3. OLE DB 연결

Tibero OLE DB를 사용하는 애플리케이션에서 연결 문자열(Connection String)을 사용하는 방식과 Provider 테스트 방식에 대해서 설명한다.

연결 문자열을 사용하는 방식

- 사용 방법

```
Provider=<공급자 이름>;Data Source=<데이터 원본 이름>;User ID=<접속 사용자 ID>;
        Password=<접속 패스워드>;Updatable Cursor=<Updateable Cursor 사용여부>;
        OLE DB Services=<Connection Pooling 사용여부>
```

항목	설명
<공급자 이름>	Tibero 버전 및 리비전에 따라 공급자 이름이 조금씩 다르다. – Tibero 4 SP1 또는 Tibero 5 r67770 이전 • tbprov.Tbprov 또는 tbprov.Tbprov.1 : OraOLEDB의 데이터 타입 spec • tbprov.MSDTB 또는 tbprov.MSDTB.1 : MSDAORA의 데이터 타입 spec – Tibero 5 r67770 이후(Tibero 4, Tibero 5 용 OLE DB를 같은 클라이언트에서 동시에 사용하기 위해서는 해당 리비전 이후를 사용한다) • tbprov.Tbprov 또는 tbprov.Tbprov.5 : OraOLEDB의 데이터 타입 spec • tbprov.MSDTB 또는 tbprov.MSDTB.5 : MSDAORA의 데이터 타입 spec
<데이터 원본 이름>	데이터 원본 이름에는 아래와 같은 항목이 있다. 단, Tibero 버전에 따라 일부 다른 정보가 들어가기도 한다. – ODBC 데이터 원본 관리자에 등록된 이름 – IP, PORT(Tibero 4 SP1) 또는 IP, PORT, DBNAME(Tibero 5)
<접속 사용자 ID>	Tibero 서버에 접속할 사용자 이름이다.
<접속 패스워드>	Tibero 서버에 접속할 패스워드이다.
<Updateable Cursor 사용여부>	Updatable Cursor 사용여부를 설정한다. (기본값 : False) – True : 사용한다. – False: 사용하지 않는다.
<Connection Pooling 사용여부>	Connection Pooling 기능 사용여부를 설정한다. (기본값 : -1) – -1 : 사용한다. – -2 : 사용하지 않는다.

- 사용 예

```
Provider=tbprov.Tbprov.5;Data Source=t5;User ID=dbtech;Password=dbtech;
Updatable Cursor=True;OLE DB Services=-2
```

Provider 테스트

앞절에서 만든 tibero.udl 파일로 연결 테스트를 할 수 있으며 아래와 같이 2가지 방식으로 가능하다.

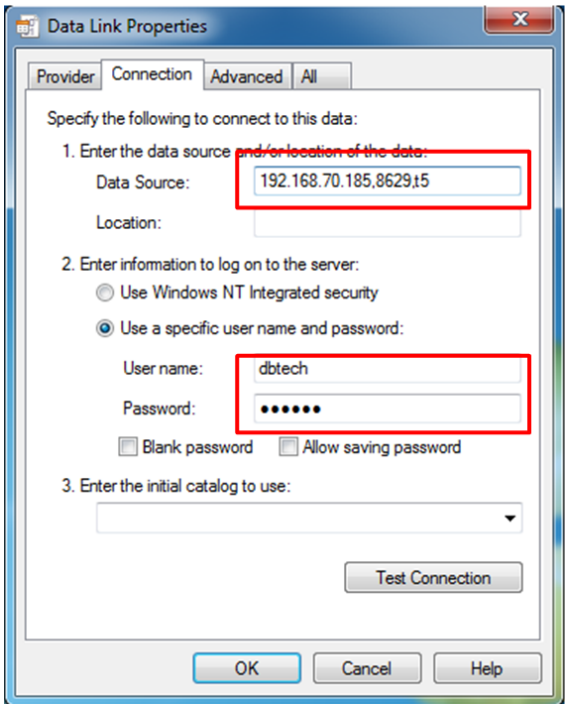
- IP, PORT 등의 정보를 이용한 직접 연결 방식

  데이터 원본에 IP, PORT, DB NAME 순으로 설정하여 접속한다. (예 : 192.168.70.185,8629,t5)

  만약 Tibero 4 SP1 및 이전 버전의 경우 DB NAME 정보없이 IP, PORT만으로 접속이 가능하다.

  (예 : 192.168.70.185,8629)

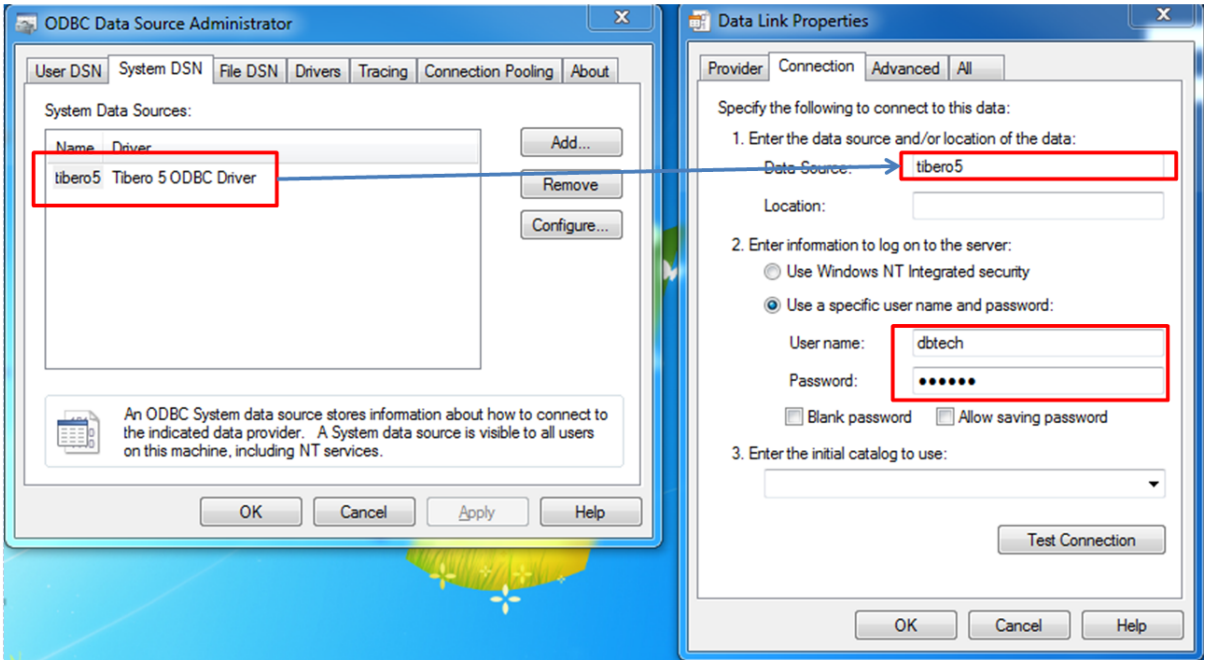
  [그림 2.6] OLE DB 직접 연결 방식



- ODBC 원본 관리자의 DSN 또는 tbdsn.tbr의 Alias를 이용한 방식

  데이터 원본에 ODBC 원본 관리자의 DSN 또는 "$TB_HOME/client/config/tbdsn.tbr"의 Alias를 이용하여 Tibero에 접속이 가능하다. 해당 Alias 정보를 찾는 방법은 **[ODBC 원본 관리자]** > **[tbdsn.tbr의 Alias]** 순서대로 찾는다.

[그림 2.7] OLE DB DSN 이용 방식

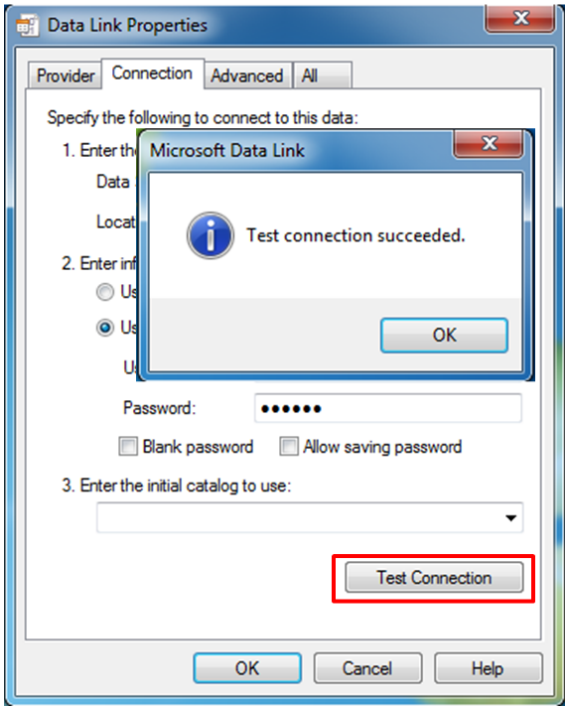


참고

ODBC 원본 관리자 등록 방법은 “1.2.3. ODBC 연결”의 "DSN을 등록하는 방식"을 참고한다.

아래와 같이 테스트 버튼을 눌러서 정상 접속되는지 확인한다.

[그림 2.8] OLE DB 연결 테스트



2.2.4. Windows 64bit에 32bit OLE DB 설치

32bit ODBC가 미리 설치 된 후 진행해야 한다.

1. **바이너리 복사**

   Tibero 버전에 따라 OLE DB 바이너리를 복사하는 위치가 다르다. 버전에 맞게 OLE DB 바이너리를 위치한다.

   - Tibero 4 SP1 및 이전버전

     %WINDIR%\SysWOW64(예 : c:\windows\SysWOW64)

   - Tibero 5 이상

     임의의 위치로 가능하다. 단, 미리 설치된 32bit ODBC 바이너리가 있는 디렉터리에 위치해야 한다.

2. **Provider 등록 및 해제**

   - 등록

     command 창을 열어 "%WINDIR%\SysWOW64" 폴더로 이동한 후 명령어를 실행한다. 만약 Windows 7 이상일 경우 command 창을 관리자 권한으로 실행한다.

     – 등록 방법

     ```
     regsvr32.exe <dllname>
     ```

항목	설명
<dllname>	등록할 32bit dll 파일의 위치를 절대경로로 입력한다. – Tibero 4 SP1 또는 Tibero 5 r67770 이전 • tbprov.dll : OraOLEDB의 데이터 타입 spec • msdtb.dll : MSDAORA의 데이터 타입 spec – Tibero 5 r67770 이후 • tbprov5.dll : OraOLEDB의 데이터 타입 spec • msdtb5.dll : MSDAORA의 데이터 타입 spec

     – 등록 예

     ```
     c:\Windows\system32>cd %WINDIR%\SysWOW64
     c:\Windows\SysWOW64>regsvr32.exe c:\TmaxSoft\win32\tbprov5.dll
     c:\Windows\SysWOW64>regsvr32.exe c:\TmaxSoft\win32\msdtb5.dll
     ```

- 해제

  command 창을 열어 "%WINDIR%\SysWOW64" 폴더로 이동한 후 명령어를 실행한다. 만약 Windows 7 이상일 경우 command 창을 관리자 권한으로 실행한다.

  – 해제 방법

```
regsvr32.exe /u <dllname>
```

항목	설명
<dllname>	해제할 32bit dll 파일의 위치를 절대경로로 입력한다. – Tibero 4 SP1 또는 Tibero 5 r67770 이전 • tbprov.dll : OraOLEDB의 데이터 타입 spec • msdtb.dll : MSDAORA의 데이터 타입 spec – Tibero 5 r67770 이후 • tbprov5.dll : OraOLEDB의 데이터 타입 spec • msdtb5.dll : MSDAORA의 데이터 타입 spec

  – 해제 예

```
c:\Windows\system32>cd %WINDIR%\SysWOW64
c:\Windows\SysWOW64>regsvr32.exe /u c:\TmaxSoft\win32\tbprov5.dll
c:\Windows\SysWOW64>regsvr32.exe /u c:\TmaxSoft\win32\msdtb5.dll
```

3. **Provider 등록 확인**

   Provider 등록 확인은 아래와 같은 순서로 진행한다.

   a. C:\ 디렉터리에 tibero.udl 파일을 생성한다.

   b. command 창을 열고 **rundll32.exe** 명령어를 실행한다. 명령어를 실제 사용할 때는 한줄로 입력한다.

   - 사용 방법

```
C:\Windows\syswow64\rundll32.exe
"C:\Program Files (x86)\Common Files\System\Ole DB\oledb32.dll"
,OpenDSLFile <udl 파일>
```

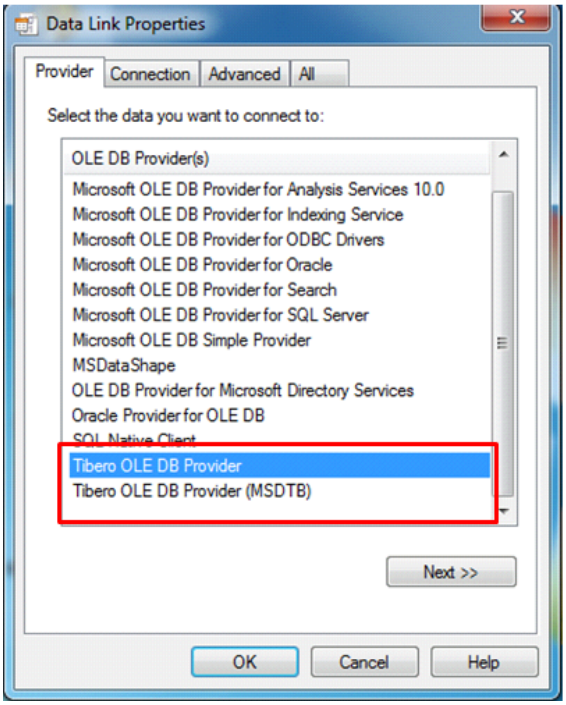
항목	설명
<udl 파일>	C:\ 디렉터리에 생성한 udl 파일의 경로 및 파일명을 입력한다.

   - 사용 예

```
C:\Windows\syswow64\rundll32.exe
"C:\Program Files (x86)\Common Files\System\Ole DB\oledb32.dll"
,OpenDSLFile C:\tibero.udl
```

c. Provider가 등록된 것을 확인한다.

[그림 2.9] 32bit용 OLE DB 등록 확인



4. **OLE DB 연결**

OLE DB 연결에 대한 자세한 내용은 “2.2.3. OLE DB 연결”을 참고한다.

2.3. Tibero OLE DB 연동

OLE DB를 사용하는 인터페이스에 대하여 연동하는 방법을 설명한다.

해당 인터페이스를 사용하는 서버에 미리 Tibero OLE DB가 설치되어야 한다.

2.3.1. ADO(ActiveX Data Objects)

Microsoft사에서 개발한 데이터베이스 연동용 인터페이스로 ADO 컴포넌트를 사용해 데이터베이스에 접근 후 여러 작업을 할 수 있다.

– 데이터베이스에 접근하기 위한 COM 객체들의 모음이다.

– 사용자의 프로그래밍 언어와 OLE DB사이를 연결하는 계층이다.

– Windows 운영체제를 설치할 때 자동으로 설치된다.

– ASP, PHP, VB 등의 환경에서 사용 가능하다.

– 사용자가 복잡한 OLE DB API 대신 ADO의 단순화된 API로 용이하게 개발이 가능하다.

– 각종 Script 언어와 연동되어 있으므로 다양한 환경에서의 개발이 가능하다.

다음은 ADO에서 사용하는 주요 객체이다.

- Connection

  데이터베이스를 연결할 때 사용하는 객체이다.

```
Set conn=Server.CreateObject(“ADODB.Connection”);
```

- Recordset

  조회 쿼리를 보내 그 쿼리 결과를 받을 때 사용한다.

```
Set rs = Server.CreateObject(“ADODB.Recordset”) rs.open “select * from emp”,conn
```

- Command

  이 객체만 데이터베이스 연결, 명령, 쿼리 실행 등이 가능하고 주로 프로시저 실행이나 매개변수가 포함된 쿼리를 실행할 때 사용한다.

```
Set Cmd = Server.CreateObject("ADODB.Command") with Cmd .ActiveConnection =
objConn .CommandType = adCmdStoredProc
.CommandText = "ADD"
…
End with
Set Cmd = Nothing
```

- Transaction

  트랜잭션을 처리할 때 사용 트랜잭션은 Connection객체의 .BeginTrans와 .CommitTrans 메소드 호출 사이에 존재한다.

```
myConnection.BeginTrans
While Not myRecordset.EOF
  mcounter = mcounter + 1
  myRecordset.Update
  myRecordset.MoveNext
Wend

myConnection.CommitTrans
```

```
myRecordset.Close
myConnection.Close
```

- Property

  ADO 객체의 동적인 속성을 가지고 있다.

```
set prop=Server.CreateObject("ADODB.Property")
for each prop in rs.Properties
  response.write("Attr:" & prop.Attributes & "<br>")
  response.write("Name:" & prop.Name & "<br>")
  response.write("Value:" & prop.Value & "<br>")
next
```

- Field

  ADO Field 객체는 Recordset 객체의 컬럼에 대한 정보를 가지고 있다.

```
set f=Server.CreateObject("ADODB.Field")

for each f in rs.Fields
  response.write("Attr:" & f.Attributes & "<br>")
  response.write("Name:" & f.Name & "<br>")
  response.write("Value:" & f.Value & "<br>")
Next
```

- Parameter

  Stored Procedure 또는 쿼리에서 사용되는 파라미터의 정보를 제공한다. (파라미터 타입 : input,output, input/output, return)

```
set comm=Server.CreateObject("ADODB.Command")
set para=Server.CreateObject("ADODB.Parameter")

para.Type=adVarChar
para.Size=20
para.Direction=adParamInput
para.Value=username

comm.Parameters.Append para
```

- Error

  실행할 때 발생한 에러를 Errors collection에 저장한다.

```
for each objErr in objConn.Errors
 response.write("<p>")
 response.write("Description: ")
```

```
 response.write(objErr.Description & "<br>")
 response.write("Help context: ")
 response.write(objErr.HelpContext & "<br>")
 response.write("Help file: ")
 response.write(objErr.HelpFile & "<br>")
 response.write("Native error: ")
 response.write(objErr.NativeError & "<br>")
 response.write("Error number: ")
 response.write(objErr.Number & "<br>")
 response.write("Error source: ")
 response.write(objErr.Source & "<br>")
 response.write("SQL state: ")
 response.write(objErr.SQLState & "<br>")
 response.write("</p>")
next
```

- Record

  Recordset, directory, file에서 하나의 행(row)을 가리킨다.

```
countfields=rec.Fields.Count
```

Cursor Location

다음은 Cursor 서비스 위치를 가르키는 속성값이며 Connection 또는 Recordset 단위로 설정 가능하다.

속성값	설명
adUseNode	OBSOLETE(하위 호환을 위해서만 존재한다) Value 1
adUseServer	사용자가 row를 이동할 때마다 OLE DB로부터 조금씩 데이터를 가져오는 방식으로 row를 일부만 가져올 경우(전체 메모리 사용량이 적음)에 유리하고 접속을 끊은 후에는 데이터를 가져올 수 없다. Updatable Cursor가 지원된다. Value 2(기본값)
adUseClient	OLE DB로부터 전체 데이터를 가져와 ADO단에서 저장해 두었다가 ADO 함수를 통하여 row를 이동할 때 바로 꺼내어 쓰는 방식으로 이미 필요한 데이터를 가져온 상태이므로 접속을 끊어도 데이터에 접근이 가능하다. Updatable Cursor는 지원되지 않는다. Value 3

사용 예제(ado_query.vbs)

다음은 ADO를 사용하는 VBScript 예제이다.

```
Dim conn, rs, sql
Set conn = CreateObject("ADODB.Connection")
conn.Open "Provider=tbprov.Tbprov.5;Data Source=t5;User ID=dbtech;Password=dbtech;"
conn.CursorLocation = adUseClient      'adUseServer = 2, adUseClient = 3

Set rs = CreateObject("ADODB.RecordSet")
sql = "select sysdate from dual"
rs.Open sql, conn

While Not rs.EOF
    WScript.Echo rs("sysdate")
    rs.MoveNext
Wend

rs.Close()

Set rs = Nothing
Set conn = Nothing
```

2.3.2. ADO.NET

ADO.NET은 .NET 개발자에게 데이터 액세스 서비스를 노출하는 클래스 집합이며 .NET 응용 프로그램은 ADO.NET을 통해 데이터소스에 연결하여 데이터를 검색, 조작 및 업데이트 할 수 있다.

– 기존의 ADO와 비교할 때 ADO.Net은 확장성과 상호 운영성이 개선된 ADO이다.

– XML을 보편적인 데이터 전송 형식으로 사용한다.

– C#, C++, VB.NET을 지원한다.

다음은 ADO.NET에서 사용하는 주요 객체이다.

객체	설명
Connection	가장 먼저 사용되는 개체로 데이터 원본에 대한 기본적인 연결을 제공한다.
Command	데이터베이스에 수행하는 명령(**"select * from emp"**)을 대표한다. OLE DB인 경우 OleDbCommand이다.
DataReader	데이터 원본으로부터 forward only, read only 등의 데이터를 빠르고 손쉽게 얻을 수 있는 개체이다. 단순히 데이터를 읽는 경우라면 이 개체가 가장 좋은 성능을 낸다. OLE DB인 경우 OleDbDataReader이다.
DataAdapter	데이터 원본에 대한 다양한 작업(갱신, DataSet 채우기)을 수행하는 법용적인 클래스이다. OLE DB인 경우 OleDbDataAdapter이다.

객체	설명
DataSet	응용 프로그램 안에서 하나의 단위로 참조되는 관련된 테이블들의 집합을 대표한다. 예를들어 Customers, Orders, Products의 모든 고객들과 그들이 주문한 제품들은 하나의 DataSet에 담기는 것이 가능하고 이 개체를 이용하여 원하는 데이터를 각 테이블로부터 빨리 가져와 서버와 연결이 끊어진 상태에서 데이터를 변경하고 한번의 명령으로 변경된 것들을 데이터베이스 서버에 저장할 수 있다.

네임스페이스의 사용

C#에서 사용한다는 가정하에 공급자에 따른 네임스페이스 사용법이다

- ADO.NET 클래스들이 있는 System.Data NameSpace에 참조를 정의한다.

```
using System.Data;
```

- OLE DB .NET 공급자를 위한 using 문이다.

```
using System.Data.OleDb;
```

- Tibero .Net 공급자를 위한 using 문이다

```
using Tibero.DbAccess;
```

- SQL Server .Net 공급자를 위한 using 문이다.

```
using System.Data.SqlClient;
```

- Oracle 전용 Data Provider for .Net의 경우 using 문이다.

```
using System.Data.OracleClient;
```

사용 예제(test_oledb.cs)

다음은 ADO.NET를 사용하는 C# 예제이다.

```
class Class1 {
     static void Main(string[] args) { Class1 c1 = new Class1(args); }

     public Class1(string[] args)
     {
         OleDbConnection conn = new OleDbConnection("Provider=tbprov.Tbprov; …“);
         conn.Open();

         OleDbCommand cmd = new OleDbCommand();
         cmd.Connection = conn;
         cmd.CommandText = "select * from test order by a";
```

```
        OleDbDataAdapter oda = new OleDbDataAdapter(cmd);
        DataSet dset = new DataSet();
        oda.Fill(dset);

        DataTable table = dset.Tables[0];
        DataRow[] rows = table.Select();
        Console.WriteLine(rows[0][“c1"].ToString());

        conn.Close();
    }
}
```

2.3.3. Enterprise Library 연동

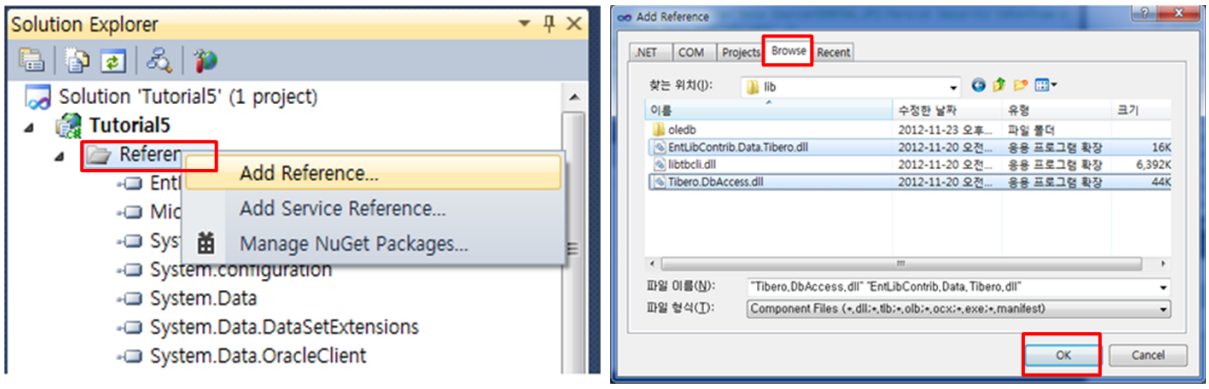
Enterprise Library는 MS에서 제공하는 오픈소스 프레임워크이다. Enterprise Library에 대한 자세한 내용은 http://msdn.microsoft.com/en-US/library/ff632023.aspx을 참고한다.

Tibero OLE DB 및 Enterprise Library가 설치 되었다면 연동 순서는 아래와 같다.

1. 프로젝트 내에 Tibero DLL 파일 추가

   Visual Studio의 프로젝트내에서 **[Add Reference]** > **[Browse]**탭에서 Tibero.DbAccess.dll, EntLibContrib.Data.Tibero.dll을 선택한다.

   [그림 2.10] Visual Studio 에 Tibero DLL 참조추가



2. 소스 내에 네임스페이스 추가

   필요할 경우 소스 내에 아래와 같이 네임스페이스를 추가한다.

```
using Tibero.DbAccess;
using EntLibContrib.Data.Tibero;
```

3. 관련 설정 추가

Enterprise Library의 설정 파일에 Tibero OLE DB에 관련 내용을 설정한다. 아내 내용 중에 설정 정보 중에 <providerMappings>, <connectionStrings> 부분을 참조한다.

```
<configuration>
  <configSections>
    <section name="dataConfiguration"
             type="Microsoft.Practices.EnterpriseLibrary.Data....생략" />
  </configSections>
  <dataConfiguration defaultDatabase="tibero">
    <providerMappings>
    <add databaseType=
          "EntLibContrib.Data.Tibero.TiberoDatabase,EntLibContrib.Data.Tibero,
           Version=1.0.0.0,Culture=neutral, PublicKeyToken=null"
         name="Tibero.DbAccess" />
    </providerMappings>
  </dataConfiguration>
  <connectionStrings>
    <add name="tibero"
         connectionString="Provider=tbprov.Tbprov.5;Data Source=t5
                          ;User ID=dbtech;Password=dbtech;Pooling=True
                          ;Max Pool Size=50;Min Pool Size=2;Connection Lifetime=0"
         providerName="Tibero.DbAccess" />
  </connectionStrings>
</configuration>
```

참고

1. <add databaseType="EntLibContrib.Data~" 부분은 한줄로 작성한다.
2. <connectionStrings> 부분은 고객사 상황에 맞게 적용한다.

2.4. OLE DB 전환

본 절에서는 Oracle OLE DB를 전환하는 내용과 예제를 설명한다.

2.4.1. Oracle OLE DB

오브젝트를 변경할 때 다음을 참고한다.

Oracle	Tibero
OracleConnection	OleDbConnectionTbr
OracleCommand	OleDbCommandTbr

Oracle	Tibero
OracleDataReader	OleDbDataReader
OracleDataAdapter	OleDbDataAdapterTbr
OracleDbType	OleDbTypeTbr
OracleParameter	OleDbParameterTbr
OracleCommandBuilder	OleDbCommandBuilderTbr
OracleTransaction	OleDbTransaction

OracleLob	OleDbYpeTbr.LongVarChar(CLOB 대응) 또는 OleDbTypeTbr.LongVarBinary(BLOB 대응)
	사용 방법은 reader에서 읽은 후 GetString으로 읽어들인다.
	`(예 : if (reader.Read())`
	`{ string clob = reader.GetString(0); .. })`

소스 변경 예제

다음은 Cursor에 관련된 예제이다.

- 예제1
  - Oracle

```
selectCmd.Parameters.Add("rs_Lst", OracleDbType.RefCursor).Direction \
= ParameterDirection.Output;
```

  - Tibero

```
OleDbCommandTbr selectCmd =
new OleDbCommandTbr();
selectCmd.Connection = oConn;
selectCmd.CommandType = CommandType.StoredProcedure;
selectCmd.CommandText = "PKG.Get_List";
selectCmd.Parameters.Add("rs_Lst", OleDbTypeTbr.Cursor).Direction \
= ParameterDirection.Output;
```

- 예제2
  - Oracle

```
OracleParameter a =
new OracleParameter("i_fromdate", fromdate);
a.Direction = ParameterDirection.Input;
a.OracleType = OracleType.VarChar;
```

```
OracleParameter b =
new OracleParameter ("o_rc", DBNull.Value);
b.Direction = ParameterDirection.Output;
b.OracleType = OracleType.RefCursor;

selectCmd.Parameters.Add(a);
selectCmd.Parameters.Add(b);
```

- Tibero

```
OleDbParameterTbr a =
new OleDbParameterTbr("i_fromdate", fromdate);
a.Direction = ParameterDirection.Input;
a.OleDbType = OleDbTypeTbr.VarChar;

OleDbParameterTbr b =
new OleDbParameterTbr("o_rc", DBNull.Value);
b.Direction = ParameterDirection.Output;
b.OleDbType = OleDbTypeTbr.Cursor;

selectCmd.Parameters.Add(a);
selectCmd.Parameters.Add(b);
```

Named Parameter 기능(.NET Tibero OLE DB provider에서 지원)

Tibero에서 Named Parameter 기능을 사용한 예제이다.

```
OleDbCommandTbr oCmd = new OleDbCommandTbr();
oCmd.Connection = oConn;
oCmd.CommandType = CommandType.Text;
oCmd.CommandText = "insert into aaa(aa,bb) values (:aa,:bb)";
oCmd.Parameters.Add(new OleDbParameterTbr("bb", "bb"));
oCmd.Parameters.Add(new OleDbParameterTbr("aa", "aa"));
```

2.5. 문제 해결

특정 문제가 발생했을 때 해결하는 방법을 설명한다.

2.5.1. 로그 발생

문제가 발생할 경우 Tibero OLE DB에서 특정 환경변수를 적용하여 로그를 발생한다. 로그를 발생시키는 환경변수의 경우 일부 성능이 느려질 수 있다. 따라서 문제가 있을 때 일시적으로만 사용한다. 참고로 Tibero OLE DB 경우는 내부적으로 Tibero ODBC를 사용하므로 ODBC 환경변수를 적용하여 발생한 로그, OLE DB 로그를 함께 발생시킨다.

환경변수명	설명
TB_OLEDB_LOG	로그를 출력하게 하는 환경변수이다. 아래와 같은 값을 설정 가능하다. (3 또는 4를 주로 설정한다) – 1 : API 주요 함수 – 2 : 1 + API 비 주요 함수 – 3 : 2 + 내부 함수 – 4 : 동적 메모리 할당 및 해제
TB_OLEDB_LOG_DIR	로그를 생성하는 디렉터리를 설정한다. 설정하지 않을 경우 "C:\tboledb_날짜시간.log" 경로에 로그가 생성된다. Windows 7의 경우 C:\ 디렉터리에 파일을 생성하려면 관리자 권한이 필요하다. 따라서 해당 경로에 로그를 생성하려면 애플리케이션을 관리자 권한으로 실행한다.
FORCED_USE_WCHAR	1로 설정할 경우 Tibero OLE DB에서 varchar,char를 wchar로 강제 바인드 시킨다. (Tibero5 r70330 이후)

2.5.2. 에러 메시지 처리

수행 중 발생할 수 있는 에러 메시지에 대해서 설명한다.

Current cursor is not updatable(tberr 2135)

설명	Updatable Cursor가 비활성화 상태인 경우 또는 불가능한 쿼리(view를 정의한 쿼리에 join문이 있는 경우)인 경우 발생한다.
대응 방법	Updatable Cursor가 비활성화 상태인 경우 연결 문자열에 **Updatable Cursor=True**를 추가하고, Updatable Cursor가 불가능한 쿼리인 경우 대체할 수 있는 쿼리를 작성 하거나 직접 DML을 작성하여 수정한다.

2.5.3. 연동할 때 문제점

연동할 때 발생할 수 있는 문제점에 대해서 설명한다.

UTF8 글자 깨짐 현상

[Application] ↔ [Enterprise Library] ↔ [Tibero OLE DB] ↔ [Tibero Server(UTF8)]와 같은 구조(중간에 다른 프레임워크가 들어갈 수도 있음)에서 발생할 수 있는 문제점이다.

설명	DB에 저장된 UTF8 데이터를 애플리케이션에서 화면에 ASCII로(영문 Windows의 기본 캐릭터 셋) 출력하면서 깨지는 것으로 보통 char, varchar은 char로 nchar, nvarchar은 wchar로 바인드하기 때문에 발생한다.
대응 방법	아래의 방법 중 하나를 선택한다. – 컬럼타입 char, varchar을 nchar, nvarchar로 변환한다. – Tibero OLE DB에서 char, varchar를 wchar로 강제 바인드 시키는 환경변수를 다음과 같이 설정한다. (Tibero5 r70330 이후) `FORCED_USE_WCHAR=1`

2.6. 예제

본 절에서는 OLE DB를 이용한 예제를 설명한다.

2.6.1. ASP

조회

다음은 간단한 조회 예제이다.

```
<%@ Language=VBScript %>

<%Option Explicit%>

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<table style="font-size:10pt" border="1">
<%
Dim Con, Rs

Set Con = Server.CreateObject ("ADODB.Connection")

// 아래 코드 실제사용시에는 한 줄로 사용할 것
Con.Open "Provider=tbprov.Tbprov.5;Password=dbtech;Persist Security Info=True
          ;User ID=dbtech;Data Source=t5"


Set Rs = Con.Execute ("Select * From user_objects")
```

```
while not Rs.eof
%>
<TR>
<TD><%=Rs("OBJECT_NAME")%></TD>
<TD><%=Rs("OBJECT_ID")%></TD>
<TD><%=Rs("OBJECT_TYPE")%></TD>
<TD><%=Rs("OBJECT_TYPE_NO")%></TD>
<TD><%=Rs("CREATED")%></TD>
<TD><%=Rs("STATUS")%></TD>
<TD><%=Rs("TEMPORARY")%></TD>
</TR>
<%
rs.movenext
wend
Set Rs = Nothing
Con.Close
Set Con = Nothing
%>
</table>
</BODY>
</HTML>
```

Procedure 호출 및 LOB 처리

다음은 Procedure 및 LOB 관련 예제이다.

- 테스트 오브젝트 생성

```
CREATE TABLE NOTICE(seq number, contents clob);

CREATE TABLE NOTICE3(seq number, contents blob);


CREATE or REPLACE PROCEDURE UPDATE_NOTICE
(
  V_SEQ IN number
, V_CONTENTS IN CLOB
) AS
loba clob
BEGIN
  insert into notice(SEQ, CONTENTS) values(V_SEQ, EMPTY_CLOB());
  select contents into loba from notice where seq=v_seq;
  dbms_lob.write(loba,length(v_contents),1,v_contents);
COMMIT;
END;
/
```

● Procedure 사용

```
<%@ CodePage=51949 Language="VBScript"%>
<%
 Set objConn = server.CreateObject("ADODB.Connection")
  'objConn.Open "Provider=tbprov.Tbprov.5;Data Source=t5;User ID=dbtech;
  'Password=dbtech;"

 objConn.Open "Provider=MSDASQL;DSN=t5;UID=dbtech;PWD=dbtech;"

 Set cmd= Server.CreateObject("ADODB.Command")

 contents = "aaaaaaaaaaaaaaaaaaaaa"

 with cmd
  set  .ActiveConnection = objConn
        .CommandType   = 4
        .CommandText   = "UPDATE_NOTICE"
        .Parameters.Append .CreateParameter("@V_SEQ",3, 1, 4)
  .Parameters.Append .CreateParameter("@V_CONTENTS", 201, 1, Len(contents))
  .Parameters("@V_SEQ") = 4
  .Parameters("@V_CONTENTS") = contents
  .Execute
 end with

 Set cmd = Nothing
 objConn.Close
 response.Write("sql : " & Sql & "<br>")

%>
```

● CLOB 조회

```
<%
Dim Con, Rs, strQry
Set Con = Server.CreateObject ("ADODB.Connection")
Con.Open "Provider=MSDASQL;DSN=t5;UID=dbtech;PWD=dbtech;"
Set Rs = Server.CreateObject("ADODB.Recordset")


Function CLOBRead(pLen, seq)
   Dim K
   Dim strMok
   Dim strLast
   Dim strReturn
   Dim Res
   strMok    = int(Clng(pLen) / 2000)
```

```
   strLast    = int(strMok + 1)

   Set Res = Server.CreateObject("ADODB.Recordset")

   For K = 1 To strLast
    sQuery = ""
    sQuery = sQuery & " SELECT DBMS_LOB.SUBSTR(CONTENTS, 2000
                               , 2000 * (" & K & " - 1) + 1) MEMO
                        FROM NOTICE where seq=" & seq & vbCrLf
    Res.Open sQuery, Con
    strReturn = strReturn & Res("memo")
    Res.Close
   Next

   Set Res = Nothing
   CLOBRead = strReturn

End Function

strQry = "SELECT DBMS_LOB.GETLENGTH(contents) CONTENT_LENGTH
          FROM NOTICE WHERE seq=4"

Rs.Open strQry, Con, 3

content = CLOBRead(Rs.Fields("CONTENT_LENGTH"), 4)

%>

<%=content%>

<%
Set Rs = Nothing
Con.Close
Set Con = Nothing
%>
```

- BLOB 등록

```
<%
Function ReadByteArray(strFileName)
Const adTypeBinary = 1
Dim bin
Set bin = CreateObject("ADODB.Stream")
bin.Type = adTypeBinary
bin.Open
bin.LoadFromFile strFileName
ReadByteArray = bin.Read
```

```
End Function
Set objConn = server.CreateObject("ADODB.Connection")
Set cm = Server.Createobject("ADODB.Command")

// 아래 코드 실제사용시에는 한 줄로 사용할 것
objConn.Open "Provider=tbprov.Tbprov.5;Data Source=t5
              ;User ID=dbtech;Password=dbtech;"

strQry = "INSERT INTO NOTICE3 (SEQ, CONTENTS) VALUES (?, ?)"
response.Write("strQry : " & strQry & "<br>")
Dim seq
'notice3테이블에 입력할 seq필드 값
seq=44

cm.ActiveConnection = objConn
cm.CommandText = strQry
cm.Parameters.Append cm.CreateParameter("@SEQ", 3, 1, Len(seq), seq)
Dim fileURI
fileURI = "D:\tibero\binary\tibero5-bin-5.0-windows64-69860-opt-tested.tar.gz"
'ReadByteArray(파일이 위치한 절대경로)
cm.Parameters.Append cm.CreateParameter("@CONTENTS", 205, 1
                      ,LenB(ReadByteArray(fileURI)) ,ReadByteArray(fileURI))
'cm.CommandType = 1 <<= 사용 금지
cm.Execute
Set cmd = Nothing
objConn2.Close
Set objConn2 = Nothing
%>
```

2.6.2. ASP.NET

조회

다음은 간단한 조회 예제이다. (Windows 2008 Server IIS 확인)

```
<%@ Import Namespace="System.Data.OleDb" %>

<script  runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread

// 아래 코드 실제사용시에는 한 줄로 사용할 것
dbconn=New OleDbConnection("Provider=tbprov.Tbprov.5;User ID=dbtech;
                            Password=dbtech;Data Source=t5;Updatable Cursor=True;")
dbconn.Open()
```

```
sql="SELECT sysdate FROM dual"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
SystemDate.DataSource=dbread
SystemDate.DataBind()
dbread.Close()
dbconn.Close()
end sub
</script>

<!DOCTYPE html>
<html>
<body>

<form runat="server">
<asp:Repeater id="SystemDate" runat="server">

<HeaderTemplate>
<table border="1" width="30%">
<tr bgcolor="#b0c4de">
<th>date</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr bgcolor="#f0f0f0">
<td><%#Container.DataItem("sysdate")%> </td>
</tr>
</ItemTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>
```

2.6.3. C#

조회

다음은 간단한 조회 예제이다.

- 예제1

```
using System;
using System.Data;
using System.Data.OleDb;
using System.Xml.Serialization;

public class MainClass
{
 public static void Main ()
 {

  string strAccessConn = "Provider=tbprov.Tbprov.5;
  Data Source=t5;User ID=dbtech;Password=dbtech";
         Console.WriteLine("Connection.....");
  string strAccessSelect = "SELECT * FROM test3";

  DataSet myDataSet = new DataSet();
  OleDbConnection myAccessConn = null;
  try
  {
   myAccessConn = new OleDbConnection(strAccessConn);
  }
  catch(Exception ex)
  {
   Console.WriteLine("Error: Failed to create a database connection.
                      \n{0}", ex.Message);
   return;
  }

  try
  {

   OleDbCommand myAccessCommand =
    new OleDbCommand(strAccessSelect,myAccessConn);
   OleDbDataAdapter myDataAdapter =
    new OleDbDataAdapter(myAccessCommand);

   myAccessConn.Open();
   myDataAdapter.Fill(myDataSet,"test3");
```

```
  }
  catch (Exception ex)
  {
   Console.WriteLine("Error: Failed to retrieve the required data
                      from the DataBase.\n{0}", ex.Message);
   return;
  }
  finally
  {
   myAccessConn.Close();
  }

  DataTableCollection dta = myDataSet.Tables;
  foreach (DataTable dt in dta)
  {
   Console.WriteLine("Found data table {0}", dt.TableName);
  }


  Console.WriteLine("{0} tables in data set", myDataSet.Tables.Count);
  Console.WriteLine("{0} tables in data set", dta.Count);

  Console.WriteLine("{0} rows in Categories table",
                    myDataSet.Tables["test3"].Rows.Count);

  Console.WriteLine("{0} columns in Categories table",
                    myDataSet.Tables["test3"].Columns.Count);
  DataColumnCollection drc = myDataSet.Tables["test3"].Columns;
  int i = 0;
  foreach (DataColumn dc in drc)
  {

   Console.WriteLine("Column name[{0}] is {1}, of type {2}",i++
                      , dc.ColumnName, dc.DataType);
  }
  DataRowCollection dra = myDataSet.Tables["test3"].Rows;
  foreach (DataRow dr in dra)
  {
   Console.WriteLine("CategoryName[{0}] is {1}", dr[0], dr[1]);
  }

 }
}
```

- 예제2

```
using System;
using System.Data;
using System.Data.OleDb;

class TableAnalysis
{
    static void Main(string[] args)
    {
        //data source=name

// 아래 코드 실제 사용시에는 한 줄로 사용할 것
        string sql =
            "Provider=tbprov.Tbprov.5;User ID=dbtech;Password=dbtech;
  Data Source=t5;Persist Security Info=True";
        //location=ip,port
        //string sql = "Provider=tbprov.Tbprov.5;Password=dbtech;
 User ID=dbtech;Location=127.0.0.1,8629,t5;Persist Security Info=True";
        OleDbConnection conn = new OleDbConnection(sql);

        try
        {
            conn.Open();            //데이터베이스 연결
            OleDbCommand cmd = new OleDbCommand();
            cmd.CommandText = "select * from all_tables";   // 테이블
            cmd.CommandType = CommandType.Text;     //검색명령을 쿼리 형태로
            cmd.Connection = conn;

//select * from all_tables 결과
            OleDbDataReader read = cmd.ExecuteReader();
Console.WriteLine("***** ALL_TABLES 테이블 *****");
            for (int i = 0; i < read.FieldCount; i++)
            {
                Console.WriteLine("필드명 : {0} \n", read.GetName(i));
            }
            Console.WriteLine("총필드 개수는" + read.FieldCount);
            read.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine("에러발생{0}", ex.Message);
        }
        finally
        {
            if (conn != null)
            {
```

```
                conn.Close();       //데이터베이스 연결 해제
                Console.WriteLine("데이터베이스 연결 해제..");
            }
        }
    }
}
```

REFCURSOR

다음은 SYS_REFCURSOR를 사용하는 예제 소스이다.

- 테스트 오브젝트 생성

```
CREATE OR REPLACE PACKAGE pkg003
IS
  PROCEDURE get_forms(curs out SYS_REFCURSOR);
END;


CREATE OR REPLACE PACKAGE BODY pkg003
IS

    -- procedure implementation
    PROCEDURE get_forms(curs out SYS_REFCURSOR) IS
       BEGIN
          open curs FOR select sysdate FROM dual;
       END;
END;
```

- 예제 소스

```
string connectionString = System.Configuration.ConfigurationManager
                          .ConnectionStrings["tibero5_oledb"].ConnectionString;

OleDbConnectionTbr myConnection = new OleDbConnectionTbr(connectionString);

DataTable dt = new DataTable();
System.Data.OleDb.OleDbDataReader reader = null;
try{

                myConnection.Open();
                OleDbCommandTbr cmd = new OleDbCommandTbr();
                cmd.Connection = myConnection;
                cmd.CommandText = "pkg003.get_forms";
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.Add("curs", OleDbTypeTbr.Cursor).Direction
```

```
                                            = ParameterDirection.Output;

                reader = cmd.ExecuteReader();
                myDataGrid.DataSource = reader;
                myDataGrid.DataBind();
}
catch (Exception)
{ }
finally
{
                 reader.Close();
                 myConnection.Close();
}
```

제3장 JDBC 연결

본 장에서는 JDBC에 대한 개념과 Tibero JDBC의 드라이버 연동에 대해서 설명한다.

3.1. JDBC 개념

Java Database Connectivity의 약자로서 Java로 만들어진 클래스와 인터페이스로 이루어진 API이다. DBMS의 종류와 관련없는 독립적인 프로그래밍을 가능하게 해주며 JDBC는 java.sql, javax.sql 두 개의 패키지에 포함된다.

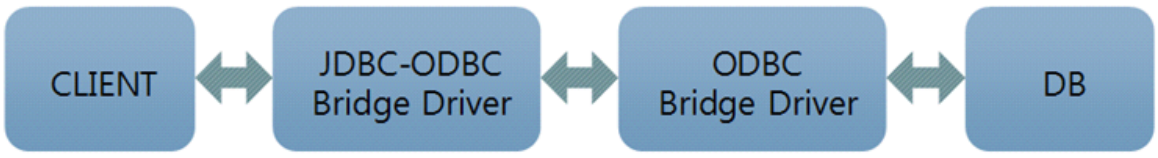
- java.sql은 데이터베이스에 접근하고 데이터를 검색하거나 업데이트 하는 핵심 JDBC API를 제공한다.
- javax.sql은 JDBC 클라이언트가 서버측의 데이터소스를 접근할 수 있게 하는 API를 제공한다.

JDBC Driver Types

- Type 1 : JDBC-ODBC Bridge Driver

  ODBC 같이 다른 Data Access API와 매핑하는 형태의 JDBC API를 구현한 것이다. 이것은 Native Client Library에 종속적으로 되는 경우가 많아 이식성에 제약이 있다. Sun의 JDBC-ODBC Bridge Driver가 이에 속하지만 JDBC 3.0의 지원이나 멀티 쓰레딩을 사용할 수 없는 등의 여러가지 제약을 지닌다.

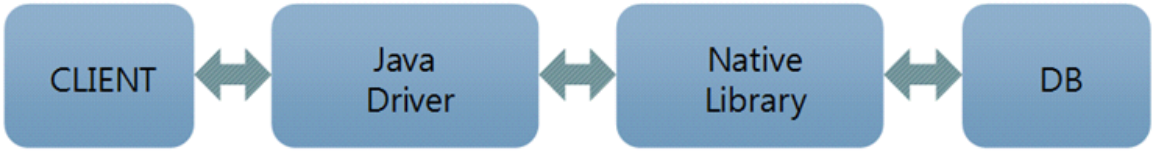
  [그림 3.1] TYPE1



- Type 2 : Native-API Driver

  Native Code와 Java Code가 혼합되어 구현이 되어있다. 주로 Interface만 Java인 경우가 많고 접근하는 데이터소스에 따라 각기 다른 Native Client Library가 필요하다. 그래서 이것을 Thick Driver 라고도 한다. Native Code로 인하여 이식성에 제한이 있다.

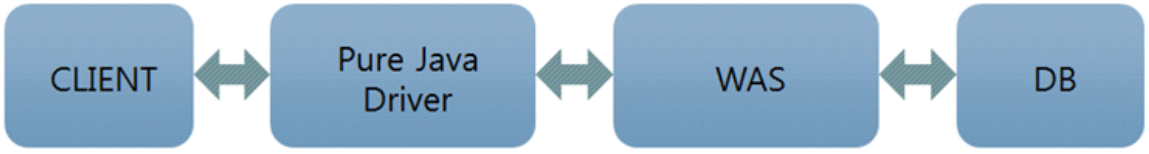
  [그림 3.2] TYPE2



- Type 3 : Net-Protocol Driver

  데이터베이스에 독립적인 프로토콜을 이용하는 미들웨어 서버와 통신하고 순수 자바 클라이언트를 사용하는 드라이버로 미들웨어 서버는 클라이언트의 요청을 데이터소스에 독립적인 프로토콜로 변환하여 사용한다. 이 경우 드라이버가 직접 데이터베이스를 제어하지 않고 미들웨어를 통하여 제어하기 때문에 유연성을 지닌다.

  [그림 3.3] TYPE3



- Type 4 : Native-Protocol Driver

  모두 자바로 구현되어 있어 플랫폼에 제한을 받지 않으며 별도의 클라이언트 소프트웨어가 없이 표준 자바 소켓을 이용하여 데이터소스와 직접 통신한다. 보통 Thin Driver라고 한다.

  [그림 3.4] TYPE4

3.2. Tibero JDBC

본 절에서는 JDBC 드라이버 설명과 연동하는 방법에 대해서 설명한다.

참고

Tibero에서는 현재 Type 4만 지원한다.

3.2.1. JDBC 드라이버

JDBC 드라이버는 "$TB_HOME/client/lib/jar" 경로에 존재한다.

다음은 드라이버 파일에 대한 설명이다.

- 파일명 규칙

```
tibero[DB_MAJOR_VERSION]-jdbc-[JDK_VERSION]-[DEBUG].jar
```

항목	설명
[DB_MAJOR_VERSION]	Tibero 제품의 메인 버전를 표시한다.
[JDK_VERSION]	JDBC 드라이버가 동작하는 JDK 버전을 표시한다. 기본형의 경우 생략한다.
[DEBUG]	문제가 발생할 때 로그를 발생하는 JDBC 파일(디버그 용도)인지 알려준다.

- 사용 예

파일명	설명
tibero5-jdbc.jar	JDK(또는 JRE) 1.6 이상에서 수행 가능한 드라이버 파일(기본형)이다.
tibero5-jdbc-dbg.jar	JDK(또는 JRE) 1.6 이상에서 수행 가능한 JDBC 파일(디버그 용도)이다.
tibero5-jdbc-14.jar	JDK(또는 JRE) 1.4 이상에서 수행 가능한 드라이버 파일(JRE 버전 문제가 발생할 경우 주로 사용)이다.
tibero5-jdbc-14-dbg.jar	JDK(또는 JRE) 1.4 이상에서 수행 가능한 JDBC 파일(디버그 용도)이다.

버전 확인

주로 JDBC 리비전 정보가 필요할 때 아래와 같은 방법으로 확인할 수 있다.

- 형식

```
java -jar [driver_name]
```

항목	설명
[driver_name]	JDBC 드라이버 파일명이다.

- 사용 예

```
$ java -jar tibero5-jdbc.jar

Tibero JDBC Driver 5.0   (Rev.71713M)

Tibero, Co. Copyright(C) 2011-. All rights reserved.
```

3.2.2. 드라이버 연동

Tibero JDBC를 가지고 서버와 연동할 때 연동할 때 다음의 클래스를 사용한다.

– Connection을 맺을 때 사용하는 클래스 이름 : com.tmax.tibero.jdbc.TbDriver

– 데이터소스를 사용할 때 클래스 이름 : com.tmax.tibero.jdbc.ext.TbConnectionPoolDataSource

– XA 데이터소스를 사용할 때 클래스 이름 : com.tmax.tibero.jdbc.ext.TbXADataSource

데이터베이스 URL

데이터베이스 URL은 Single 노드 구성과 TAC 노드 구성으로 나누어 설명한다.

- Single 노드 구성

  1개의 Tibero 서버로 구성된 경우이다.

  – 사용 방법

  ```
  jdbc:tibero:thin:@<ip>:<port>:<db_name>
  ```

항목	설명
<ip>	접속하려는 Tibero 서버 IP 주소이다.
<port>	접속하려는 Tibero 서버 포트 번호이다.
<db_name>	접속하려는 Tibero 서버 DB 이름이다.

  – 사용 예

  ```
  jdbc:tibero:thin:@127.0.0.1:8629:t5
  ```

- TAC 노드 구성

  2개로 구성된 TAC 서버일 경우이다.

  – 사용 방법

  ```
  jdbc:tibero:thin:@(description=
            (failover=on)(load_balance=on)
            (address_list=(address=(host=<node1_ip>)(port=<node1_port>))
                          (address=(host=<node2_ip>)(port=<node2_port>))
            )(DATABASE_NAME=<db_name>))
  ```

항목	설명
failover	연결이 끊어진 경우 자동으로 복구해주는 기능(on 또는 off 설정)으로 순차적 방식으로 새로운 서버에 접속하며 현재는 데이터베이스와의 연결만 복구한다.
load_balance	사용자 Connection을 여러 서버로 분산시키는 기능(on 또는 off 설정)으로 Dedicate 방식(=전용방식)으로 구성할 경우 해당 기능을 off 한다.
<node1_ip> *<node1_port>*	1번 노드에 대한 접속 정보이다. (IP 주소, 포트 번호)
<node2_ip> *<node2_port>*	2번 노드에 대한 접속 정보이다. (IP 주소, 포트 번호)
<db_name>	접속하려는 TAC의 DB 이름이다.

– 사용 예

```
jdbc:tibero:thin:@(description=
          (failover=on)(load_balance=on)
          (address_list=(address=(host=127.0.0.1)(port=8629))
                        (address=(host=127.0.0.2)(port=8629))
          )(DATABASE_NAME=t5))
```

3.3. JDBC 전환

본 절에서는 Oracle JDBC를 전환하는 내용을 설명한다.

3.3.1. Oracle JDBC

클래스를 변경할 때 다음을 참고한다.

Oracle	Tibero
oracle.jdbc.driver.OracleTypes.CURSOR	com.tmax.tibero.TbTypes.CURSOR
oracle.sql.BLOB	com.tmax.tibero.jdbc.TbBlob
oracle.sql.CLOB	com.tmax.tibero.jdbc.TbClob
oracle.jdbc.driver.OracleResultSet	com.tmax.tibero.jdbc.TbResultSet
oracle.jdbc.OracleCallableStatement	com.tmax.tibero.jdbc.TbCallableStatement

3.4. 문제 해결

특정 문제가 발생했을 때 해결하는 방법을 설명한다.

3.4.1. 로그 발생

Tibero JDBC를 사용하여 특정 문제가 발생했을 때 추가적인 로그를 발생하여 확인하는 방법이다. 로그가 추가적으로 발생하므로 일부 성능이 느려질 수 있다. 따라서 문제가 있을 때 일시적으로만 사용한다.

로그 발생은 다음과 같은 순서로 적용한다.

1. 드라이버 교체

   기존에 사용하던 JDBC 드라이버 파일을 tibero5-jdbc-dbg.jar와 같은 디버그 용도 파일로 교체한다.

2. 클라이언트 프로그램 재기동

   디버그 용도 파일을 가지고 동작할 수 있도록 프로그램을 재기동한다.

3. 로그 파일 위치 확인(해당 폴더 확인)

   – UNIX 계열 : /home/사용자 이름

   – Windows 계열 : C:\Documents and Settings\사용자 이름

4. 로그 파일 이름 확인

   tbjdbc-{yyMMdd}-{HHmmss}-{S}-{random}.log 형식으로 파일이 생성된다.

3.5. 예제

본 절에서는 JDBC를 이용하여 연결하는 기본적인 형태의 예제와 특수한 타입의 예제를 설명한다.

3.5.1. 기본 예제

다음은 JDBC를 이용한 연결 예제이다.

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.tmax.tibero.jdbc.ext.TbDataSource;

public class TiberoJDBC {

 public static void main(String[] args) throws SQLException {

     TbDataSource tds = new TbDataSource();
     tds.setURL("jdbc:tibero:thin:@127.0.0.1:8629:t5");
     tds.setUser("dbtech");
     tds.setPassword("dbtech");

     Connection conn = tds.getConnection();
     Statement stmt = conn.createStatement();

     String query =  "select table_name from user_tables where rownum < 10 " ;
     ResultSet rs = stmt.executeQuery(query);

     String sTname=null;

     int irows = 1;
     while ( rs.next() ) {
         sTname = rs.getString("table_name");
         System.out.println("row[" + irows + "] : " + sTname);
```

```
        irows++;
    }

    rs.close();
    stmt.close();
    conn.close();
 }
}
```

3.5.2. 타입 관련 예제

NVARCHAR 처리

com.tmax.tibero.jdbc.TbPreparedStatement에 NCharset을 지원하기 위해 다음과 같은 API를 제공한다.

- setNCharacterStream()
- setNClob()
- setNString()

다음은 NVARCHAR 처리의 예제이다.

```
ds = (DataSource)ctx.lookup("tibero");
conn = ds.getConnection();

ins_pstmt = (TbPreparedStatement)conn.prepareStatement(ins_query);
ins_pstmt.setString(1, "test");
ins_pstmt.setNString(2, "multinational character");
```

ARRAY

동일한 타입의 Primitive 값이 Array인 경우에만 지원이 가능하다.

- 테스트 오브젝트 생성

```
CREATE OR REPLACE PACKAGE TYPE_PKG
AS
    TYPE "TY_CHOICE_IDX_ARR" IS VARRAY(20000) OF CLOB;
END;
/

CREATE OR REPLACE PROCEDURE TB_ARRAY_TEST
(
   I_INPT_EXPC_DIV IN  VARCHAR2,
   ....
```

```
   O_CHOICE_PACKET OUT TYPE_PKG.TY_CHOICE_IDX_ARR   /* TYPE */
)

IS
  .....
  L_DATA                TYPE_PKG.TY_CHOICE_IDX_ARR;
  TMP_CLOB_RTN   CLOB;

BEGIN
  -- init
  L_DATA                  := TYPE_PKG.TY_CHOICE_IDX_ARR();
  TYPE AAA_SET IS TABLE OF CBT_CAT%ROWTYPE;
  AAA AAA_SET;

....

  FOR I IN AAA.FIRST .. AAA.LAST
  LOOP
   -- CLOB return function
   T_CHOICE_IDX_LIST := FN_CHOICE_LIST_ADD_01();
   O_CHOICE_IDX_LIST := O_CHOICE_IDX_LIST || T_DELIM_PIPE || T_CHOICE_IDX_LIST;
  END LOOP;
  L_DATA.EXTEND;
  L_DATA(L_DATA.COUNT) := O_CHOICE_IDX_LIST || 'FALSE'  || 'string test'
  O_CHOICE_PACKET := L_DATA;

END;
/
```

- 예제 소스(소스 전체는 아님)

```
CallableStatement   cstmt       = null;
Array packetArray = null;

callableSql = "begin ? :=TB_ARRAY_TEST(?,?,?,?,?); end;";

try{
    cstmt = (CallableStatement)con.prepareCall(callableSql);
    cstmt.setString(1, domainCd.trim());
    cstmt.setString(2, tsNo.trim());
    cstmt.setDouble(3, Double.parseDouble(verNo.trim()));
    cstmt.setString(4, inptExpcDiv);
    cstmt.setString           (5, choiceGrpNo);
    //out parameter
    cstmt.registerOutParameter(6, TbTypes.ARRAY, "TYPE_PKG.TY_CHOICE_IDX_ARR" );

    cstmt.execute();
```

```
    packetArray = cstmt.getArray(6);

    // Get the ARRAY object
    if(packetArray == null){
        throw new BCException("no data");
    }
    Object o = packetArray.getArray();
    clobs = (Clob[])o;
    choiceGrpPacketList = new ArrayList(clobs.length);
    Clob clob       = null;
    String packet   = null;
    long clobLength = 0;

    for( int i = 0 ; i < clobs.length; i++ ){
        clob       = clobs[i];
        clobLength = clob.length();
        packet = clob.getSubString(1, (int)clobLength);
        choiceGrpPacketList.add(JdbcCodeConvertor.toKor(packet));
    }
}catch(SQLException sqle){
    sqle.printStackTrace();
}finally{
    if( cstmt != null ){
        try{
            cstmt.close();
        }catch(SQLException sqle) {
            logger.error("CallableStatement close Error !");
            sqle.printStackTrace();
        }
    }
}
```

제4장 tbESQL 연결

본 장에서는 tbESQL에 대한 개념과 사용 방법에 대해서 설명한다.

4.1. tbESQL 개념

ESQL(Embedded SQL : 내장 SQL)은 프로그래밍 언어의 연산 능력과 SQL의 데이터베이스(Database)를 조작하는 능력을 결합하기 위한 방법이며 ANSI 및 ISO 표준으로 정의되어 있다.

tbESQL은 ESQL의 사용을 위해 Tibero가 제공하는 인터페이스이며 C와 COBOL을 제공한다.

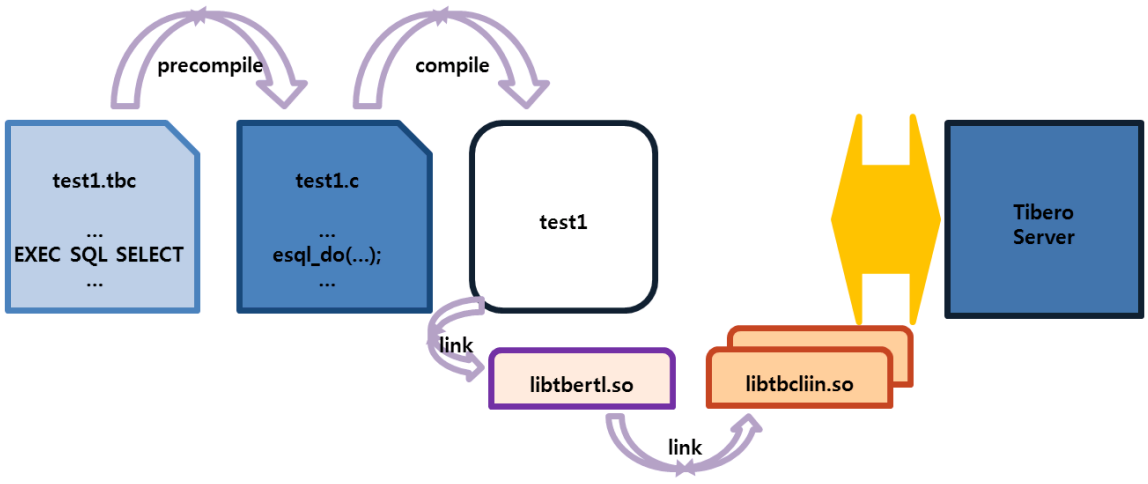
4.1.1. tbESQL 기본동작

tbESQL에서는 프리컴파일, 컴파일 및 링크 과정을 거친다. 처리가 완료된 실행 파일을 가지고 실행하게 되면 Tibero 서버에 접속하게 되는 것이다.

참고로 tbESQL은 내부적으로 tbCLI(=ODBC) 인터페이스를 사용한다. 따라서 tbCLI(=ODBC) 관련 환경 변수가 모두 적용 가능하다.

다음은 tbESQL의 기본동작을 나타내는 그림이다.

[그림 4.1] tbESQL 기본동작



4.2. tbESQL 사용

본 절에서는 tbESQL을 사용하기 위한 방법을 설명한다.

4.2.1. 관련 파일

다음은 tbESQL과 관련된 바이너리 파일에 대한 설명이다.

- **프리컴파일러 및 스크립트**

  프리컴파일할 때 사용하는 프리컴파일러 및 컴파일할 때 참조하는 스크립트로 구성되어 있다.

  바이너리의 위치는 다음과 같다.

Tibero 버전	OS	위치
Tibero 4 SP1 및 이전 버전	UNIX 계열	$TB_HOME/client/bin
	Windows 계열	%TB_HOME%\client\bin
Tibero 5 이상	UNIX 계열	$TB_HOME/client/bin
	Windows 계열	%TB_HOME%\bin

  다음은 바이너리 파일에 대한 설명이다.

파일명	설명
tbpc or tbpc.exe	tbESQL/C 프리컴파일러이다.
tbpcc	tbESQL/C 컴파일 관련 스크립트(참조용)이다.
tbpcb or tbpcb.exe	tbESQL/COBOL 프리컴파일러이다.
tbpcbc	tbESQL/COBOL 컴파일 관련 스크립트(참조용)이다.

- **설정 파일**

  OS별 바이너리의 위치는 다음과 같다.

OS	위치
UNIX 계열	$TB_HOME/client/config
Windows 계열	%TB_HOME%\client\config

  다음은 바이너리 파일에 대한 설명이다.

파일명	설명
tbpc.cfg	tbESQL/C 프리컴파일할 때 설정 파일이다.
tbpcb.cfg	tbESQL/COBOL 프리컴파일할 때 설정 파일이다.

파일명	설명
tbertl.cfg	tbESQL/COBOL 런타임용 설정 파일이다.

- **INCLUDE 파일**

  컴파일할 때 해당 파일 또는 디렉터리를 Include해야 한다.

  OS별 바이너리의 위치는 다음과 같다.

OS	위치
UNIX 계열	$TB_HOME/client/include
Windows 계열	%TB_HOME%\client\include

  다음은 바이너리 파일에 대한 설명이다.

파일명	설명
sqlca.h	ESQL 관련 규약(구조체, 매크로 등)이 정의되어 있고 컴파일할 때 해당 파일의 Include가 필수이다.
sqlda.h, sqlcpr.h	Dynamic SQL을 사용하는 방법 중 방법4를 사용할 때 필요한 헤더 파일이다.
SQLCA	ESQL/COBOL에서 필요한 헤더 파일이다.

- **동적 라이브러리**

  링크할 때 해당 파일이 존재하는 디렉터리에 대하여 설정이 필요하다.

  실행할 경우 동적 라이브러리를 찾는 경로의 환경변수 LD_LIBRARY_PATH(Linux), LIBPATH(AIX), SHLIB_PATH(HP)에 해당 디렉토리가 설정되어야 한다.

  OS별 바이너리의 위치는 다음과 같다.

OS	위치
UNIX 계열	$TB_HOME\client\lib
Windows 계열	%TB_HOME%\bin

  다음은 동적 라이브러리 관련 파일로 OS에 따라 파일명이 다르다.

OS	파일명
UNIX 계열	libtbecbpp.so, libtbecommon.so, libtbecpp.so, libtbertl.so, libtbpc.so, libtbpcb.so
Windows 계열	libtbcli.dll

4.2.2. 프리컴파일

프리컴파일러 바이너리를 이용하여 프리컴파일을 수행한다.

다음은 프리컴파일할 때 동작하는 순서이다.

1. Preprocess

   매크로 변환, 헤더 파일의 복사 등의 preprocess 과정을 수행한다.

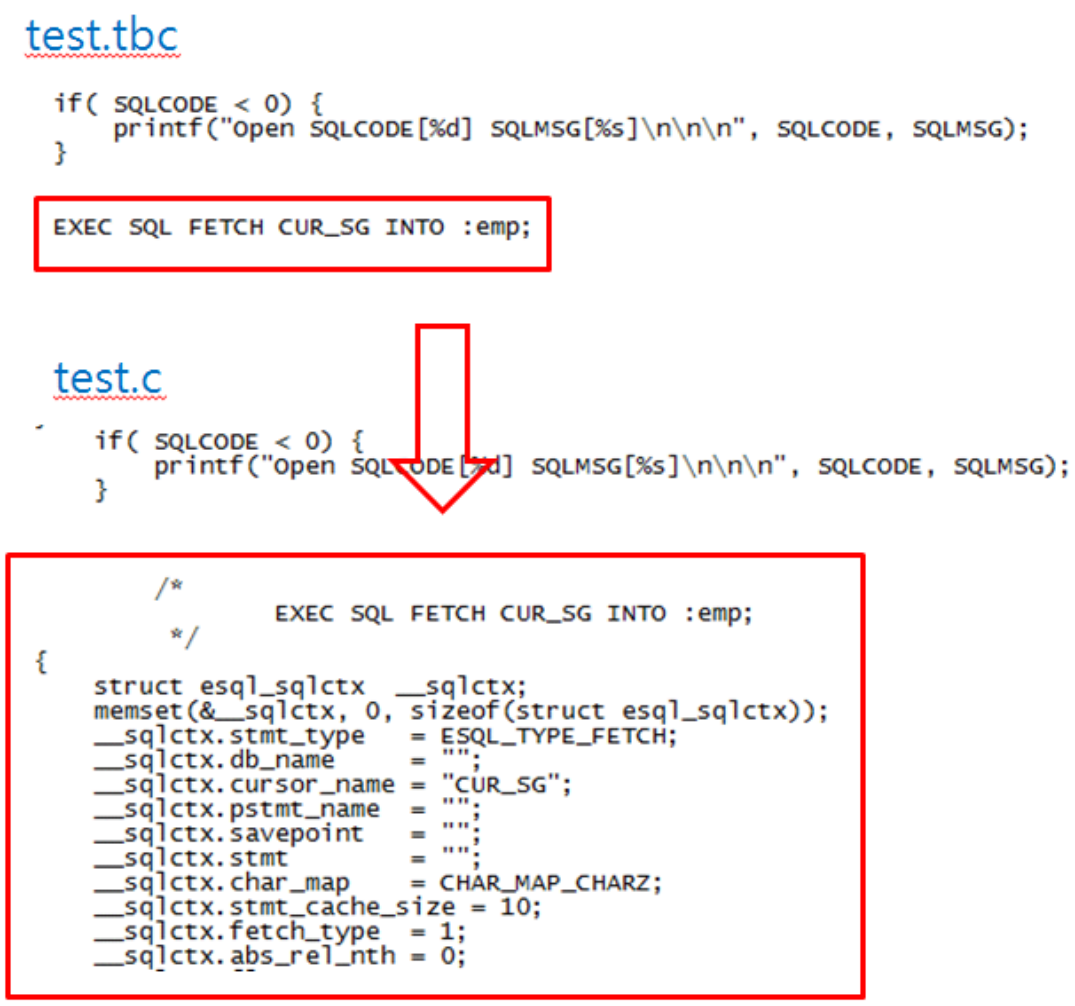
2. Parsing & Check

   해당 언어, ESQL, DML, 파라미터 등의 Parsing 및 Check를 수행한다.

3. 소스 생성

   원시 소스 파일(tbc or tbco)을 읽어서 ESQL 문법에 해당하는 부분을 주석 처리한 후 해당 부분의 소스를 생성한다.

[그림 4.2] 프리컴파일 소스생성



프리컴파일 옵션

해당 부분의 내용은 Tibero 정식 매뉴얼("Tibero tbESQL 안내서")을 참고한다.

사용 예제

ESQL/C로 예를 들면 아래와 같다.

```
tbpc –h
tbpc test1
tbpc iname=test1.tbc
tbpc test1 lines=yes
```

4.2.3. 컴파일 및 링크

프리컴파일 과정을 거친 후 해당 프로그래밍 언어의 컴파일러를 이용하여 컴파일 및 링크를 수행한다.

각 OS별로 ESQL/C의 예를 들면 아래와 같다. (프리컴파일 완료된 파일이 test.c일 경우)

- Linux

```
cc -m64 -O -I$TB_HOME/client/include -L$TB_HOME/client/lib -c test.c
cc -m64 -O -I$TB_HOME/client/include -o test.bin -L$TB_HOME/client/lib -ltbxa
-ltbertl -ltbcli -lclialloc test.o
```

- AIX

```
gcc -maix64 -Wl,-brtl -o $1 $1.c -g -I$TB_HOME/client/include -L$TB_HOME/client/
lib -ltbertl -ltbcli -lpthread -lm
cc -q64 -brtl -o $1 $1.c -g -I$TB_HOME/client/include -L$TB_HOME/client/lib
-ltbertl -ltbcli -lpthread -lm
```

- HP(IA64)

```
gcc -mlp64 -o $1 $1.c -g -I$TB_HOME/client/include -L$TB_HOME/client/lib -ltbertl
-ltbcli -lpthread -lm
cc +DD64 -o $1 $1.c -g -I$TB_HOME/client/include -L$TB_HOME/client/lib -ltbertl
-ltbcli -lpthread -lm
```

- HP(PARISC)

```
gcc -o $1 $1.c -g -I$TB_HOME/client/include -L$TB_HOME/client/lib -ltbertl -ltbcli
-lpthread -lm
cc +DD64 -o $1 $1.c -g -I$TB_HOME/client/include -L$TB_HOME/client/lib -ltbertl
-ltbcli -lpthread -lm
```

- SunOS 5.9

```
gcc -m64 -o $1 $1.c -g -I$TB_HOME/client/include -L$TB_HOME/client/lib -ltbertl
-ltbcli -lpthread -lm
cc -o $1 $1.c -g -xarch=v9 -xcode=pic32 -I$TB_HOME/client/include -L$TB_HOME/
client/lib -ltbertl -ltbcli -lpthread -lm
```

4.3. 문제 해결

특정 문제가 발생했을 때 해결하는 방법을 설명한다.

4.3.1. 로그 발생

tbESQL은 내부적으로 tbCLI(=ODBC) 인터페이스를 사용한다. 따라서 tbCLI(=ODBC) 관련 환경변수를 적용하면 로그를 발생시킬 수 있다.

4.3.2. 단계별 대처 방안

다음은 문제 발생에 대한 단계별 대처 방법이다.

- 프리컴파일할 때 문제 발생

  소스 라인의 문제인지 프리컴파일러의 옵션이 미적용되어 발생한 문제인지 확인한다.

- 컴파일 및 링크할 때 문제 발생

  일반 소스상의 문제라면 해당 프로그램 언어의 문법이 맞는지 확인하고 프리컴파일을 통해 Tibero에서 생성한 소스 부분이 문제라면 기술지원이 필요하다.

- 실행할 때 문제 발생

  tbCLI(=ODBC) 환경변수를 적용하여 로그를 출력한다.

4.3.3. 에러 코드

다음은 자주 사용되는 에러 코드에 대한 설명이다.

에러코드	설명
1403	NO_DATA_FOUND
-1405	NULL INDICATOR
-1	DUP_VAL_ON_INDEX
-5070	ZERO_DIVIDE

에러코드	설명
-5074	INVALID_NUMBER
-11025	Data is too long for the column
-17001	Login failed: invalid user name or password
-9071	Not logged into the server
-8026	Specified column name was not found
-8033	object not found
-2011	Binding Count Error
-2025	fetch out of sequence
-12003	Session Full

4.4. 예제

본 절에서는 makefile을 이용한 예제를 설명한다.

4.4.1. makefile 예제

다음 예제는 Linux에서 사용하는 예제로 각 사이트 상황에 맞추어 수정해서 사용한다. 참고로 아래 내용을 복사할 때 탭 부분의 복사가 잘되지 않으면 직접 탭을 적용해야 한다.

```
# Server ESQL*C makefile

TBLIBDIR = $(TB_HOME)/client/include

LFLAGS = -ltbertl -ltbcli
CFLAGS = -I$(TB_HOME)/client/include -L$(TB_HOME)/client/lib
ADDFLAGS = -m64 -O

# command
CC = /usr/bin/cc
TBPC   = $(TB_HOME)/client/bin/tbpc

########### modify start###############

# ESQL Include Path
INCLUDE1 = /usr/include
INCLUDE2 = $(TB_HOME)/client/include
INCLUDE3 = /usr/lib/gcc/x86_64-redhat-linux/4.1.1/include

# File Name
SRC_NAME=test
```

```
########### modify end##############

TARGET  = $(SRC_NAME).bin
OBJECT = $(SRC_NAME).o
TBPCFILE = $(SRC_NAME).tbc


all : $(TARGET)

# Build
$(TARGET): $(OBJECT)
 $(CC) $(ADDFLAGS) $(CFLAGS) $(LFLAGS) $(OBJECT) -o $(TARGET)

# C Compile
$(OBJECT): $(SRC_NAME).c
 $(CC) $(ADDFLAGS) $(CFLAGS) -c $(SRC_NAME).c

# Tibero ESQL PreCompile
$(SRC_NAME).c : $(TBPCFILE)
 $(TBPC) iname=$(TBPCFILE) \
                oname=$(SRC_NAME).c \
                include=$(INCLUDE1) \
                include=$(INCLUDE2) \
                include=$(INCLUDE3)

# cleaning object
clean :
 rm -f $(SRC_NAME).o $(SRC_NAME).c $(SRC_NAME).bin
```