Tibero

유틸리티 안내서

Tibero 7

대한민국 경기도 성남시 분당구 황새울로258번길 29, BS 타워 9층 우)13595

Website

http://www.tmaxtibero.com

기술서비스센터

Tel : +82-1544-8629

E-Mail : info@tmax.co.kr

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

Detailed Information related to the license can be found in the following directory : ${INSTALL_PATH}/license/oss_licenses

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

관련 상세한 정보는 제품의 다음의 디렉터리에 기재된 사항을 참고해 주십시오. : ${INSTALL_PATH}/license/oss_licenses

안내서 정보

안내서 제목: Tibero 유틸리티 안내서

발행일: 2024-08-22

소프트웨어 버전: Tibero 7.2.2

안내서 버전: v7.2.2

내용 목차

그림 목차

예 목차

안내서에 대하여

안내서의 대상

본 안내서는 Tibero®(이하 Tibero)에서 제공하는 각종 유틸리티를 사용하려는 모든 데이터베이스 사용자를 대상으로 기술한다.

안내서의 전제 조건

본 안내서를 원활히 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- 데이터베이스의 이해
- RDBMS의 이해
- SQL의 이해
- Eclipse 툴의 이해

  Eclipse 툴의 사용 방법은 웹사이트(http://www.eclipse.org/documentation)나 관련 문서를 참고한다.

안내서의 제한 조건

본 안내서는 Tibero를 실무에 적용하거나 운용하는 데 필요한 모든 사항을 포함하고 있지 않다. 따라서 설치, 환경설정 등 운용 및 관리에 대해서는 각 제품 안내서를 참고하기 바란다.

참고

Tibero의 설치 및 환경설정에 관한 내용은 "Tibero 설치 안내서"를 참고한다.

안내서 구성

Tibero 유틸리티 안내서는 총 6개의 장으로 구성되어 있다.

각 장의 주요 내용은 다음과 같다.

- 제1장: tbSQL

  대화형 SQL 명령어 처리 유틸리티인 tbSQL을 소개하고 사용 방법을 기술한다.

- 제2장: tbExport

  Tibero에 저장된 데이터베이스 객체의 전체 또는 일부를 추출하여 저장하는 유틸리티인 tbExport를 소개하고 사용 방법을 기술한다.

- 제3장: tbImport

  tbExport에 의하여 생성된 Export 파일로부터 데이터베이스 객체를 Tibero 데이터베이스에 저장하는 유틸리티인 tbImport를 소개하고 사용 방법을 기술한다.

- 제4장: tbLoader

  대량의 데이터를 한 번에 Tibero 데이터베이스에 적재하기 위한 유틸리티인 tbLoader를 소개하고 사용 방법을 기술한다.

- 제5장: tbdv

  tbdv를 소개하고 사용법을 기술한다.

- 제6장: 유틸리티 API

  Tibero의 유틸리티를 애플리케이션 프로그램에서 호출할 때 필요한 함수를 기술한다.

안내서 규약

표기	의미
<<AaBbCc123>>	프로그램 소스 코드의 파일명
<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	메일 계정, 웹 사이트
>	메뉴의 진행 순서
+----	하위 디렉터리 또는 파일 있음
|----	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
주의	주의할 사항
[그림 1.1]	그림 이름
[예 1.1]	예제 이름
`AaBbCc123`	Java 코드, XML 문서
[*command argument*]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨
|	선택 사항. 예) A|B: A나 B 중 하나
…	파라미터 등이 반복되어서 나옴
${ }	환경변수

시스템 사용 환경

	요구 사항
Platform	HP-UX 11i v3(11.31)
	Solaris (Solaris 11)
	AIX (AIX 7.1/AIX 7.2/AIX 7.3)
	GNU (X86, 64, IA64)
	Red Hat Enterprise Linux 7 kernel 3.10.0 이상
	Windows(x86) 64bit
Hardware	최소 2.5GB 하드디스크 공간
	1GB 이상 메모리 공간
Compiler	PSM (C99 지원 필요)
	tbESQL/C (C99 지원 필요)

관련 안내서

안내서	설명
Tibero 설치 안내서	설치 과정에 필요한 시스템 요구사항과 설치 및 제거 방법을 기술한 안내서이다.
Tibero tbCLI 안내서	Call Level Interface인 tbCLI의 개념과 구성요소, 프로그램 구조를 소개하고 tbCLI 프로그램을 작성하는 데 필요한 데이터 타입, 함수, 에러 메시지를 기술한 안내서이다.
Tibero 애플리케이션 개발자 안내서	각종 애플리케이션 라이브러리를 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero External Procedure 안내서	External Procedure를 소개하고 이를 생성하고 사용하는 방법을 기술한 안내서이다.
Tibero JDBC 개발자 안내서	Tibero에서 제공하는 JDBC 기능을 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero tbESQL/C 안내서	C 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbESQL/COBOL 안내서	COBOL 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbPSM 안내서	저장 프러시저 모듈인 tbPSM의 개념과 문법, 구성요소를 소개하고, 프로그램을 작성하는 데 필요한 제어 구조, 복합 타입, 서브프로그램, 패키지와 SQL 문장을 실행하고 에러를 처리하는 방법을 기술한 안내서이다.
Tibero tbPSM 참조 안내서	저장 프러시저 모듈인 tbPSM의 패키지를 소개하고, 이러한 패키지에 포함된 각 프러시저와 함수의 프로토타입, 파라미터, 예제 등을 기술한 참조 안내서이다.
Tibero 관리자 안내서	Tibero의 동작과 주요 기능의 원활한 수행을 보장하기 위해 DBA가 알아야 할 관리 방법을 논리적 또는 물리적 측면에서 설명하고, 관리를 지원하는 각종 도구를 기술한 안내서이다.
Tibero 에러 참조 안내서	Tibero를 사용하는 도중에 발생할 수 있는 각종 에러의 원인과 해결 방법을 기술한 안내서이다.
Tibero 참조 안내서	Tibero의 동작과 사용에 필요한 초기화 파라미터와 데이터 사전, 정적 뷰, 동적 뷰를 기술한 참조 안내서이다.

안내서	설명
Tibero SQL 참조 안내서	데이터베이스 작업을 수행하거나 애플리케이션 프로그램을 작성할 때 필요한 SQL 문장을 기술한 참조 안내서이다.
Tibero Spatial 참조 안내서	Tibero에서 Geometry 타입에 대한 설명과 Spatial 기능 관련 프러시저 함수 목록 및 사용 방법 등을 기술한 안내서이다.
Tibero TEXT 참조 안내서	Tibero의 제공하는 Text Index를 소개하고, Text Index를 생성 하고 사용하는 방법을 기술하는 안내서이다.
Tibero TDP.NET 안내서	Tibero Data Provider for .NET 기능을 기술하는 안내서이다.
Tibero IMCS 안내서	Tibero에서 제공하는 In-Memory Column Store(이하 IMCS) 기능을 기술하는 안내서이다.

제1장 tbSQL

본 장에서는 tbSQL 유틸리티를 소개하고 사용 방법을 설명한다.

1.1. 개요

tbSQL은 Tibero®(이하 Tibero)에서 제공하는 SQL 문장을 처리하는 대화형 유틸리티이다. 이 유틸리티로 SQL 질의, 데이터 정의어(DDL: Data Definition Language), 트랜잭션과 관련된 SQL 문장을 실행할 수 있다. 또한, PSM 프로그램을 생성하고 실행할 수 있으며 DBA는 Tibero의 시스템 관리를 위한 명령을 실행할 수 있다.

tbSQL은 이러한 기본 기능 외에도 자동 커밋을 설정하거나 운영체제 관련 명령어의 실행, 출력 저장, 스크립트 기능 등을 제공한다. 특히 스크립트 기능은 여러 SQL 문장 및 PSM 프로그램, tbSQL 유틸리티의 명령어를 하나의 스크립트 파일로 생성할 수 있어 편리하다.

tbSQL 유틸리티는 Tibero의 유틸리티 중에서 가장 빈번히 사용되는 것 중의 하나이며, SQL 문장의 실행 이외에 다음과 같은 기능을 제공한다.

- 일반적인 SQL 문장 및 PSM 프로그램의 입력, 편집, 저장, 실행
- 트랜잭션의 설정 및 종료
- 스크립트를 통한 일괄 작업의 실행
- DBA에 의한 데이터베이스 관리
- 데이터베이스의 기동 및 종료
- 외부 유틸리티 및 프로그램의 실행
- tbSQL 유틸리티의 환경설정

1.2. 빠른 시작

tbSQL 유틸리티는 Tibero를 설치하는 과정에서 함께 설치되며, Tibero를 제거하면 함께 제거된다.

1.2.1. 실행

tbSQL 유틸리티를 실행하는 방법은 다음과 같다.

[예 1.1] tbSQL 유틸리티의 실행

```
$ tbsql

tbSQL 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.
```

tbSQL 유틸리티가 정상적으로 실행되면 위 예제처럼 **SQL 프롬프트**가 나타난다. 이 프롬프트에서 데이터베이스 사용자는 SQL 문장을 실행할 수 있다.

tbSQL 유틸리티를 실행하는 명령어의 문법은 다음과 같다.

- 사용법

```
tbsql [[options]|[connect_string]|[start_script]]
```

  - [options]

항목	설명
-h, --help	도움말 화면을 출력한다.
-v, --version	버전을 출력한다.
-s, --silent	화면에 시작 메시지와 프롬프트를 출력하지 않는다.
-i, --ignore	로그인 스크립트(tbsql.login)를 실행하지 않는다.

  - [*connect_string*]

    Tibero에 접속하려는 사용자의 계정에 대한 정보를 포함하며, 다음과 같은 형식으로 지정한다.

```
username[/password[@connect_identifier]]
```

    다음은 **connect_string**에 사용할 수 있는 항목이다.

항목	설명
username	사용자명으로 대소문자를 구분하지 않는다. 만약 대소문자를 구분해야 한다면 이스케이프 문자(\)를 포함한 큰따옴표(" ")로 감싸야 한다.
password	사용자가 설정한 패스워드의 유형에 따라 대소문자를 구분해야 하는 경우도 있다.
connect_identifier	데이터베이스에 대한 접속 정보를 가진 DSN(Data Source Name)이거나 정해진 규칙의 연결 명세서이다.

  - [*start_script*]

    tbSQL 유틸리티의 시작과 함께 실행할 스크립트 파일을 설정할 수 있으며, 다음과 같은 형식으로 지정한다.

```
@filename[.ext] [parameter ...]
```

다음은 **start_script**에 사용할 수 있는 항목이다.

항목	설명
filename	파일명이다.
ext	파일의 확장자로, 지정하지 않을 경우 SUFFIX 시스템 변수에 지정된 확장자가 기본값이다.
parameter	파일에 사용된 치환변수 값이다.

1.2.2. 데이터베이스 접속

tbSQL 유틸리티를 실행한 후 SQL 프롬프트가 나타나면 데이터베이스에 접속할 수 있는 상태가 된다.

tbSQL 유틸리티는 데이터베이스의 세션을 시작하기 전에 수행해야 할 작업이 있을 경우 tbsql.login 파일을 작성하면 된다. 이 파일이 현재 디렉터리에 있는 경우에는 바로 실행하고, 그렇지 않은 경우에는 환경변수 **TB_SQLPATH**에 설정된 디렉터리에서 찾는다.

tbSQL 유틸리티를 이용하여 데이터베이스에 접속하는 방법은 다음과 같다.

[예 1.2] tbSQL 유틸리티를 이용한 데이터베이스 접속

```
$ tbsql SYS/syspassword

tbSQL 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.

Connected.
```

위 예에서는 UNIX 셸 프롬프트에서 tbSQL 유틸리티의 실행과 함께 사용자명과 패스워드를 입력한다.

사용자명과 **패스워드**를 입력할 때에는 다음과 같은 규칙이 있다.

항목	설명
사용자명	스키마 객체의 이름과 마찬가지로 대소문자를 구분하지 않는다. 단, 큰따옴표(" ")에 사용자명을 입력하는 경우는 예외다.
패스워드	사용자가 설정한 패스워드의 유형에 따라 대소문자를 구분해야 하는 경우도 있다.

[예 1.2]와 같이 connect_identifier를 생략하는 경우엔 디폴트 데이터베이스에 접속이 된다. 만약 특정 데이터베이스에 접속하고자 하는 경우엔 connect_identifier를 명시하면 되고, 다음과 같은 두가지 형식으로 사용할 수 있다.

- DSN(Data Source Name)

DSN은 tbdsn.tbr 파일이나 Windows에서의 데이터 원본(ODBC)에 정의된 이름을 명시하는 방법이다.

다음은 tbdsn.tbr 파일의 설정 예이다.

```
tibero7=(
    (INSTANCE=(HOST=192.168.36.42)
              (PORT=8629)
              (DB_NAME=tibero7)
    )
)
```

위와 같이 정의된 경우 접속하는 방법은 다음과 같다.

```
$ tbsql tibero/tmax@tibero7
```

- 연결 명세서

  연결 명세서는 tbdsn.tbr 파일을 사용하지 않고, 직접 접속 정보를 명시하는 방법으로 다음 두 가지 형식으로 사용할 수 있다.

  – 방법1

  ```
  (INSTANCE=(HOST=host)(PORT=port)(DB_NAME=dbname))
  ```

  사용 예는 다음과 같다.

  ```
  $ tbsql 'tibero/tmax@(INSTANCE=(HOST=192.168.36.42)(PORT=8629)(DB_NAME=tibero7))'
  ```

  – 방법2

  ```
  host:port/dbname
  ```

  사용 예는 다음과 같다.

  ```
  $ tbsql 'tibero/tmax@192.168.36.42:8629/tibero7'
  ```

1.2.3. 인터페이스

다음은 tbSQL 유틸리티를 실행했을 때의 화면이다.

```
$ tbsql

tbSQL 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.

SQL> CONNECT dbuser
Enter password : dbuserpassword
Connected to Tibero.
```

위의 예에서는 tbSQL 유틸리티를 실행한 뒤 CONNECT 명령어를 통해 dbuser라는 사용자명으로 데이터베이스에 접속했다. tbSQL 유틸리티는 이처럼 텍스트 모드의 화면에서 입력을 받고, 사용자의 요구에 따라 결과를 출력한다.

참고

본 안내서에서는 특별한 경우를 제외하고는 모든 SQL 문장과 PSM 프로그램, tbSQL 유틸리티의 명령어를 대문자로 표현한다. 명령어의 파라미터로 소문자가 사용된 경우는 다른 파라미터로 확장될 수 있는 경우이다.

---

tbSQL 유틸리티는 다음과 같은 특성을 가진 인터페이스로 실행한다.

- tbSQL 유틸리티가 정상적으로 실행되면 SQL 프롬프트가 출력된다.

  프롬프트에서 SQL 문장, PSM 프로그램, tbSQL 유틸리티의 명령어를 입력할 수 있다.

- 여러 라인에 걸쳐 입력할 수 있다.

  SQL 문장과 PSM 프로그램은 입력과 실행을 분리할 수 있다. 하지만 tbSQL 유틸리티의 명령어는 입력과 동시에 실행된다.

- 대소문자를 구분하지 않는다.

  SQL 문장 내의 문자열 데이터처럼 특별한 경우를 제외하고는 대소문자를 구분하지 않는다.

  예를 들어 다음의 두 문장은 서로 같은 의미이다.

```
SQL> SET AUTOCOMMIT ON
SQL> set autocommit on
```

1.2.4. 환경설정

tbSQL 유틸리티의 사용 환경을 설정하려면 **SET** 명령어를 사용해야 한다. SET 명령어를 통해 SQL 질의를 수행한 결과의 출력 형태, 트랜잭션의 커밋 여부 등을 설정할 수 있다.

다음은 SET 명령어의 문법이다.

```
SET [system_variable] [system_variable_value]
```

참고

자세한 내용은 “1.3. 시스템 변수”를 참고한다.

1.2.5. 종료

tbSQL 유틸리티를 종료하려면 SQL 프롬프트에서 **EXIT** 또는 **QUIT** 명령어를 입력해야 한다.

```
SQL> EXIT
```

참고

tbSQL 유틸리티에서 제공하는 명령어에 대한 자세한 내용은 “1.6. 명령어”를 참고한다.

1.3. 시스템 변수

본 절에서는 tbSQL 유틸리티의 시스템 변수에 대하여 설명한다. tbSQL 유틸리티의 시스템 변수에 설정할 값은 SET 명령어로 설정하고, SHOW 명령어로 출력한다.

다음은 SET 명령어에서 설정할 수 있는 시스템 변수를 요약한 표이다.

시스템 변수	기본값	설명
AUTOCOMMIT	OFF	자동 커밋 여부를 설정하는 시스템 변수이다.
AUTOTRACE	OFF	수행 중인 질의의 플랜이나 통계 정보의 출력 여부를 설정하는 시스템 변수이다.
BLOCKTERMINATOR	"." (0x2E)	PSM 문장에서 입력의 마지막을 나타내는 문자를 설정하는 시스템 변수이다.
COLSEP	" " (0x20)	SQL 문장 중 조회 쿼리에 대한 수행 결과를 보여줄 때 컬럼 사이를 구분하는 문자를 지정하는 시스템 변수이다.
CONCAT	"." (0x2E)	치환 변수 이름의 끝을 나타내는 문자를 설정하는 시스템 변수이다.
DDLSTATS	OFF	DDL 문장의 플랜이나 통계 정보의 표시 여부를 설정하는 시스템 변수이다.
DEFINE	"&" (0x26)	치환 변수를 정의할 때 사용할 문자를 지정하는 시스템 변수이다.
DESCRIBE	DEPTH 10	DESCRIBE 명령어에서 보여줄 객체 명세의 단계를 지정하는 시스템 변수이다.
ECHO	OFF	@ 또는 START 명령으로 스크립트 파일을 실행시킬 때 스크립트 내에서 실행되는 쿼리의 화면 출력 여부를 결정하는 시스템 변수이다.
EDITFILE	".tbedit.sql"	EDIT 명령어에서 사용하는 파일 이름의 기본값을 설정하는 시스템 변수이다.
ESCAPE	OFF	이스케이프 문자를 설정하는 시스템 변수이다.
EXITCOMMIT	ON	유틸리티를 종료할 때 커밋 여부를 설정하는 시스템 변수이다.

시스템 변수	기본값	설명
FEEDBACK	0	SQL 문장의 수행 결과를 화면에 출력할지를 설정하는 시스템 변수이다.
HEADING	ON	쿼리 실행 결과를 출력할 때 컬럼의 머릿글 표시 여부를 설정하는 시스템 변수이다.
HEADSEP	"|" (0x7C)	머릿글의 줄바꿈 문자를 설정하는 시스템 변수이다.
HISTORY	50	명령어 히스토리의 크기를 설정하는 시스템 변수이다.
INTERVAL	1	LOOP 명령어에서 각 문장을 수행한 후 대기하는 시간을 설정하는 시스템 변수이다.
LINESIZE	80	한 라인에 출력할 문자 수를 설정하는 시스템 변수이다.
LONG	80	VARCHAR보다 큰 문자형 타입의 데이터를 표시하기 위해 사용할 문자 수를 설정하는 시스템 변수이다.
MARKUP	OFF	유틸리티의 출력 내용을 HTML로 출력할 것인지를 설정하는 시스템 변수이다.
NEWPAGE	1	각 페이지 시작 부분에 추가할 빈 줄 수를 설정하는 시스템 변수이다.
NUMFORMAT	""	숫자형 데이터의 기본 컬럼 포맷을 설정하는 시스템 변수이다.
NUMWIDTH	10	숫자형 데이터의 기본 출력 길이를 설정하는 시스템 변수이다.
PAGESIZE	24	한 화면에 출력할 라인 수를 설정하는 시스템 변수이다.
PAUSE	OFF	한 페이지를 출력한 후 다음 페이지를 출력하기 전에 사용자 입력을 기다릴지를 지정하는 시스템 변수이다.
RECSEP	WRAPPED	로우 구분자를 출력할 단위를 지정하는 시스템 변수이다.
RECSEPCHAR	" " (0x20)	로우 구분자로 사용할 문자를 설정하는 시스템 변수이다.
ROWS	ON	질의문의 결과를 화면에 출력할 것인지를 설정하는 시스템 변수이다.
SERVEROUTPUT	OFF	DBMS_OUTPUT 패키지의 결과를 출력할 것인지를 설정하는 시스템 변수이다.
SQLCODE	0	마지막으로 실행된 SQLCODE 값을 보여주는 시스템 변수이다. 이 값은 SET으로 설정할 수 없다.
SQLPROMPT	"SQL> "	화면상의 프롬프트 문자를 설정하는 시스템 변수이다.
SQLTERMINATOR	";" (0x3B)	SQL 문장을 종료하는 문자를 설정하는 시스템 변수이다.
SUFFIX	"sql"	파일 확장자의 기본값을 설정하는 시스템 변수이다.
TERMOUT	ON	스크립트에서 수행된 명령어의 결과를 화면에 출력할 것인지를 설정하는 시스템 변수이다.
TIME	OFF	현재 시간을 화면에 출력할 것인지를 설정하는 시스템 변수이다.

시스템 변수	기본값	설명
TIMEOUT	3	PING 명령어에서 서버가 응답할 때까지 기다릴 시간을 설정하는 시스템 변수이다.
TIMING	OFF	SQL, PSM 문장의 결과를 출력할 때마다 수행 시간을 출력할 것인지를 설정하는 시스템 변수이다.
TRIMOUT	ON	화면에 출력되는 라인 뒤에 오는 공백의 제거 여부를 설정하는 시스템 변수이다.
TRIMSPOOL	OFF	스풀링 중인 라인 뒤에 오는 공백의 제거 여부를 설정하는 시스템 변수이다.
UNDERLINE	"-" (0x2D)	머릿글의 밑줄로 사용할 문자를 설정하는 시스템 변수이다.
VERIFY	ON	명령을 실행할 때 치환변수가 적용된 내역에 대한 출력 여부를 설정하는 시스템 변수이다.
WRAP	ON	출력할 라인이 긴 경우 나머지를 다음 라인에 출력할 것인지를 설정하는 시스템 변수이다.

다음은 시스템 변수를 설정하는 예이다.

```
SET AUTOCOMMIT ON
SET PAGESIZE 32
SET TRIMSPOOL ON
```

1.3.1. AUTOCOMMIT

INSERT, UPDATE, DELETE, MERGE 혹은 PSM 블록 등의 SQL 문장을 실행한 후 자동으로 커밋을 수행하도록 설정한다.

AUTOCOMMIT의 세부 내용은 다음과 같다.

- 문법

```
SET AUTO[COMMIT] {ON|OFF|n}
```

항목	설명
ON	자동 커밋을 수행한다.
OFF	자동 커밋을 수행하지 않는다. OFF로 설정한 경우에는 명시적으로 커밋을 수행해야 한다. (기본값)
n	n개의 INSERT, UPDATE, DELETE, MERGE 또는 PSM 블록을 성공적으로 수행한 후에 커밋을 수행한다. n이 0이면 OFF와 동일하고, 1이면 ON과 동일하다.

1.3.2. AUTOTRACE

수행 중인 질의의 플랜이나 통계 정보를 보여준다. DBA 권한 또는 PLUSTRACE 권한이 있어야 사용할 수 있다. PLUSTRACE는 AUTOTRACE에 필요한 특권들이 포함된 권한으로 DBA 권한을 가진 사용자가 생성하여 다른 사용자에게 부여할 수 있다. 생성 스크립트는 $TB_HOME/scripts/plustrace.sql이다.

AUTOTRACE의 세부 내용은 다음과 같다.

- 문법

```
SET AUTOT[RACE] {ON|OFF|TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]] [PLANS[TAT]]
```

  – 입력 항목

항목	설명
ON	질의의 결과 및 추가 옵션에 따라 플랜 정보나 통계 정보를 출력한다.
OFF	플랜 및 통계 정보를 출력하지 않는다. (기본값)
TRACE[ONLY]	질의의 결과를 보여주지 않으며, 추가 옵션에 따라 플랜 정보나 통계 정보를 출력한다.

  – 옵션

  다음은 플랜이나 통계 정보를 보여줄지를 지정하는 옵션이다.

항목	설명
	아무것도 명시하지 않으면 플랜과 통계 정보를 모두 보여준다.
EXP[LAIN]	플랜 정보를 보여준다.
STAT[ISTICS]	통계 정보를 보여준다.
PLANS[TAT]	질의에 대한 노드 별 수행 정보(수행 시간, 처리 로우 수, 수행 횟수 등)을 보여준다.

1.3.3. BLOCKTERMINATOR

PSM 문장에서 입력의 마지막을 나타내는 문자를 설정한다.

BLOCKTERMINATOR의 세부 내용은 다음과 같다.

- 문법

```
SET BLO[CKTERMINATOR] {c|ON|OFF}
```

항목	설명
c	PSM 프로그램 입력을 마치는 것을 나타내는 문자이다. (기본값: ".")

항목	설명
ON	BLOCKTERMINATOR를 활성화한다. (기본값)
OFF	BLOCKTERMINATOR를 비활성화한다.

1.3.4. COLSEP

SELECT 문장을 실행한 후 출력되는 여러 컬럼 사이의 구분을 나타내는 문자를 설정한다.

COLSEP의 세부 내용은 다음과 같다.

- 문법

```
SET COLSEP {text}
```

항목	설명
text	컬럼 사이를 구분짓는 문자이다. (기본값: " ")

1.3.5. CONCAT

치환 변수 이름의 끝을 나타내는 문자를 설정한다.

CONCAT의 세부 내용은 다음과 같다.

- 문법

```
SET CON[CAT] {c|ON|OFF}
```

항목	설명
c	치환 변수 이름에 대한 종료 문자이다. (기본값: ".")
ON	CONCAT을 활성화한다. (기본값)
OFF	CONCAT을 비활성화한다.

1.3.6. DDLSTATS

수행 중인 DDL 문장의 플랜이나 통계 정보를 보여준다. 단, 주의할 점은 AUTOTRACE 기능도 반드시 활성화시켜야 한다.

DDLSTATS의 세부 내용은 다음과 같다.

- 문법

```
SET DDLSTAT[S] {ON|OFF}
```

항목	설명
ON	DDLSTATS를 활성화한다.
OFF	DDLSTATS를 비활성화한다. (기본값)

1.3.7. DEFINE

치환 변수를 정의할 때 사용할 문자를 설정한다.

DEFINE의 세부 내용은 다음과 같다.

- 문법

```
SET DEF[INE] {c|ON|OFF}
```

항목	설명
c	치환 변수를 나타내는 문자이다. (기본값: "&")
ON	DEFINE을 활성화한다. (기본값)
OFF	DEFINE을 비활성화한다.

1.3.8. DESCRIBE

DESCRIBE 명령어를 통한 객체 명세를 어느 단계까지 보여줄지를 설정한다.

DESCRIBE의 세부 내용은 다음과 같다.

- 문법

```
SET DESCRIBE DEPTH {n}
```

항목	설명
n	재귀적으로 출력할 단계이다. (기본값: 10)

1.3.9. ECHO

@ 또는 START 명령으로 스크립트 파일을 실행시킬 때 스크립트 내에서 실행되는 쿼리의 화면 출력 여부를 설정한다.

ECHO의 세부 내용은 다음과 같다.

- 문법

```
SET ECHO {ON|OFF}
```

항목	설명
ON	ECHO를 활성화한다.
OFF	ECHO를 비활성화한다. (기본값)

1.3.10. EDITFILE

EDIT 명령어에서 사용할 파일 이름의 기본값을 설정한다. 확장자를 생략할 경우 SUFFIX에 설정된 값을 사용한다.

EDITFILE의 세부 내용은 다음과 같다.

- 문법

```
SET EDITF[ILE] filename[.ext]
```

항목	설명
filename[.ext]	EDIT 명령어에서 사용할 파일 이름이다. (기본값: .tbedit.sql)

1.3.11. ESCAPE

DEFINE에서 정의한 치환 변수 문자를 무시하도록 하는 이스케이프 문자를 설정한다. 이스케이프를 활성화한 후 설정한 이스케이프 문자를 '&<문자열>' 앞에 붙여 쓰면, 치환 변수로 인식되지 않는다.

ESCAPE의 세부 내용은 다음과 같다.

- 문법

```
SET ESC[APE] {c|ON|OFF}
```

항목	설명
c	이스케이프 문자이다. (기본값: "\")
ON	ESCAPE를 활성화한다.
OFF	ESCAPE를 비활성화한다. (기본값)

1.3.12. EXITCOMMIT

유틸리티를 종료할 때 커밋 여부를 설정한다.

EXITCOMMIT의 세부 내용은 다음과 같다.

- 문법

```
SET EXITC[OMMIT] {ON|OFF}
```

항목	설명
ON	EXITCOMMIT을 활성화한다. (기본값)
OFF	EXITCOMMIT을 비활성화한다.

1.3.13. FEEDBACK

SQL 문장의 수행 결과를 화면에 출력할지 설정한다.

FEEDBACK의 세부 내용은 다음과 같다.

- 문법

```
SET FEED[BACK] {n|ON|OFF}
```

항목	설명
n	수행 결과를 화면에 출력하기 위한 최소 로우 개수이다. (기본값: 0)
ON	FEEDBACK을 활성화한다. (기본값)
OFF	FEEDBACK을 비활성화한다.

1.3.14. HEADING

쿼리 실행 결과를 출력할 때 컬럼의 머릿글 표시 여부를 설정한다.

HEADING의 세부 내용은 다음과 같다.

- 문법

```
SET HEA[DING] {ON|OFF}
```

항목	설명
ON	HEADING을 활성화한다. (기본값)
OFF	HEADING을 비활성화한다.

1.3.15. HEADSEP

머릿글의 줄바꿈 문자를 설정한다.

HEADSEP의 세부 내용은 다음과 같다.

- 문법

```
SET HEADS[EP] {c|ON|OFF}
```

항목	설명
c	줄바꿈 문자이다. (기본값: "|")
ON	HEADSEP을 활성화한다. (기본값)
OFF	HEADSEP을 비활성화한다.

1.3.16. HISTORY

명령어 히스토리의 크기를 설정한다.

HISTORY의 세부 내용은 다음과 같다.

- 문법

```
SET HIS[TORY] {n}
```

항목	설명
n	명령어 히스토리의 크기이다. (기본값: 50)

1.3.17. INTERVAL

LOOP 명령어에서 각 수행이 끝난 후 대기하는 시간을 설정한다.

INTERVAL의 세부 내용은 다음과 같다.

- 문법

```
SET INTER[VAL] {n}
```

항목	설명
n	대기 시간으로 단위는 초다. (기본값: 1)

1.3.18. LINESIZE

화면상의 한 라인의 길이를 설정한다. 라인 길이의 최솟값은 1이며, 최댓값은 운영체제에 따라 다르다.

LINESIZE의 세부 내용은 다음과 같다.

- 문법

```
SET LIN[ESIZE] {n}
```

항목	설명
n	화면상의 한 라인의 길이이다. (기본값: 80)

1.3.19. LONG

CLOB이나 BLOB, NCLOB, LONG, XML 타입의 데이터를 읽어 와서 출력할 길이를 설정한다. 길이는 2,000,000,000을 넘을 수 없다.

LONG의 세부 내용은 다음과 같다.

- 문법

```
SET LONG {n}
```

항목	설명
n	대용량 데이터의 기본 출력 길이이다. (기본값: 80)

1.3.20. MARKUP

유틸리티의 출력 내용을 HTML로 작성한다.

MARKUP의 세부 내용은 다음과 같다.

- 문법

```
SET MARKUP HTML {ON|OFF} [HEAD text] [BODY text] [TABLE text] [ENTMAP {ON|OFF}]
[SPOOL {ON|OFF}] [PRE[FORMAT] {ON|OFF}]
```

  – 입력 항목

항목	설명
ON	MARKUP HTML을 활성화한다.
OFF	MARKUP HTML을 비활성화한다. (기본값)

  – 옵션

  다음은 HTML 출력 세부 내용을 지정하는 옵션이다.

항목	설명
HEAD text	<head> 태그에 포함할 내용을 지정한다. (기본값: 유틸리티 기본값)
BODY text	<body> 태그의 속성을 지정한다. (기본값: 없음)
TABLE text	<table> 태그의 속성을 지정한다. (기본값: 유틸리티 기본값)
ENTMAP ON	<, >, ", & 문자를 HTML 엔티티로 변환하는 기능을 활성화한다. (기본값)
ENTMAP OFF	<, >, ", & 문자를 HTML 엔티티로 변환하는 기능을 비활성화한다.
SPOOL ON	SPOOL 파일 처음과 마지막에 <html>과 <body> 태그를 작성해주는 기능을 활성화한다.
SPOOL OFF	SPOOL 파일 처음과 마지막에 <html>과 <body> 태그를 작성해주는 기능을 비활성화한다. (기본값)
PREFORMAT ON	쿼리 수행 결과를 <pre> 태그에 작성한다.
PREFORMAT OFF	쿼리 수행 결과를 HTML 테이블로 작성한다. (기본값)

참고

MARKUP 기능은 Tibero 7 FS02 릴리즈부터 지원한다.

1.3.21. NEWPAGE

각 페이지의 시작 부분에 추가할 빈 줄 개수를 설정한다.

NEWPAGE의 세부 내용은 다음과 같다.

- 문법

```
SET NEWP[AGE] {1|n|NONE}
```

항목	설명
n	빈 줄 개수이다. (기본값: 1)

1.3.22. NUMFORMAT

NUMBER 타입의 기본 컬럼 포맷을 설정한다. COLUMN 명령으로 FORMAT이 정의된 것을 제외한 숫자형 컬럼에 적용된다.

NUMFORMAT의 세부 내용은 다음과 같다.

- 문법

```
SET NUMF[ORMAT] {fmt_str}
```

항목	설명
fmt_str	NUMBER 타입 데이터의 기본 컬럼 포맷이다. 자세한 내용은 “1.7. 컬럼 포맷”의 숫자형에 대한 설명을 참조한다. (기본값: "")

1.3.23. NUMWIDTH

NUMBER 타입을 출력할 길이를 설정한다. LINESIZE를 넘을 수 없다.

NUMWIDTH의 세부 내용은 다음과 같다.

- 문법

```
SET NUM[WIDTH] {n}
```

항목	설명
n	NUMBER 타입 데이터의 기본 출력 길이이다. (기본값: 10)

1.3.24. PAGESIZE

tbSQL 유틸리티에서 출력하는 내용이 포함되는 각 페이지 내의 라인 개수를 설정한다.

PAGESIZE의 세부 내용은 다음과 같다.

- 문법

```
SET PAGES[IZE] {n}
```

항목	설명
n	한 페이지의 라인 개수이다. (기본값: 24)

1.3.25. PAUSE

한 페이지를 출력한 뒤 다음 페이지를 출력하기 전에 사용자 입력을 기다릴지를 설정한다.

PAUSE의 세부 내용은 다음과 같다.

- 문법

```
SET PAU[SE] {ON|OFF}
```

항목	설명
ON	PAUSE를 활성화한다.
OFF	PAUSE를 비활성화한다. (기본값)

1.3.26. RECSEP

로우 구분자를 출력하는 단위를 설정한다.

RECSEP의 세부 내용은 다음과 같다.

- 문법

```
SET RECSEP {WR[APPED]|EA[CH]|OFF}
```

항목	설명
WRAPPED	로우가 WRAPPING되는 경우 구분자를 출력한다. (기본값)
EACH	모든 로우 단위로 구분자를 출력한다.
OFF	RECSEP을 비활성화한다.

1.3.27. RECSEPCHAR

로우 구분자로 사용할 문자를 설정한다. 이 구분자는 LINESIZE 값만큼 반복해서 출력된다.

RECSEPCHAR의 세부 내용은 다음과 같다.

- 문법

```
SET RECSEPCHAR {c}
```

항목	설명
c	로우 구분자이다. (기본값: " ")

1.3.28. ROWS

질의문의 결과를 화면에 출력할지를 설정한다.

ROWS의 세부 내용은 다음과 같다.

- 문법

```
SET ROWS {ON|OFF}
```

항목	설명
ON	ROWS를 활성화한다. (기본값)
OFF	ROWS를 비활성화한다.

1.3.29. SERVEROUTPUT

DBMS_OUTPUT 패키지의 결과를 출력할 것인지 설정한다.

SERVEROUTPUT의 세부 내용은 다음과 같다.

- 문법

```
SET SERVEROUT[PUT] {ON|OFF} [SIZE n]
```

항목	설명
ON	SERVEROUTPUT을 활성화한다.
OFF	SERVEROUTPUT을 비활성화한다. (기본값)
n	SERVEROUTPUT 버퍼 크기를 지정한다. (기본값: 1000000)

1.3.30. SQLCODE

가장 최근에 실행된 SQLCODE 값을 보여주는 시스템 변수이다. 이 값은 SET으로 설정할 수 없다.

SQLCODE의 세부 내용은 다음과 같다.

- 문법

```
SHOW SQLCODE
```

1.3.31. SQLPROMPT

화면상의 프롬프트 문자를 설정한다.

SQLPROMPT의 세부 내용은 다음과 같다.

- 문법

```
SET SQLP[ROMPT] {prompt_string}
```

항목	설명
prompt_string	프롬프트로 사용할 문자열이다. (기본값: "SQL>") 이 문자열 안의 환경변수와 _user 식별자를 인식한다. 예를 들어 '$ISQL_PROMPT'라고 지정하면 환경변수 $ISQL_PROMPT의 값이 치환되어 프롬프트로 사용된다. 이때 환경변수의 이름은 대소문자를 구분한다. '_user'라고 지

항목	설명
	정하면 _user의 값이 현재 접속된 유저의 이름으로 동적으로 치환되어 프롬프트로 사용된다. 환경변수와 _user 식별자를 동시에 한 문자열 안에 포함시킬 수 있다. 문자열의 최대 길이는 128자로 제한된다.

1.3.32. SQLTERMINATOR

SQL 문장을 종료하는 문자를 설정한다.

SQLTMINATOR의 세부 내용은 다음과 같다.

- 문법

```
SET SQLT[MINATOR} {c|ON|OFF}
```

항목	설명
c	SQL 문장의 종료를 알리는 문자이다. (기본값: ";")
ON	SQLTERMINATOR를 활성화한다.
OFF	SQLTERMINATOR를 비활성화한다.

1.3.33. SUFFIX

파일 확장자를 생략했을 때 사용할 파일 확장자를 설정한다.

SUFFIX의 세부 내용은 다음과 같다.

- 문법

```
SET SUF[FIX] {extension}
```

항목	설명
extension	기본으로 사용할 파일 확장자이다. (기본값: sql)

1.3.34. TERMOUT

스크립트에서 수행된 명령어의 결과를 화면에 출력할 것인지 설정한다.

TERMOUT의 세부 내용은 다음과 같다.

- 문법

```
SET TERM[OUT] {ON|OFF}
```

항목	설명
ON	TERMOUT을 활성화한다. (기본값)
OFF	TERMOUT을 비활성화한다.

1.3.35. TIME

프롬프트에 현재 시간을 출력할 것인지 설정한다.

TIME의 세부 내용은 다음과 같다.

- 문법

```
SET TI[ME] {ON|OFF}
```

항목	설명
ON	TIME을 활성화한다.
OFF	TIME을 비활성화한다. (기본값)

1.3.36. TIMEOUT

PING 명령어에서 서버가 응답할 때까지 기다릴 시간을 설정한다. (단위: 초)

TIMEOUT의 세부 내용은 다음과 같다.

- 문법

```
SET TIMEOUT {n}
```

항목	설명
n	서버의 응답을 기다리는 시간이다. (기본값: 3)

1.3.37. TIMING

SQL, PSM 문장의 실행 결과를 출력할 때마다 수행 시간을 출력할 것인지 설정한다.

TIMING의 세부 내용은 다음과 같다.

- 문법

```
SET TIMI[NG] {ON|OFF}
```

항목	설명
ON	TIMING을 활성화한다.
OFF	TIMING을 비활성화한다. (기본값)

1.3.38. TRIMOUT

SQL, PSM 문장의 실행 결과를 출력할 때마다 모든 라인의 뒤에 오는 공백을 제거할 것인지 설정한다.

TRIMOUT의 세부 내용은 다음과 같다.

- 문법

```
SET TRIM[OUT] {ON|OFF}
```

항목	설명
ON	TRIMOUT을 활성화한다. (기본값)
OFF	TRIMOUT을 비활성화한다.

1.3.39. TRIMSPOOL

SQL, PSM 문장의 실행 결과를 스풀링할 때마다 모든 라인의 뒤에 오는 공백을 제거할 것인지 설정한다.

TRIMSPOOL의 세부 내용은 다음과 같다.

- 문법

```
SET TRIMS[POOL] {ON|OFF}
```

항목	설명
ON	TRIMSPOOL을 활성화한다.

항목	설명
OFF	TRIMSPOOL을 비활성화한다. (기본값)

1.3.40. UNDERLINE

머릿글의 밑줄로 사용할 문자를 설정한다.

UNDERLINE의 세부 내용은 다음과 같다.

- 문법

```
SET UND[ERLINE] {c|ON|OFF}
```

항목	설명
c	밑줄 문자이다. (기본값: "-")
ON	UNDERLINE을 활성화한다. (기본값)
OFF	UNDERLINE을 비활성화한다.

1.3.41. VERIFY

치환 변수가 포함된 명령을 실행할 때 치환 변수가 값으로 치환되는 결과의 출력 여부를 설정한다.

VERIFY의 세부 내용은 다음과 같다.

- 문법

```
SET VER[IFY] {ON|OFF}
```

항목	설명
ON	VERIFY를 활성화한다. (기본값)
OFF	VERIFY를 비활성화한다.

1.3.42. WRAP

화면에서 출력된 라인이 LINESIZE 변수로 설정된 값보다 긴 경우 나머지를 다음 라인에 출력할 것인지 아니면 LINESIZE 만큼만 출력할 것인지 설정한다.

WRAP의 세부 내용은 다음과 같다.

- 문법

```
SET WRA[P] {ON|OFF}
```

항목	설명
ON	WRAP을 활성화한다. (기본값)
OFF	WRAP을 비활성화한다.

1.4. 기본 기능

tbSQL 유틸리티에서 주로 사용하는 기능은 SQL 문장이나 PSM 프로그램을 직접 입력하여 실행하는 것이다. 본 절에서는 명령어의 입력과 실행에 관하여 설명한 후 기타 기능에 대해서 설명한다.

1.4.1. 명령어의 입력

tbSQL 유틸리티의 명령 프롬프트에서의 입력은 크게 SQL 문장, PSM 프로그램, 유틸리티 명령어의 세 가지로 구분할 수 있다. 명령어의 입력 방법은 대체로 서로 유사하다. 각 명령어를 입력하는 방법에 대하여 차례로 설명한다.

SQL 문장의 입력

다음은 SQL 문장을 입력하는 방법이다.

- 일반적인 입력

  일반적인 SQL 문장은 tbSQL 유틸리티의 프롬프트에서 입력한다. 하나의 SQL 문장을 여러 라인에 걸쳐 입력할 수 있으며, SQL 문장의 입력을 취소하려면 빈 라인을 입력한다.

- 줄 바꿈

  하나의 SQL 문장을 여러 라인에 걸쳐 입력할 경우 연속된 문자열이 아닌 어떤 곳에서도 줄 바꿈을 할 수 있다. 대개의 경우 읽기에 편하고 변경하기에 용이하도록 절 단위로 줄 바꿈을 하여 입력한다.

- 주석(comment)의 삽입

  SQL 문장을 입력하는 중간에 주석을 삽입할 수 있다. 주석은 두 개의 마이너스 부호(--)로 시작되며, 그 라인의 마지막까지 포함한다. 주석은 그 자체만으로 하나의 라인을 형성할 수 있으며, 한 라인에서 다른 문자열의 뒤에 올 수도 있다.

- 이전에 저장된 문장을 이용하여 입력

입력한 SQL 문장은 tbSQL 유틸리티의 SQL 버퍼에 저장된다. 따라서, 같거나 유사한 SQL 문장을 입력하기 위하여 이전에 저장된 문장을 이용할 수 있다. 이전 문장을 필요에 따라 변경하면 새로운 문장으로 입력되어 SQL 버퍼에 저장된다. SQL 버퍼에는 하나의 SQL 문장 또는 PSM 프로그램이 저장된다. 운영체제에 따라서 키보드의 위쪽 방향키(↑)나 아래쪽 방향키(↓)를 누르면 이전에 입력한 문장을 다시 불러올 수 있다. 키를 누를 때마다 이전에 저장된 문장이 한 라인씩 나타나므로 이전에 저장된 SQL 문장 전체뿐만 아니라 일부를 불러올 수 있다.

다음은 tbSQL 유틸리티에서 SQL 문장을 입력하는 예이다.

```
SQL> SELECT ENAME, SALARY, ADDR
    FROM EMP
    -- this is a comment.
    WHERE DEPTNO = 5;
SQL>
```

PSM 프로그램의 입력

PSM 프로그램은 다수의 SQL 문장 또는 PSM 문장으로 이루어지며, 각 SQL 문장은 세미콜론(;)으로 마친다. PSM 프로그램을 입력하기 시작하면 tbSQL 유틸리티는 자동으로 PSM 프로그램 입력 모드로 전환한다. PSM 프로그램 입력 모드에서는 SQL 문장의 입력이 완료되었을 때 SQL 문장이 개별적으로 실행되지 않는다.

tbSQL 유틸리티가 PSM 프로그램 입력 모드로 전환하도록 만드는 문장에는 DECLARE, BEGIN 등 이름 없는 블록(anonymous block)과 각각 프러시저, 함수, 트리거를 생성하는 CREATE (OR REPLACE) PROCEDURE, FUNCTION, TRIGGER가 있다.

PSM 프로그램에서의 입력 방법은 일반 SQL 문장의 경우와 유사하다.

다음은 PSM 프로그램을 입력하는 방법이다.

- 일반적인 입력

  PSM 프로그램을 여러 라인에 걸쳐서 입력할 수 있다. SQL 문장의 입력을 취소하려면 빈 라인을 입력하였으나, PSM 프로그램의 입력을 취소하려면 블록종료 문자(BLOCKTERMINATOR)를 입력한다. 블록종료 문자의 기본값은 점(.)이다. 블록종료 문자는 해당 문자만 한 라인에 입력해야 하며, 다른 문자열과 함께 입력하면 안 된다.

- 이전에 저장된 문장을 이용하여 입력

  한번 입력된 프로그램은 SQL 버퍼에 저장되어 다시 사용할 수 있다.

- 주석의 삽입

  SQL 문장과 같은 방식으로 주석을 삽입한다.

다음은 tbSQL 유틸리티에서 이름 없는 블록을 입력하는 예이다.

```
SQL> DECLARE
    deptno NUMBER(2);
    BEGIN
        deptno := 5;
        UPDATE EMP SET SALARY = SALARY * 1.05
        WHERE DEPTNO = deptno;
        -- this is a comment.
    END;
    .
SQL>
```

위의 예에서는 블록 내에 한 라인의 주석이 삽입되었으며, END 문장 아래의 9번째 라인에 블록종료 문자로 점(.)을 입력하여 PSM 프로그램의 입력을 마쳤다. 마지막 라인에서 다른 문자나 문자열 없이 오직 블록종료 문자만 단독으로 한 라인을 형성하고 있는 것을 확인할 수 있다.

참고

PSM 사용에 대한 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.

---

tbSQL 유틸리티의 명령어 입력

tbSQL 유틸리티의 명령어에는 SQL 수행과 관련된 명령어나 기타 Tibero 데이터베이스 관리를 위한 명령어 등이 포함된다. tbSQL 유틸리티의 명령어에 대한 자세한 내용은 “1.6. 명령어”를 참고한다.

1.4.2. 명령어의 실행

tbSQL 유틸리티의 명령 프롬프트에 입력된 명령어를 실행하는 방법은 다음과 같은 세 가지가 있다.

- SQL 버퍼에 저장된 SQL 문장이나 PSM 프로그램의 실행

  SQL 버퍼에는 가장 최근에 입력된 SQL 문장이나 PSM 프로그램 하나만 저장되어 있다. 이러한 SQL 문장이나 PSM 프로그램을 실행하기 위해서는 공통적으로 RUN 또는 / 명령어를 입력한다.

- SQL 문장의 실행

  전체 문장을 입력하고 세미콜론(;)으로 종료하면 SQL 문장이 바로 실행된다.

- SQL 버퍼에 저장과 동시에 실행

  SQL 문장이나 PSM 프로그램 입력을 마치고, SQL 버퍼에 저장함과 동시에 바로 실행하려면 / 명령어를 입력한다. 이때 점(.)과 마찬가지로 그 자체만으로 하나의 라인이 되어야 한다.

tbSQL 유틸리티의 명령어는 SQL 문장이나 PSM 프로그램과 달리 실행을 위한 명령어가 따로 없으며, SQL 버퍼에도 저장되지 않는다. tbSQL 유틸리티의 명령어는 입력을 마침과 동시에 실행된다.

다음은 SQL 버퍼에 저장된 SQL 문장을 실행하는 예이다.

```
SQL> SELECT ENAME, SALARY, ADDR
    FROM EMP
    -- this is a comment.
    WHERE DEPTNO = 5;
......실행 결과 ①......
SQL> /
......실행 결과 ②......
SQL>
```

위의 예에서는 첫 번째 SQL 프롬프트에서 SQL 문장을 입력하고 세미콜론(;)으로 종료하여 바로 실행하였다. 그리고 두 번째 SQL 프롬프트에서는 /를 입력하여 SQL 버퍼에 저장된 SQL 문장을 실행하였다. SQL 버퍼에는 가장 최근에 입력된 SQL 문장이 저장되어 있으므로, 첫 번째 SQL 프롬프트에서 입력한 SQL 문장과 동일한 문장이 다시 실행된다. 따라서 실행 결과 ①과 실행 결과 ②는 동일한 결과를 출력한다.

다음은 앞에서 보인 SQL 문장을 슬래시(/)를 이용하여 실행하는 예이다. 이때 SQL 문장의 맨 마지막에 세미콜론(;)을 입력하지 않는다.

```
SQL> SELECT ENAME, SALARY, ADDR
    FROM EMP
    -- this is a comment.
    WHERE DEPTNO = 5
    /
......실행 결과......
SQL>
```

1.4.3. 기타 기능

본 절에서는 tbSQL 유틸리티의 기본 기능 중에서 주로 사용되는 주석의 삽입, 자동 커밋, 운영체제 명령어 실행, 출력 내용을 저장하는 기능에 대하여 차례로 설명한다.

주석 삽입

주석은 다음의 두 가지 방법으로 삽입할 수 있다.

- /* ... */를 이용

  /* ... */를 이용하는 방법은 C나 C++ 프로그래밍 언어에서 사용하는 방법과 동일하다. tbSQL 유틸리티에서는 /*와 */로 둘러싸인 부분은 주석으로 인식하여 처리하지 않는다. 이 주석은 중첩하여 사용할 수 없다. 즉, /* …… */ 주석 내에 또 다른 /* …… */ 주석이 포함될 수 없다.

- 두 개의 마이너스 부호(--)를 이용

  두 개의 마이너스 부호(--)를 이용하면, 각 라인에서 두 개의 마이너스 부호(--) 다음부터 라인의 마지막까지를 주석으로 인식하여 무시된다. 이 주석은 /* …… */를 이용하는 방법과 마찬가지로 어떤 위치에도 올 수 있으나, PSM 프로그램 입력을 종료하기 위한 점(.)과는 같은 라인에 올 수 없다.

따라서, 다음과 같이 작성된 스크립트 파일은 실행 중에 에러를 발생한다.

```
(PSM 프로그램)
.-- 잘못된 주석
RUN
```

자동 커밋

SQL 문장으로 갱신된 내용은 트랜잭션이 커밋되기 전에는 데이터베이스에 영구적으로 반영되지 않는다. 하나의 트랜잭션은 대개 여러 개의 SQL 문장으로 이루어지며, SQL 문장으로 갱신된 내용은 데이터베이스에 바로 반영되지 않는다.

tbSQL 유틸리티에서는 SQL 문장이 실행될 때마다 COMMIT 문장을 자동으로 실행하도록 설정하거나 이 기능을 정지시킬 수 있다. 디폴트는 자동 커밋을 사용하지 않는 것이다.

SET AUTOCOMMIT 명령어로 이러한 설정을 할 수 있으며, 현재의 자동 커밋 설정을 출력하여 확인하려면 SHOW AUTOCOMMIT 명령어를 이용한다.

운영체제 명령어 실행

tbSQL 유틸리티를 시작한 상태에서 운영체제 명령어를 실행하려면 HOST 명령어를 입력한다.

다음은 확장자가 .sql인 모든 스크립트 파일을 나열하는 예이다. HOST 명령어 대신에 ! 명령어를 사용해도 같은 동작을 수행한다.

```
SQL> HOST dir *.sql
..... 운영체제 명령어 실행 결과 .....
SQL>
```

운영체제 명령어를 실행한 후에는 다시 tbSQL 유틸리티의 프롬프트가 나타나며, 계속하여 tbSQL 유틸리티의 명령어를 입력할 수 있다.

HOST 명령어나 ! 명령어 다음 문장을 생략할 경우 운영체제 명령 프롬프트가 출력된다. 다시 tbSQL 유틸리티로 돌아오기 위해서는 EXIT를 입력한다.

```
SQL> !
$ dir *.sql
..... 운영체제 명령어 실행 결과 .....
$ EXIT
SQL>
```

출력 내용 저장

tbSQL 유틸리티에서 입력하거나 출력한 모든 내용을 텍스트 파일로 저장하려면 SPOOL 명령어를 사용한다. SPOOL 명령어를 사용하면, 사용자가 입력한 SQL 문장이나 PSM 프로그램, tbSQL 유틸리티의 명령어는 물론, 질의 결과 및 프로그램 실행 결과, tbSQL 유틸리티 프롬프트까지도 파일에 저장된다.

SPOOL 명령어를 실행하면 바로 다음 라인부터 파일에 저장된다. 이 기능을 정지하려면 SPOOL OFF를 입력한다. SPOOL OFF를 입력하면 그 다음 라인부터는 파일에 저장되지 않는다.

다음은 SPOOL 명령어를 사용하는 예이다. 만약 SPOOL 명령에서 사용되는 save.txt 파일이 이미 존재한다면 이전 파일 위에 덮어 쓰게 되므로 이전의 내용은 사라진다.

```
SQL> SPOOL save.txt
Spooling is started.
SQL> SELECT
    *
    FROM DUAL;

DUMMY
-----
X

1 row selected.

SQL> SPOOL OFF
Spooling is stopped: save.txt
```

다음은 앞의 예에서 save.txt 파일에 저장된 내용이다.

```
SQL> SELECT
    *
    FROM DUAL;

DUMMY
-----
X

1 row selected.

SQL> SPOOL OFF
```

사용자가 입력한 SQL 문장, 질의 결과, 마지막의 SPOOL OFF 명령까지도 파일에 저장되어 있다.

1.5. 고급 기능

본 절에서는 tbSQL 유틸리티의 기본 기능에 비하여 좀 더 고급 사용자가 사용하는 기능으로 스크립트를 이용한 일괄 작업 실행과 Tibero 시스템을 관리하는 DBA를 위한 기능에 대하여 설명한다.

1.5.1. 스크립트 기능

스크립트는 한 번의 명령으로 일괄적으로 작업을 실행하기 위한 SQL 문장과 PSM 프로그램, tbSQL 유틸리티의 명령어의 모임이다. tbSQL 유틸리티에서 스크립트를 실행하면 그 안에 포함된 모든 명령어가 차례로 실행된다.

스크립트 생성

스크립트 파일은 외부에서 생성, 편집하여 tbSQL 유틸리티 내에서 실행할 수도 있고, tbSQL 유틸리티를 실행한 후에 외부 편집기를 호출하여 생성, 편집할 수도 있다. 외부 편집기를 호출할 경우 어떤 편집기를 이용할 것인지 설정할 수 있다.

다음은 외부 편집기로 vi를 이용하는 예이다.

```
$ export TB_EDITOR=vi
```

외부 편집기를 이용하여 특정 스크립트 파일을 편집하기 위해서는 EDIT 명령어를 사용한다. EDIT 명령어와 함께 파일명을 제시해야 하며, 확장자가 SUFFIX 시스템 변수와 같은 경우에는 생략할 수 있다.

다음은 스크립트 파일 run.sql을 편집하기 위하여 외부 편집기를 호출하는 예이다.

```
SQL> EDIT run
```

다음은 스크립트 파일 내에 SQL 문장, PSM 프로그램, tbSQL 유틸리티의 명령어를 입력하는 방법이다.

- 일반적인 입력 방법

  tbSQL 유틸리티의 명령 프롬프트에서 입력하는 방법과 거의 동일하며, 여러 라인에 걸쳐서 입력할 수 있다.

- SQL 문장과 PSM 프로그램의 종료

  SQL 문장은 반드시 항상 세미콜론(;)을 문장 끝에 입력해야 하며, PSM 프로그램은 마지막 라인을 점(.)만으로 마쳐야 한다.

- 주석의 삽입

  스크립트 파일 내에 주석을 삽입할 수 있다.

스크립트를 실행하면 SQL 문장은 바로 실행되며, PSM 프로그램은 RUN 또는 / 명령어를 입력하여 실행한다.

다음은 테이블 EMP에 대한 몇 가지 작업을 수행하는 스크립트 파일의 예이다. 라인 사이에 공백이 들어갈 수 있다.

```
-- SQL 문장
SELECT ENAME, SALARY, ADDR FROM EMP
    WHERE DEPTNO = 5;
UPDATE EMP SET SALARY = SALARY * 1.05
    WHERE DEPTNO = 5;

-- PSM 프로그램
DECLARE
    deptno NUMBER(2);
BEGIN
    deptno := 20;
    UPDATE EMP SET SALARY = SALARY * 1.05
    WHERE DEPTNO = deptno;
END;

RUN -- PSM 프로그램 실행
/* 최종으로 갱신된 내용을 반영한다. */
COMMIT;
```

스크립트 실행

스크립트 파일을 실행하려면 START 또는 @ 명령어를 사용한다. 명령어와 함께 파일명을 지정하며 확장자가 SUFFIX 시스템 변수(sql)와 동일한 경우에는 생략해도 무방하다.

스크립트 파일이 현재 디렉터리에 있는 경우에는 바로 실행하고, 그렇지 않은 경우에는 환경변수 **TB_SQLPATH**에 설정된 디렉터리에서 찾는다.

다음의 두 라인은 스크립트 파일 run.sql을 실행하는 예이며 결과는 동일하다.

```
SQL> START run
SQL> @run
```

스크립트 파일 내에서 하나 이상의 다른 스크립트 파일을 실행할 수도 있다. 즉, 스크립트 파일 내에 START 또는 @ 명령어를 포함할 수 있다. 스크립트 파일을 재귀적으로 실행할 때 무한 루프에 빠지지 않도록 해야 한다. tbSQL 유틸리티를 시작할 때 @ 명령어를 사용하면 시작과 동시에 스크립트 파일을 실행할 수 있다. 이러한 방법은 운영체제에서 배치 프로그램을 실행할 때 필요하다.

다음은 tbSQL 유틸리티를 시작하면서 동시에 스크립트 파일 run.sql을 실행하는 예이다.

```
$ tbsql dbuser/dbuserpassword @run
```

또는 다음과 같이 셸의 리다이렉션 명령을 이용하여 스크립트를 실행시키는 것도 가능하다.

```
$ tbsql dbuser/dbuserpassword < run.sql
```

1.5.2. DBA를 위한 기능

tbSQL 유틸리티로 DBA 기능을 수행할 수 있다. tbSQL 유틸리티로 DBA 기능을 수행하려면 먼저 DBA 권한을 가진 사용자로 Tibero에 로그인한다.

다음은 DBA 권한을 가지고 있는 SYS 사용자로 로그인을 하는 예이다.

```
$ tbsql sys/syspassword
```

tbSQL 유틸리티를 시작한 후에도 DBA로 연결할 수 있다. 이때 CONNECT 명령어를 사용하며 앞에서와 마찬가지로 DBA 권한을 가진 사용자로 연결하면 된다.

다음은 CONNECT 명령어를 이용하여 DBA로 연결하는 예이다.

```
SQL> CONNECT sys/syspassword
```

다음은 DBA가 tbSQL 유틸리티로 수행할 수 있는 기능이다.

- Tibero의 종료

  DBA 권한을 가진 사용자는 tbSQL 명령어인 TBDOWN으로 Tibero를 종료한다.

1.5.3. 사용자 접근 제어 기능

tbSQL 유틸리티는 특정 사용자가 수행하는 명령어를 제한할 수 있는 기능을 제공한다.

이 기능을 위해 tbSQL 유틸리티는 SYS 사용자에 생성된 CLIENT_ACCESS_POLICY 테이블을 참조하며, 이 테이블은 DBA로 하여금 특정 사용자가 수행하는 명령어에 대한 접근 정책을 정의할 수 있게 한다.

tbSQL 유틸리티는 사용자가 데이터베이스에 접속할 때 해당 테이블에 있는 정보를 로딩하고, 사용자가 명령어를 수행할 때마다 권한 체크를 통해 수행 가능 여부를 판단한다. 한번 로딩된 정보는 사용자가 접속을 종료할 때 해제된다.

참고

SYS 사용자로 접속할 경우엔 권한 체크를 수행하지 않는다.

접근 제어 테이블 생성

접근 제어 테이블은 SYS 사용자로 접속하여 $TB_HOME/scripts/client_policy.sql 파일을 실행시켜 생성한다.

참고

만약 테이블이 존재하지 않거나 제대로 생성되지 않은 경우엔 권한 체크를 하지 않는다.

CLIENT_ACCESS_POLICY 테이블의 세부 내용은 다음과 같다.

```
CLIENT  VARCHAR2(32)   NOT NULL
USERID  VARCHAR2(128)
ACTION  VARCHAR2(256)
POLICY  VARCHAR2(64)
```

항목	설명
CLIENT	클라이언트 프로그램 이름이다. 대소문자를 구분하며, tbSQL을 사용한다.
USERID	제한할 사용자 이름이다. 다음과 같이 사용할 수 있으며 와일드 카드(%)를 허용한다. – TIBERO – T% (T로 시작하는 모든 사용자) – % (모든 사용자)
ACTION	제한할 명령어다.
POLICY	제한 정책이다. DISABLED를 사용한다.

명령어 제한 설정

tbSQL 명령어나 SQL, PSM 명령어를 제한하기 위해선 다음 예제와 같이 CLIENT_ACCESS_POLICY 테이블에 로우를 삽입한다. 만약 다시 명령어를 허용하기 위해선 해당 로우를 삭제한다.

```
CLIENT   USERID       ACTION           POLICY
-------- ------------ ---------------- ----------
tbSQL    TIBERO       HOST             DISABLED
tbSQL    %            INSERT           DISABLED
tbSQL    PUBLIC       UPDATE           DISABLED
```

다음은 제한할 수 있는 명령어 목록이다.

- **tbSQL 명령어**

```
ACCEPT          APPEND          ARCHIVE         CHANGE
CLEAR           COLUMN          CONNECT         DEFINE
DEL             DESCRIBE        DISCONNECT      EDIT
EXECUTE         EXIT            EXPORT          HELP (?)
HISTORY         HOST (!)        INPUT           LIST
LOADFILE        LOOP            LS              PASSWORD
```

```
PAUSE            PING             PRINT            PROMPT
QUIT             REMARK           RESTORE          RUN
SAVE             SET              SHOW             SPOOL
START (@, @@)    TBDOWN           UNDEFINE         VARIABLE
WHENEVER
```

- **SQL 명령어**

```
ALTER            ANALYZE          AUDIT            CALL
COMMENT          COMMIT           CREATE           DELETE
DROP             EXPLAIN          FLASHBACK        GRANT
INSERT           LOCK             MERGE            NOAUDIT
PURGE            RENAME           REVOKE           ROLLBACK
SAVEPOINT        SELECT           SET CONSTRAINTS  SET ROLE
SET TRANSACTION  TRUNCATE         UPDATE
```

- **PSM 명령어**

```
BEGIN             DECLARE
```

다음은 사용자 접근 제어 기능을 사용한 예제이다.

```
..... SYS 사용자로 접속 .....
SQL> CONNECT SYS
..... 접근 제어 설정 .....
SQL> INSERT INTO CLIENT_ACCESS_POLICY VALUES ('tbSQL', 'TIBERO', 'SELECT', 'DISABLED');

..... TIBERO 사용자로 접속 .....
SQL> CONNECT TIBERO
SQL> SELECT * FROM DUAL;
TBS-70082: The 'SELECT' command has been disabled.
```

1.5.4. 접속 정보 암호화 기능

tbSQL 유틸리티는 데이터베이스 접속 정보(connect_string)를 암호화 파일(wallet)로 저장하고 사용하는 기능을 제공한다. 환경변수인 ISQL_WALLET_PATH에 지정한 경로의 파일에 tbSQL 유틸리티에서 접속한 데이터베이스의 정보를 암호화 파일로 만들거나 다음 접속 시 사용할 수 있다.

암호화 파일 생성

tbSQL 유틸리티를 이용해서 특정 데이터베이스에 접속한 후 SAVE CREDENTIAL 명령어를 통해서 암호화 파일을 생성할 수 있다.

다음은 ISQL_WALLET_PATH의 값을 현재 경로의 wallet.dat 파일로 설정한 후 접속한 데이터베이스 정보를 암호화하여 생성하는 예이다.

```
$ export ISQL_WALLET_PATH=./wallet.dat
$ tbsql

tbSQL 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.

Can't login with the wallet file.
Login the database and SAVE CREDENTIAL again.

Enter Username: dbuser
Enter Password: dbuserpassword
Connected to Tibero.

SQL> SAVE CREDENTIAL
Complete to generate the wallet file.
```

tbSQL 유틸리티를 실행하기에 앞서 ISQL_WALLET_PATH 환경변수를 설정했기 때문에 ./wallet.dat 파일의 내용을 복호화하려고 시도하지만 해당 파일이 존재하지 않을 경우 위와 같은 에러가 발생한다.

데이터베이스에 정상적으로 다시 접속 후 SAVE CREDENTIAL 명령어를 통해서 ./wallet.dat 파일이 생성된다.

다음은 ISQL_WALLET_PATH 환경변수 설정없이 현재 경로의 wallet.dat 파일에 접속한 데이터베이스 정보를 암호화하여 생성하는 예이다.

```
$ tbsql

tbSQL 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.

SQL> CONN dbuser/dbuserpassword
Connected to Tibero.

SQL> SAVE CREDENTIAL "./wallet.dat"
Complete to generate the wallet file.
```

SAVE CREDENTIAL의 파라미터로 ./wallet.dat 파일 경로를 주면, 데이터베이스 접속한 정보를 암호화하여 ./wallet.dat 파일을 생성한다.

암호화 파일 사용

tbSQL 유틸리티 실행 전 환경변수 ISQL_WALLET_PATH의 값으로 위에서 생성한 암호화 파일(wallet)을 지정하면 암호화 파일 생성 전 접속한 정보를 다시 이용할 수 있다.

다음은 ISQL_WALLET_PATH의 값의 파일을 사용해서 데이터베이스에 접속하는 예이다.

```
$ tbsql

tbSQL 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.

Connected to Tibero.
```

참고

암호화 파일을 사용하기 위해서는 반드시 환경변수 ISQL_WALLET_PATH를 설정해야 한다.

암호화된 파일은 암호화 파일을 생성한 tbSQL 유틸리티에서만 사용이 가능하다. 해당 파일을 다른 tbSQL 유틸리티에서 사용하고 싶다면 위 과정을 통해서 암호화 파일을 다시 생성해야만 한다.

Windows 환경에서는 위의 암호화 파일 생성 및 사용 기능은 사용할 수 없다.

1.6. 명령어

본 절에서는 tbSQL 유틸리티가 제공하는 명령어를 자세히 설명한다.

다음은 tbSQL 유틸리티의 명령어를 표현하는 문법의 예이다.

```
COM[MAND] param {choice1|choice2} [option] [arg]*
```

위의 예를 기준으로 tbSQL 유틸리티에서 사용하는 명령어의 문법을 해석하는 방법은 다음과 같다.

항목	설명
대괄호([ ])	대괄호([ ])에 포함된 내용은 입력하지 않아도 명령어를 실행할 수 있다. 위의 예에서 COMMAND 명령어의 뒷부분(MAND)과 option, arg는 명령 프롬프트에 포함되지 않을 수 있다.
중괄호({ })	중괄호({ })에 포함된 내용은 반드시 입력해야 명령어를 실행할 수 있다. 위의 예에서 choice1과 choice2는 중괄호({ }) 내에 있고 버티컬 바(|)로 분리되어 있으므로 둘 중 하나는 명령 프롬프트에 포함되어야 한다.
버티컬 바(|)	버티컬 바(|)로 분리된 내용은 그 중 하나를 선택한다.

항목	설명
애스터리스크(*)	애스터리스크(*)로 표시된 내용은 포함되지 않을 수도 있고, 여러 번 포함될 수도 있다. 위의 예에서 arg는 대괄호([ ]) 바로 뒤에 애스터리스크(*)가 있으므로 포함되지 않을 수도 있고, 한 번 이상 여러 번 포함될 수도 있다.
이탤릭체	이탤릭체로 표시된 내용은 명령어에 따라 적절한 문자열로 대체되어야 한다.
대소문자	명령어는 대소문자를 구분하지 않는다.

다음에 나열된 세 개의 명령어는 위의 명령어 문법 표현에 따라 모두 유효하다.

```
COMMAND param choice1
COM param choice1 option
COM param choice2 arg1 arg2 arg3
```

tbSQL 유틸리티의 명령어에는 SQL 문장의 수행 또는 데이터베이스 관리에 필요한 명령어가 포함되어 있다. 각 명령어를 알파벳 순으로 나열하고, 문법(syntax), 파라미터, 예제의 순서로 설명한다.

tbSQL 유틸리티에서 사용할 수 있는 명령어는 다음과 같다.

명령어	설명
!	운영체제의 명령어를 실행하는 명령어이다. HOST 명령어와 동일하다.
%	히스토리 버퍼에 저장된 명령어를 재수행하는 명령어이다.
@, @@	스크립트를 실행하는 명령어이다. START 명령어와 동일하다.
/	현재 SQL 버퍼 내의 SQL 문장 또는 PSM 프로그램을 실행하는 명령어이다. RUN 명령어와 동일하다.
ACCEPT	사용자의 입력을 받아 치환 변수의 값을 설정하는 명령어이다.
APPEND	SQL 버퍼에 특정 텍스트 혹은 statement를 덧붙이는 명령어이다.
ARCHIVE LOG	Redo 로그 파일 정보를 출력하는 명령어이다.
CHANGE	SQL 버퍼의 현재 라인에서 패턴 문자를 찾아 주어진 문자로 변환하는 명령어이다.
CLEAR	설정된 옵션을 초기화하거나 지우는 명령어이다.
COLUMN	컬럼의 출력 속성을 설정하는 명령어이다.
CONNECT	특정 사용자 ID로 데이터베이스에 접속하는 명령어이다.
DEFINE	치환 변수를 정의하거나 출력하는 명령어이다.
DEL	SQL 버퍼에 저장된 라인을 지우는 명령어이다.
DESCRIBE	지정된 객체의 컬럼 정보를 출력하는 명령어이다.
DISCONNECT	현재 데이터베이스로부터 접속을 해제하는 명령어이다.

명령어	설명
EDIT	특정 파일 또는 SQL 버퍼의 내용을 외부 편집기를 이용하여 편집하는 명령어이다.
EXECUTE	단일 PSM 문장을 수행하는 명령어이다.
EXIT	tbSQL 유틸리티를 종료하는 명령어이다. QUIT 명령어와 동일하다.
EXPORT	SELECT 문장의 수행 결과나 테이블 데이터를 파일로 출력하는 명령어이다.
HELP	도움말을 출력하는 명령어이다.
HISTORY	실행한 명령어의 히스토리를 출력하는 명령어이다.
HOST	운영체제 명령어를 실행하는 명령어이다. ! 명령어와 동일하다.
INPUT	SQL 버퍼의 현재 라인 뒤에 새로운 라인을 추가하는 명령어이다.
LIST	SQL 버퍼 내의 특정 내용을 출력하는 명령어이다.
LOADFILE	Tibero의 테이블을 Oracle의 SQL*Loader 툴이 인식할 수 있는 형식으로 저장하는 명령어이다.
LOOP	단일 명령어를 무한 반복 수행하는 명령어이다.
LS	현재 사용자가 생성한 데이터베이스 객체를 출력하는 명령어이다.
PASSWORD	사용자 패스워드를 변경하기 위한 명령어이다.
PAUSE	사용자가 <Enter> 키를 누를 때까지 실행을 멈추는 명령어이다.
PING	특정 데이터베이스에 대해 접속가능한 상태인지를 출력하는 명령어이다.
PRINT	사용자가 정의한 바인드 변수의 값을 출력하는 명령어이다.
PROMPT	사용자가 정의한 SQL 문장이나 빈 라인을 그대로 화면에 출력하는 명령어이다.
QUIT	tbSQL 유틸리티를 종료하는 명령어이다. EXIT 명령어와 동일하다.
RESTORE	선택한 정보를 파일로부터 복원하는 명령어이다.
RUN	현재 SQL 버퍼 내의 SQL 문장이나 PSM 프로그램을 실행하는 명령어이다. / 명령어와 동일하다.
SAVE	선택한 정보를 파일에 저장하는 명령어이다.
SET	tbSQL 유틸리티의 시스템 변수를 설정하는 명령어이다.
SHOW	tbSQL 유틸리티의 시스템 변수를 출력하는 명령어이다.
SPOOL	화면에 출력되는 내용을 모두 외부 파일에 저장하는 과정을 시작하거나 종료하는 명령어이다.
START	스크립트 파일을 실행하는 명령어이다. @ 명령어와 동일하다.
TBDOWN	Tibero를 종료하는 명령어이다.
UNDEFINE	하나 이상의 치환 변수를 삭제하는 명령어이다.
VARIABLE	사용자가 정의한 바인드 변수를 선언하는 명령어이다.
WHENEVER	에러가 발생한 경우의 동작을 정의하는 명령어이다.

1.6.1. !

tbSQL 유틸리티 내에서 운영체제의 명령어를 실행한다. ! 명령어 대신에 HOST 명령어를 사용할 수 있다.

! 명령어의 세부 내용은 다음과 같다.

- 문법

```
! [command]
```

항목	설명
	운영체제의 명령어 없이 ! 명령어만 입력하면 운영체제의 명령 프롬프트로 나가서 운영체제의 명령어를 여러 번 입력할 수 있다. 이때 다시 tbSQL 유틸리티로 돌아오려면 EXIT 명령어를 입력한다.
command	운영체제의 명령어이다.

- 예제

```
SQL> ! dir *.sql
SQL> !
```

1.6.2. %

tbSQL 유틸리티 내에서 히스토리 버퍼에 저장된 명령어를 다시 입력할 필요없이 재수행한다.

% 명령어의 세부 내용은 다음과 같다.

- 문법

```
% number
```

항목	설명
number	히스토리 버퍼에 저장된 명령어 번호이다.

- 예제

```
SQL> history
  1: set serveroutput on
  2: set pagesize 40
  3: select 1 from dual;
SQL> %3

          1
-----------
```

```
          1

1 row selected.
```

1.6.3. @, @@

스크립트 파일을 실행한다. 스크립트 파일이 SUFFIX 시스템 변수에 등록된 확장자를 가지면 스크립트 파일의 이름을 확장자 없이 지정할 수 있다. tbSQL 유틸리티는 지정된 스크립트 파일을 현재 디렉터리 내에서 찾는다. 스크립트 파일의 실행 전에 SET 명령어로 설정된 시스템 변수는 스크립트 파일을 실행하는 중에도 유효하다. 스크립트 파일 내에서 EXIT 또는 QUIT 명령어를 실행하면 tbSQL 유틸리티를 종료한다.

@ 명령어 대신에 START 명령어를 사용할 수 있다.

@ 명령어의 세부 내용은 다음과 같다.

- 문법

```
@ {filename}
```

항목	설명
filename	스크립트 파일의 이름이다.

- 예제

```
SQL> @ run
SQL> @ run.sql
```

1.6.4. /

현재 SQL 버퍼 내의 SQL 문장 또는 PSM 프로그램을 실행한다. / 명령어 대신에 RUN 명령어를 사용할 수 있다.

/ 명령어의 세부 내용은 다음과 같다.

- 문법

```
/
```

- 예제

```
SQL> SELECT * FROM DUAL;
..... SQL문장 실행 결과 .....
```

```
SQL> /
..... 위와 동일한 결과 .....
```

1.6.5. ACCEPT

사용자의 입력을 받아 치환 변수의 값을 설정한다. 이때 설정된 치환 변수의 값은 이후에 사용자가 입력하는 SQL 문장이나 PSM 프로그램에서 **&variable**과 일치하는 단어가 있을 경우 자동으로 치환된다.

ACCEPT 명령어의 세부 내용은 다음과 같다.

- 문법

```
ACC[EPT] variable [FOR[MAT] format] [DEF[AULT] default]
[PROMPT statement|NOPR[OMPT]] [HIDE]
```

항목	설명
variable	저장할 치환 변수의 이름이다. 만약 존재하지 않을 경우 새로 생성한다.
FOR[MAT] *format*	치환 변수의 포맷이다. 만약 치환 변수의 값과 포맷이 일치하지 않을 경우 에러가 발생한다.
DEF[AULT] *default*	사용자로부터 입력값이 없을 경우 대신 사용할 치환 변수 값이다.
PROMPT *statement*	사용자로부터 치환 변수의 값을 입력 받기 전에 프롬프트를 화면에 출력한다.
NOPR[OMPT]	프롬프트를 출력하지 않고 사용자의 입력을 기다린다.
HIDE	사용자로부터의 입력값이 출력되는 것을 방지한다.

- 예제

```
SQL> ACCEPT name PROMPT 'Enter name : '
Enter name : 'John'
SQL> SELECT &name FROM DUAL;
At line 1, column 8
old value : SELECT &name FROM DUAL
new value : SELECT 'John' FROM DUAL

'JOHN'
------
John

1 row selected.

SQL>
```

1.6.6. APPEND

SQL 버퍼에 사용자가 입력한 텍스트 또는 statement를 덧붙인다.

APPEND 명령어의 세부 내용은 다음과 같다.

- 문법

```
A[PPEND] statement
```

항목	설명
statement	SQL 버퍼에 덧붙일 텍스트를 의미한다.

- 예제

```
SQL> LIST 2
..... 두 번째 라인을 선택한다. .....
SQL> APPEND text
..... 두 번째 라인에 text를 추가한다. .....
```

1.6.7. ARCHIVE LOG

Archive 로그 파일 정보를 출력한다.

ARCHIVE LOG 명령어의 세부 내용은 다음과 같다.

- 문법

```
ARCHIVE LOG LIST
```

- 예제

```
SQL> ARCHIVE LOG LIST

NAME                           VALUE
------------------------------ -------------------------------------------------
Database log mode              Archive Mode
Archive destination            /home/tibero/database/tibero/archive
Oldest online log sequence     300
Next log sequence to archive   302
Current log sequence           302

SQL>
```

1.6.8. CHANGE

SQL 버퍼에 있는 문장의 현재 라인에서 첫 번째 old 패턴을 찾아 new 패턴으로 변환한다. 일반적으로 마지막으로 실행된 SQL 문장의 현재 라인은 가장 마지막 라인이다. 현재 라인을 변경하려면 예제를 참고한다.

CHANGE 명령어의 세부 내용은 다음과 같다.

- 문법

```
C[HANGE] delim old [delim [new [delim [option]]]]
```

  – 입력 항목

항목	설명
delim	숫자를 제외한 구분자이다. 반드시 old나 new 패턴에 없는 문자를 사용해야 한다.
old	바꾸려고 하는 패턴이다. 대소문자를 구분하지 않는다. 이때 사용되는 단어는 일반적인 단어(예: dual, ksc911 등) 외에도 임의의 패턴을 나타내는 **'...'**을 사용할 수 있다. 사용방법은 예제를 참조한다.

  – [option]

항목	설명
delim	숫자를 제외한 구분자이다. 반드시 old나 new 패턴에 없는 문자를 사용해야 한다.
new	새로 치환할 패턴이다.
option	– g : 현재 라인에서 전체 패턴을 바꾼다. – c : 현재 라인에서 사용자의 선택에 따라 바꾼다. – a : 전체 문장에서 전체 패턴을 바꾼다.

- 예제

  현재 라인은 디폴트로 항상 마지막 라인을 가리키므로 두 번째 라인에 있는 DUAL이 T로 변환된다.

```
SQL> SELECT *
    FROM DUAL;
..... SQL 실행결과 .....
SQL> C/DUAL/T
```

```
    FROM T
SQL>
```

현재 라인을 바꾸기 위해서는 원하는 라인 번호를 입력한다.

```
SQL> 5
    5 WHERE ROWNUM < 5 AND
```

임의의 패턴을 나타내기 위해서 '...'을 사용할 수 있다. 이때 '...'은 앞, 뒤, 가운데에 올 수 있다.

```
SQL> CHANGE /RE...AND/RE ROWNUM >= 5 AND/
    5 WHERE ROWNUM >= 5 AND
SQL> CHANGE /...AND/WHERE ROWNUM < 3/
    5 WEHRE ROWNUM < 3
SQL> CHANGE /WHE.../WHERE ROWNUM < 5 AND/
    5 WHERE ROWNUM < 5 AND
```

a 옵션을 지정할 경우 전체 문장에서 주어진 패턴을 찾아 전부 바꾼다. 따라서 첫 번째 라인에 있는 '*'가 문자열로 변환된다.

```
SQL> SELECT *
    FROM DUAL;
..... SQL 실행결과 .....
SQL> C/*/'replaced'/a
    SELECT 'replaced'
    FROM DUAL;
SQL>
```

1.6.9. CLEAR

설정된 옵션을 초기화하거나 지운다.

CLEAR 명령어의 세부 내용은 다음과 같다.

- 문법

```
CL[EAR] [option]
```

  – [option]

항목	설명
BUFF[ER]	SQL 버퍼에 있는 모든 내용을 삭제한다.
SCR[EEN]	화면에 있는 모든 내용을 삭제한다.

항목	설명
COL[UMNS]	등록된 모든 컬럼의 출력 속성을 초기화한다.

- 예제

```
SQL> CLEAR BUFFER
SQL buffer is cleared
SQL> CLEAR SCREEN
SQL> CLEAR COLUMNS
```

1.6.10. COLUMN

컬럼의 출력 속성을 지정한다. 컬럼의 이름만 명시했을 경우에는 해당 컬럼의 속성을 출력하고, 컬럼의 이름을 명시하지 않을 경우에는 등록된 모든 컬럼을 출력한다.

COLUMN 명령어의 세부 내용은 다음과 같다.

- 문법

```
COL[UMN] [name [option]]
```

  - 입력 항목

항목	설명
name	속성을 지정할 컬럼 이름이다.

  - [option]

항목	설명
CLE[AR]	컬럼의 출력 속성을 초기화한다.
FOR[MAT] *text*	컬럼의 포맷을 지정한다. 자세한 내용은 "1.7. 컬럼 포맷"을 참고한다.
HEA[DING] *text*	컬럼의 머리글을 설정한다.
NEW_V[ALUE] variable	컬럼값을 저장할 변수를 설정한다.
WRA[PPED]	컬럼 데이터의 길이가 너무 길 경우 초과되는 데이터를 두 라인으로 나눈다.
TRU[NCATED]	컬럼 데이터의 길이가 너무 길 경우 초과되는 데이터를 자른다.
ON	컬럼의 출력 속성을 켠다.
OFF	컬럼의 출력 속성을 끈다.

- 예제

```
SQL> COLUMN
SQL> COLUMN empno
SQL> COLUMN empno CLEAR
SQL> COLUMN empno FORMAT 999,999
SQL> COLUMN ename FORMAT A10
SQL> COLUMN sal HEADING the salary of this month
SQL> COLUMN sal OFF
SQL> COLUMN job WRAPPED
SQL> COLUMN job TRUNCATED
```

1.6.11. CONNECT

다른 사용자로 Tibero 데이터베이스에 접속한다. 만약 사용자의 이름 또는 패스워드를 입력하지 않은 경우에는 tbSQL 유틸리티에서 프롬프트를 출력하고 입력을 요구한다.

CONNECT 명령어를 실행하면 이전에 실행되던 트랜잭션을 커밋시키고, 이전 접속을 해제하고 나서 새로운 접속을 시도한다. 만약 새로운 접속이 실패하더라도 이전의 접속을 복구하지 않는다.

CONNECT 명령어의 세부 내용은 다음과 같다.

- 문법

```
CONN[ECT] {username[/password[@connect_identifier]]}
```

항목	설명
username	사용자의 이름이다.
password	사용자의 패스워드이다.
connect_identifier	데이터베이스에 접속하기 위한 접속 정보이다. $TB_HOME/client/config 디렉터리의 tbdsn.tbr 파일에서 지정할 수 있으며 HOST, PORT, DB_NAME 정보로 구성되어 있다. Windows에서는 데이터 원본(ODBC)에도 지정할 수 있으며 이를 가장 먼저 검색한다.

- 예제

```
SQL> CONNECT dbuser/dbuserpassword@db_id
```

1.6.12. DEFINE

치환 변수를 정의하거나 출력한다.

DEFINE 명령어의 세부 내용은 다음과 같다.

- 문법

```
DEF[INE] [variable]|[variable = value]
```

항목	설명
	치환 변수의 이름을 지정하지 않으면, 전체 치환 변수를 출력한다.
variable	지정한 치환 변수를 출력한다.
variable = *value*	정의할 치환 변수의 이름과 기본값이다.

- 예제

```
SQL> DEFINE NAME
..... NAME이라는 이름을 갖는 치환 변수를 화면에 출력한다. .....
SQL> DEFINE NAME = 'SMITH'
..... NAME이라는 이름을 갖는 치환 변수와 SMITH라는 기본값을 정의한다. .....
SQL> DEFINE
..... 전체 치환 변수를 화면에 출력한다. .....
```

1.6.13. DEL

SQL 버퍼에 설정된 라인을 지운다. 만약 라인 번호를 생략할 경우 전체 라인을 삭제한다.

DEL 명령어의 세부 내용은 다음과 같다.

- 문법

```
DEL [number|number number|number LAST|LAST]
```

항목	설명
number	지정된 번호의 라인을 삭제한다.
number number	첫 번째로 지정된 번호의 라인부터 두 번째로 지정된 번호의 라인까지 삭제한다.
number LAST	첫 번째로 지정된 번호의 라인부터 마지막 라인까지 삭제한다.
LAST	마지막 라인을 삭제한다.

- 예제

```
SQL> DEL 1
..... 첫 번째 라인을 삭제한다. .....
SQL> DEL 1 3
..... 첫 번째부터 세 번째 라인까지 삭제한다. .....
```

```
SQL> DEL 1 LAST
..... 첫 번째부터 마지막 라인까지 삭제한다. .....
SQL> DEL LAST
..... 마지막 라인을 삭제한다. .....
```

1.6.14. DESCRIBE

지정된 객체의 컬럼 정보를 출력한다. 여기에서 객체는 테이블, 뷰, 동의어, 함수, 프러시저, 패키지가 될 수 있다.

- 테이블, 뷰의 경우에는 컬럼 이름, 데이터 타입, 제약조건 및 인덱스 정보 등이 출력되며, 최대 길이, 정밀도, 스케일 등 데이터 타입에 따른 것도 포함한다.
- 함수, 프러시저의 경우에는 파라미터의 정보(이름, 데이터 타입, IN/OUT)가 출력되며, 패키지의 경우에는 소속된 모든 함수 및 프러시저의 정보가 출력된다.
- 다른 사용자가 소유한 객체의 컬럼 정보도 출력할 수 있다. 이때 소유자의 이름을 명시하며, 소유자 이름이 명시되어 있지 않으면 디폴트로 현재 사용자 소유의 객체에 해당된다.
- 지정된 객체의 소유자에 종속된 정보만 출력한다. 만약 테이블을 조회할 경우 테이블 소유자의 인덱스만 출력한다.

DESCRIBE 명령어의 세부 내용은 다음과 같다.

- 문법

```
DESC[RIBE] [schema.]object_name[@dblink]
```

항목	설명
schema	대상 객체를 포함하는 스키마(또는 소유자)이다.
object_name	컬럼 정보를 출력할 객체이다.
dblink	대상 객체를 포함하는 데이터베이스 링크이다.

- 예제

```
SQL> DESCRIBE emp
SQL> DESC scott.emp
SQL> DESC scott.emp@gateway
```

1.6.15. DISCONNECT

현재 데이터베이스로부터 접속을 종료한다. 진행 중이던 트랜잭션을 커밋하지만 tbSQL 유틸리티를 종료하지는 않는다.

스크립트 파일 내에서 CONNECT 명령어를 실행하고 그대로 종료한 경우 데이터베이스와 접속된 상태가 계속 유지된다. 따라서, 스크립트 파일 내에 CONNECT 명령어가 포함되고 있다면, 마지막에 DISCONNECT 명령어를 실행하는 것이 안전하다.

DISCONNECT 명령어의 세부 내용은 다음과 같다.

- 문법

```
DISCONN[ECT]
```

1.6.16. EDIT

특정 파일 또는 SQL 버퍼의 내용을 외부 편집기를 이용하여 편집한다. 어떤 외부 편집기를 사용할 것인지는 환경변수 $TB_EDITOR에서 설정할 수 있다.

$TB_EDITOR가 등록되지 않았다면 환경변수 $EDITOR를 참조하고, $EDITOR도 등록되지 않았다면 vi 편집기를 사용하여 편집한다. 이때 SQL 버퍼가 비어 있다면 에러를 반환한다.

편집할 파일의 확장자가 SUFFIX 시스템 변수로 지정된 디폴트 확장자이면 확장자 없이 파일의 이름을 지정할 수 있다. SUFFIX 시스템 변수의 기본값은 .sql이며, SET 명령어를 이용하여 변경할 수 있다. tbSQL 유틸리티는 현재 디렉터리에서 지정된 파일을 찾는다.

EDIT 명령어의 세부 내용은 다음과 같다.

- 문법

```
ED[IT] [filename]
```

항목	설명
	파일 이름을 지정하지 않고 EDIT 명령어를 실행하면 현재 SQL 버퍼에 저장되어 있는 내용을 편집하며 디폴트 파일인 .tbedit.sql를 사용한다. 기본 파일은 tbSQL 유틸리티가 종료될 때 자동으로 삭제된다.
filename	편집할 파일의 이름(대개 스크립트 파일의 이름)이다.

- 예제

```
SQL> EDIT run.sql
SQL> EDIT run
SQL> ED
```

1.6.17. EXECUTE

단일 PSM 문장을 수행한다. 사용할 수 있는 PSM 문장은 CALL 문장과 이름 없는 블록뿐이다. 사용자가 입력한 문장의 제일 끝에는 반드시 세미콜론(;)이 있어야 한다.

EXECUTE 명령어의 세부 내용은 다음과 같다.

- 문법

```
EXEC[UTE] {statement}
```

항목	설명
statement	단일 PSM 프로그램의 문장이다.

- 예제

```
SQL> EXECUTE begin dbms_output.put_line('success'); end;
success

PSM completed

SQL> EXECUTE call proc1();
```

또한 사용자가 정의한 바인드 변수에 값을 할당할 때에도 유용하게 사용할 수 있다.

```
SQL> VAR x NUMBER;
SQL> EXEC :x := 5;

PSM completed

SQL>
```

1.6.18. EXIT

tbSQL 유틸리티를 종료한다. 현재 진행 중이던 모든 트랜잭션을 커밋하며 데이터베이스와의 모든 접속을 종료한다.

EXIT 명령어의 세부 내용은 다음과 같다.

- 문법

```
EXIT [SUCCESS|FAILURE|WARNING|n|variable|:variable] [COMMIT|ROLLBACK]
```

항목	설명
SUCCESS	정상 종료 코드 0을 반환한다.
FAILURE	실패 종료 코드 1을 반환한다.
WARNING	경고 종료 코드 2를 반환한다.
n	종료 코드로 사용할 정수형 숫자를 직접 지정할 수 있으며, 사용할 수 있는 값의 범위는 운영체제에 따라 다르다.
variable	DEFINE 명령으로 정의한 사용자 변수나 SQL.SQLCODE와 같은 시스템 변수의 값을 반환한다. 단, 사용한 변수는 숫자형이어야 한다.
:*variable*	VARIABLE 명령으로 정의한 바인드 변수를 이용하여 종료 코드를 지정할 수 있다. 단, 사용한 바인드 변수는 숫자형이어야 한다.
COMMIT	종료하기 전 COMMIT을 수행한다.
ROLLBACK	종료하기 전 ROLLBACK을 수행한다.

1.6.19. EXPORT

SELECT 문장의 수행 결과나 테이블 데이터를 tbLoader에서 인식할 수 있는 형식의 파일로 출력한다. 고정형 포맷으로 출력하거나 각 컬럼과 로우 구분자를 지정하여 가변형 포맷으로 출력할 수 있다.

EXPORT 명령어의 세부 내용은 다음과 같다.

- 문법

```
EXP[ORT] {QUERY filename|TABLE [schema.]table} [variable_fmt|FIXED]

variable_fmt:
  [FIELDS {TERM[INATED] BY {,|text}|ENCL[OSED] BY {"|text}}]
  [LINES TERM[INATED] BY {\n|text}]
```

항목	설명
filename	출력할 파일 이름이다. 이 값은 컨트롤 파일의 대상 테이블 이름으로 사용된다.
[*schema*.]*table*	출력할 테이블 이름이다.
variable_fmt	– FIELDS ... : 가변형 포맷 컬럼 구분자이다. – LINES ... : 가변형 포맷 로우 구분자이다. – FIXED : 고정형 포맷 지시자이다.

- 예제

  아래 명령어를 수행하면 t.csv(데이터 파일), t.ctl(컨트롤 파일)이 생성된다.

```
SQL> EXPORT QUERY 't.csv'
Ready to export to 't.csv'

SQL> SELECT * FROM t;

10 rows exported.
```

가변형 포맷을 지정할 수 있다.

```
SQL> EXPORT QUERY 't.csv' FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
SQL> EXPORT QUERY 't.csv' FIELDS ENCLOSED BY '*'
```

1.6.20. HELP

지정된 단어를 포함하는 모든 항목에 대한 도움말을 화면에 출력한다.

HELP 명령어의 세부 내용은 다음과 같다.

- 문법

```
H[ELP] [topic]
```

항목	설명
	옵션을 지정하지 않으면 tbSQL 유틸리티에서 사용할 수 있는 전체 명령어를 출력한다.
topic	도움말을 출력할 단어를 지정한다.

- 예제

```
SQL> HELP SET
```

1.6.21. HISTORY

히스토리 버퍼에 저장된 명령어를 화면에 출력한다.

HISTORY 명령어의 세부 내용은 다음과 같다.

- 문법

```
HIS[TORY]
```

- 예제

```
SQL> HISTORY
..... 전체 명령어를 출력한다. .....
```

1.6.22. HOST

! 명령어와 동일하다.

HOST 명령어의 세부 내용은 다음과 같다.

- 문법

```
HO[ST] [command]
```

항목	설명
	명령어 없이 HOST만 입력하면 운영체제의 명령 프롬프트로 나가서 운영체제 명령어를 여러 번 입력할 수 있다. 이때 다시 tbSQL 유틸리티로 돌아오려면 EXIT를 입력한다.
command	운영체제의 명령어이다.

1.6.23. INPUT

SQL 버퍼에 저장된 제일 마지막 라인 뒤에 사용자가 입력한 문장을 추가한다.

INPUT 명령어의 세부 내용은 다음과 같다.

- 문법

```
I[NPUT] [statement]
```

항목	설명
	statement를 생략하면 여러 라인에 걸쳐 문장을 추가한다.
statement	추가할 SQL 문장이다.

- 예제

```
SQL> select * from dual;
..... 결과 출력 .....
SQL> LIST
     select * from dual
SQL> INPUT where rownum < 2
     select * from dual
```

```
     where rownum < 2
SQL>
```

다음은 statement 옵션을 생략한 것이다. 단, 위의 경우와 달리 입력이 끝남과 동시에 SQL 문장이 실행된다.

```
SQL> select * from dual;
..... 결과 출력 .....
SQL> INPUT
     select * from dual
     ... 여기에 입력하면 된다. ...
```

1.6.24. LIST

SQL 버퍼 내의 특정 내용을 화면에 출력한다.

LIST 명령어의 세부 내용은 다음과 같다.

- 문법

```
L[IST] [number|number number|number LAST|LAST]
```

항목	설명
	라인의 번호를 생략할 경우 전체 라인을 출력한다.
number	지정된 번호의 라인을 출력한다.
number number	첫 번째로 지정된 번호부터 두 번째로 지정된 번호까지의 라인을 출력한다.
number LAST	지정된 번호의 라인부터 마지막 라인까지 출력한다.
LAST	마지막 라인을 출력한다.

- 예제

```
SQL> LIST 1
..... 첫 번째 라인을 출력한다. .....
SQL> LIST 2 3
..... 두 번째부터 세 번째 라인까지 출력한다. .....
SQL> LIST 2 LAST
..... 두 번째부터 마지막 라인까지 출력한다. .....
SQL> LIST LAST
..... 마지막 라인을 출력한다. .....
```

1.6.25. LOADFILE

Tibero 테이블을 Oracle의 SQL*Loader가 인식할 수 있는 형식으로 저장한다.

LOADFILE 명령어의 세부 내용은 다음과 같다.

- 문법

```
LOAD[FILE] {filename}
```

항목	설명
filename	확장자를 제외한 파일의 이름이다.

- 예제

EMP라는 이름의 테이블을 Oracle의 SQL*Loader가 인식할 수 있는 파일로 저장하려면 다음과 같은 명령어를 입력한다. 이 명령을 수행하면 emp.ctl과 emp.dat라는 2개의 파일이 생성된다.

```
SQL> LOADFILE emp
SQL> select * from emp;
```

1.6.26. LOOP

임의의 문장을 무한 반복 수행하는 명령어이다. 수행을 종료하기 위해선 <Ctrl>+C를 누른다.

LOOP 명령어의 세부 내용은 다음과 같다.

- 문법

```
LOOP stmt
```

항목	설명
stmt	반복 수행할 문장이다.

- 예제

```
SQL> LOOP select count(*) from v$session
..... SQL을 1초 간격으로 반복 수행한다. .....
SQL> SET INTERVAL 10
SQL> LOOP ls
..... LS를 10초 간격으로 반복 수행한다. .....
```

1.6.27. LS

현재 사용자가 생성한 특정 타입이나 이름의 데이터베이스 객체의 정보를 출력한다.

LS 명령어의 세부 내용은 다음과 같다.

- 문법

```
LS [object_type|object_name]
```

항목	설명
	object_type이나 object_name을 생략하면 사용자가 소유한 모든 객체를 출력한다.
object_type	다음 중에 하나를 설정한다. – FUNCTION – INDEX – PACKAGE – PROCEDURE – SEQUENCE – SYNONYM – TABLE – TABLESPACE – TRIGGER – USER – VIEW
object_name	출력할 객체의 이름이다. 임의의 패턴을 나타내는 퍼센트(%) 문자를 사용할 수 있다.

- 예제

```
SQL> LS
NAME                           SUBNAME       OBJECT_TYPE
------------------------------ ------------- --------------------
SYS_CON100                                   INDEX
SYS_CON400                                   INDEX
SYS_CON700                                   INDEX
_DD_CCOL_IDX1                                INDEX
......중간 생략......
```

```
UTL_RAW                                          PACKAGE
DBMS_STATS                                       PACKGE BODY
TB_HIDDEN2                              PACKGE BODY

SQL>
..... 전체 객체를 출력한다. .....

SQL> LS TABLESPACE
TABLESPACE_NAME
------------------------------
SYSTEM
UNDO
TEMP
USR
..... 타입이 TABLESPACE인 모든 객체를 출력한다. .....

SQL> LS USER
USERNAME
------------------------------
SYS
..... 현재 시스템에 접속하고 있는 사용자를 조회한다. .....
```

1.6.28. PASSWORD

사용자의 패스워드를 바꾸기 위한 명령어이다.

PASSWORD 명령어의 세부 내용은 다음과 같다.

- 문법

```
PASSW[ORD] [username]
```

항목	설명
username	패스워드를 변경할 사용자 이름이다. 이 값이 생략된 경우 현재 접속된 사용자의 패스워드를 변경한다.

- 예제

```
SQL> PASSWORD
Changing password for 'TIBERO'

Enter old password: ...기존 패스워드를 입력한다...
Enter new password: ...새로운 패스워드를 입력한다...
Confirm new password: ...확인을 위해 한번 더 입력한다...
```

```
Password changed successfully.
SQL>
```

1.6.29. PAUSE

사용자가 <Enter> 키를 누를 때까지 잠시 수행을 멈춘다. 메시지를 입력할 경우 해당 메시지를 화면에 나타낸다.

PAUSE 명령어의 세부 내용은 다음과 같다.

- 문법

```
PAU[SE] [message]
```

항목	설명
message	사용자가 <Enter> 키를 누를 때 화면에 보여줄 메시지이다.

- 예제

```
SQL> PAUSE please enter...
please enter...
..... <Enter>키를 누른다. .....
SQL>
```

1.6.30. PING

특정 데이터베이스에 대해 접속가능한 상태인지를 출력한다.

PING 명령어의 세부 내용은 다음과 같다.

- 문법

```
PING connect_identifier
```

항목	설명
connect_identifier	접속할 데이터베이스 이름이다.

- 예제

```
SQL> PING tibero7
Server is alive.
```

```
SQL>
```

1.6.31. PRINT

사용자가 정의한 바인드 변수의 이름과 값을 출력한다.

PRINT 명령어의 세부 내용은 다음과 같다.

- 문법

```
PRI[NT] [variable...]
```

항목	설명
	variable을 생략할 경우 모든 바인드 변수를 출력한다.
variable	출력할 바인드 변수 이름의 목록이다.

- 예제

```
SQL> VARIABLE x NUMBER
SQL> EXECUTE :x := 5;
SQL> PRINT x

         X
----------
         5

SQL>
```

1.6.32. PROMPT

특정 메시지나 빈 라인을 화면에 출력한다.

PROMPT 명령어의 세부 내용은 다음과 같다.

- 문법

```
PRO[MPT] [message]
```

항목	설명
	message를 생략할 경우 빈 라인을 출력한다.

항목	설명
message	화면에 보여 줄 메시지이다.

- 예제

다음은 외부에서 작성한 SQL 파일의 예이고, 이름은 PromptUsage.sql이다.

```
PROMPT >>> Test is started.
CREATE TABLE T (c1 NUMBER);
INSERT INTO T VALUES (1);
PROMPT Value 1 is inserted.
COMMIT;
PROMPT <<< Test is ended.
```

다음은 위의 PromptUsage.sql을 실행한 결과를 보여준다.

```
SQL> @PromptUsage
>>> Test is started.
Table 'T' created.
1 row inserted.
Value 1 is inserted.
Commit succeeded.
<<< Test is ended.
File finished.

SQL>
```

1.6.33. QUIT

EXIT 명령어와 동일하다.

QUIT 명령어의 세부 내용은 다음과 같다.

- 문법

```
Q[UIT] [SUCCESS|FAILURE|WARNING|n|variable|:variable] [COMMIT|ROLLBACK]
```

1.6.34. RESTORE

사용자가 선택한 정보를 지정한 파일로부터 복원한다.

RESTORE 명령어의 세부 내용은 다음과 같다.

- 문법

```
REST[ORE] HIST[ORY] filename[.ext]
```

항목	설명
filename[.ext]	읽을 파일 이름이다. 만약 확장자가 생략된 경우 SUFFIX에 설정된 값을 사용한다.

- 예제

```
SQL> RESTORE HISTORY history.sql
```

1.6.35. RUN

/ 명령어와 동일하지만, 이 명령어를 사용할 경우 SQL 버퍼에 저장된 문장을 화면에 출력한다.

RUN 명령어의 세부 내용은 다음과 같다.

- 문법

```
R[UN]
```

- 예제

```
SQL> select 1 from dual;

         1
----------
         1

1 row selected.

SQL> RUN
   1 select 1 from dual

         1
----------
         1

1 row selected.
```

1.6.36. SAVE

사용자가 선택한 정보를 지정한 파일에 저장한다. SAVE CREDENTIAL에 대한 사용법은 “1.5.4. 접속 정보 암호화 기능”에서 구체적으로 설명한다.

SAVE 명령어의 세부 내용은 다음과 같다.

- 문법

```
SAVE CRED[ENTIAL] [filename]
SAVE HIST[ORY] filename[.ext] [CRE[ATE]|REP[LACE]|APP[END]]
```

항목	설명
CREDENTIAL	데이터베이스 접속 정보를 암호화한다. filename을 생략하면 환경변수 ISQL_WALLET_PATH에 설정된 파일에 저장한다.
HISTORY	히스토리에 저장된 명령어들을 지정한 파일로 저장한다. 만약 확장자가 생략된 경우 SUFFIX에 설정된 값을 사용한다. – CREATE : 파일이 없을 경우엔 새로 생성하며, 이미 존재할 경우엔 에러가 발생한다. (기본값) – REPLACE : 파일이 없을 경우엔 새로 생성하며, 이미 존재할 경우엔 덮어 쓴다. – APPEND : 파일이 없을 경우엔 새로 생성하며, 이미 존재할 경우엔 이어 쓴다.
filename	저장할 파일 이름이다.

- 예제

```
SQL> SAVE CREDENTIAL
SQL> SAVE CREDENTIAL "./wallet.dat"
SQL> SAVE HISTORY history.sql
SQL> SAVE HISTORY history.sql APPEND
```

1.6.37. SET

tbSQL 유틸리티의 시스템 변수를 설정한다. SET 명령어로 설정된 시스템 변수는 SHOW 명령어를 사용하여 출력한다. 단, 변경된 시스템 변수는 현재 세션 내에서만 유효하다. 각각의 시스템 변수에 대해서는 “1.3. 시스템 변수”에서 구체적으로 설명한다.

SET 명령어의 세부 내용은 다음과 같다.

- 문법

```
SET {parameter} {value}
```

항목	설명
parameter	tbSQL 유틸리티 시스템 변수의 이름이다.

항목	설명
value	tbSQL 유틸리티 시스템 변수의 값이다.

- 예제

```
SQL> SET AUTOCOMMIT ON
```

1.6.38. SHOW

tbSQL 유틸리티의 시스템 변수를 출력한다. 출력할 정보를 파라미터를 사용하여 선택할 수 있으며, 모든 정보를 출력할 수 있다.

SHOW 명령어의 세부 내용은 다음과 같다.

- 문법

```
SHO[W] {option}
```

– option

다음은 option에 입력할 수 있는 항목에 대한 설명이다.

항목	설명
system_parameter	주어진 이름에 해당하는 tbSQL 유틸리티의 시스템 변수를 출력한다.
ALL	모든 tbSQL 유틸리티 시스템 변수를 출력한다.
ERROR	앞서 발생한 PSM 프로그램의 에러를 출력한다.
PARAM[ETERS] [*name*]	주어진 이름에 해당하는 데이터베이스 시스템 변수를 출력한다. 단, 이름이 생략될 경우에는 모든 시스템 변수를 출력한다.
RELEASE	tbSQL 유틸리티의 릴리즈 정보를 출력한다.
SQLCODE	가장 최근에 수행한 SQL에 대한 SQLCODE를 출력한다.

- 예제

```
SQL> SHOW autocommit
SQL> SHOW all
SQL> SHOW error
SQL> SHOW param db_name
SQL> SHOW release
SQL> SHOW SQLCODE
```

1.6.39. SPOOL

화면에 출력되는 모든 내용에 대해 현재 디렉터리 내 파일로 생성한다. 단, HOST 명령어와 ! 명령어의 결과는 제외된다.

SPOOL 명령어의 세부 내용은 다음과 같다.

- 문법

```
SP[OOL] [filename [APP[END]]|OFF]
```

항목	설명
	SPOOL 명령어만을 입력하면 SPOOL 명령어의 현재 실행 상태를 출력한다.
filename	출력을 저장할 파일의 이름이다.
APP[END]	출력 파일의 끝에 이어붙일지 여부이다.
OFF	출력 파일의 저장을 중지한다.

- 예제

```
SQL> SPOOL report.txt
SQL> SPOOL OFF
```

1.6.40. START

@ 명령어와 동일하다.

START 명령어의 세부 내용은 다음과 같다.

- 문법

```
STA[RT] {filename}
```

항목	설명
filename	스크립트 파일의 이름이다.

1.6.41. TBDOWN

Tibero 데이터베이스를 종료한다. 긴급성에 따라 종료 옵션을 선택할 수 있으며 옵션에 따라 데이터베이스를 재시동할 때 복구 과정이 필요할 수 있다. 이 명령어를 실행하려면 SYSDBA로 데이터베이스에 접속해야 한다.

TBDOWN 명령어의 세부 내용은 다음과 같다.

- 문법

```
TBDOWN [NORMAL|POST_TX|IMMEDIATE|ABORT]
```

항목	설명
NORMAL	현재 접속 중인 모든 사용자가 접속을 종료할 때까지 기다린 후 종료한다. (기본값)
POST_TX	현재 진행 중인 트랜잭션이 종료될 때까지 기다린 후 종료한다.
IMMEDIATE	현재 진행 중인 트랜잭션을 롤백한 후 강제 종료한다.
ABORT	현재 진행 중인 트랜잭션을 롤백하지 않고 즉시 종료한다.

- 예제

```
SQL> TBDOWN
SQL> TBDOWN ABORT
```

1.6.42. UNDEFINE

ACCEPT 명령어 등으로 정의된 치환 변수를 삭제한다.

UNDEFINE 명령어의 세부 내용은 다음과 같다.

- 문법

```
UNDEF[INE] [variable...]
```

항목	설명
	[variable...]를 생략할 경우 모든 치환 변수를 삭제한다.
variable...	치환 변수 이름의 목록이다.

- 예제

```
SQL> UNDEFINE x
SQL> UNDEFINE x y z
```

1.6.43. VARIABLE

PSM 프로그램이나 SQL 문장에서 사용할 수 있는 사용자가 정의한 바인드 변수를 선언한다.

VARIABLE 명령어의 세부 내용은 다음과 같다.

- 문법

```
VAR[IABLE] [variable [datatype]]
```

항목	설명
	VARIABLE 명령어만 사용할 때에는 전체 바인드 변수를 화면에 출력한다.
variable	바인드 변수의 이름이다.
datatype	데이터 타입이다. 현재는 다음의 타입을 지원한다. – NUMBER – CHAR(n) – VARCHAR(n) – VARCHAR2(n) – NCHAR(n) – NVARCHAR2(n) – RAW(n) – BLOB – CLOB – NCLOB – DATE – TIMESTAMP – REFCURSOR

- 예제

```
SQL> VARIABLE x NUMBER
SQL> EXEC :x := 1;

PSM completed.

SQL> SELECT :x FROM DUAL;
        :x
----------
         1
```

```
1 row selected.
SQL>
```

1.6.44. WHENEVER

오류가 발생한 경우 tbSQL의 동작을 정의한다.

WHENEVER 명령어의 세부 내용은 다음과 같다.

- 문법

```
WHENEVER {OSERROR|SQLERROR}
  {EXIT [SUCCESS|FAILURE|WARNING|n|variable|:variable] |
   CONTINUE [NONE|COMMIT|ROLLBACK]}
```

  - clause1

항목	설명
OSERROR	tbSQL이 실행 중인 환경의 운영체제로부터 발생한 오류에 대해 지정한 동작을 수행한다.
SQLERROR	SQL 문장을 수행하는 중에 발생한 오류에 대해 지정한 동작을 수행한다. 단, tbSQL에서 발생한 오류는 무시한다.

  - clause2 (기본값: CONTINUE)

항목	설명
EXIT	오류가 발생할 때 프로그램을 종료한다. 반환 코드에 대한 정의는 EXIT 명령어를 참조한다.
CONTINUE	오류가 발생하더라도 계속 다음 명령을 수행한다. – NONE : 트랜잭션에 대하여 아무 처리도 하지 않는다. (기본값) – COMMIT : COMMIT을 수행한다. – ROLLBACK : ROLLBACK을 수행한다.

- 예제

```
SQL> whenever sqlerror exit failure rollback
SQL> select 1 from no_such_table;
TBR-8033: Specified schema object was not found.
at line 1, column 16:
select 1 from no_such_table
```

```
$ echo exit code: $?
exit code: 1
$
```

1.7. 컬럼 포맷

본 절에서는 tbSQL 유틸리티의 컬럼 포맷을 데이터 타입에 따라 설정하는 방법을 설명한다.

tbSQL 유틸리티의 컬럼 포맷은 이전 절에서 설명한 COLUMN 명령어를 통해 설정하고, COLUMN 명령어를 이용하여 출력한다.

1.7.1. 문자형

CHAR, NCHAR, VARCHAR, NVARCHAR 타입의 경우 데이터베이스 컬럼의 길이를 디폴트 길이로 가진다. 데이터의 값이 컬럼의 길이보다 클 때 데이터가 다음 라인에 기록되거나 잘릴 수 있는데, 문자형 포맷을 이용할 경우에는 이를 쉽게 처리할 수 있다.

문자형의 컬럼 포맷을 설정하는 세부 내용은 다음과 같다.

- 문법

```
COL[UMN] {name} FOR[MAT] A{n}
```

항목	설명
name	컬럼 이름을 지정한다.
A{*n*}	A는 소문자 a로도 사용할 수 있으며, n은 문자열 데이터의 길이를 의미한다.

- 예제

```
SQL> SELECT 'Tibero is the best choice' test FROM DUAL;

TEST
-------------------------
Tibero is the best choice

1 row selected.

SQL> COL test FORMAT a10
SQL> SELECT 'Tibero is the best choice' test FROM DUAL;

TEST
----------
Tibero is
```

```
the best c
hoice

1 row selected.

SQL>
```

1.7.2. 숫자형

숫자형의 컬럼 포맷을 설정하는 세부 내용은 다음과 같다.

- 문법

```
COL[UMN] {col_name} FOR[MAT] {fmt_str}
```

  - 입력 항목

항목	설명
col_name	컬럼 이름을 지정한다.
fmt_str	다음 표에서 설명한 컬럼 포맷을 지정한다.

  - fmt_str 포맷

    다음은 fmt_str에 지정할 수 있는 포맷이다.

포맷	설정 예	설명
쉼표(,)	9,999	주어진 위치에 쉽표(,)를 출력한다.
점(.)	9.999	정수 부분과 소수 부분을 분리하는 위치에 점(.)을 출력한다.
$	$9999	$를 맨 앞에 출력한다.
0	0999, 9990	0을 맨 앞이나 뒤에 출력한다.
9	9999	주어진 자릿수만큼 숫자를 출력한다.
B	B9999	정수 부분이 0일 경우 공백으로 치환한다.
C	C9999	주어진 위치에 ISO currency symbol을 출력한다.
D	9D999	실수의 정수와 소수를 분리하기 위해 decimal 문자를 출력한다.
EEEE	9.99EEEE	과학적 기수법에 의해 출력한다.
G	9G999	정수 부분의 주어진 위치에 그룹 분리자를 출력한다.
L	L9999	주어진 위치에 local currency symbol을 출력한다.
MI	9999MI	음수 뒤에 마이너스 기호를 출력하고, 양수 뒤에 공백을 출력한다.

포맷	설정 예	설명
PR	9999PR	음수인 경우에 '<'와 '>'로 감싸서 출력하고, 양수인 경우에 양쪽에 공백을 출력한다.
RN	RN	대문자 로마숫자로 출력한다.
rn	rn	소문자 로마숫자로 출력한다.
S	S9999, 9999S	양수/음수 기호를 맨 앞이나 뒤에 출력한다.
TM	TM	가능한 작은 수를 출력한다.
U	U9999	주어진 위치에 dual currency symbol을 출력한다.
V	99V999	10n만큼 곱한 값을 출력한다. 여기서 n은 V뒤에 오는 9의 개수이다.
X	XXXX, xxxx	16진수 형태로 출력한다.

- 예제

```
SQL> COLUMN x FORMAT 999,999
SQL> SELECT 123456 x FROM DUAL;

           X
 --------
   123,456

1 row selected.
```

1.8. 제약 사항

본 절에서는 tbSQL 유틸리티의 제약사항을 설명한다.

항목	제한
command-line 길이	최대 2047 Bytes

제2장 tbExport

본 장에서는 tbExport 유틸리티를 소개하고 사용 방법을 설명한다.

2.1. 개요

tbExport는 Tibero에서 제공하는 Export 유틸리티이다. 이 유틸리티를 통해 Tibero 데이터베이스에 저장된 스키마 객체의 전체 또는 일부를 추출해 고유 형식의 파일로 저장하므로 데이터베이스의 백업과 다른 머신 간의 데이터베이스를 전송할 때 유용하다.

tbExport 유틸리티에서 하나의 스키마 객체를 추출하면 그와 연관된 스키마 객체가 자동으로 함께 추출된다. 예를 들어 하나의 테이블을 추출하면 그 테이블에 대해 생성된 인덱스와 제약조건 등이 함께 추출된다. 필요에 따라서 연관된 일부 스키마 객체가 함께 추출되지 않도록 지정할 수 있다.

Export 모드에는 전체 데이터베이스 모드, 사용자 모드, 테이블 모드가 있다. DBA만이 사용할 수 있으며, DBA 권한을 줄 수 없을 경우 SELECT ANY DICTIONARY 권한을 부여하여 사용하는 것을 권장한다.

tbExport 유틸리티를 실행한 결과로 생성된 파일은 운영체제 파일이다. 따라서 Tibero 데이터베이스 파일과는 달리 일반 파일과 같은 작업을 실행할 수 있다. 예를 들어 파일을 FTP를 이용하여 전송하거나 CD-ROM 등에 저장하여 원격지의 Tibero 데이터베이스로 옮길 수도 있다.

Export가 실행되는 과정에서 발생하는 로그는 LOG 파라미터를 사용하여 지정한다.

다음은 tbExport 유틸리티를 실행한 결과로 생성되는 완료, 경고 및 에러 메시지에 대한 설명이다.

항목	설명
완료 메시지	Export가 성공적으로 완료된 후에 출력된다.
경고 메시지	Export가 완료되었으나 경고가 발생한 경우에 출력된다. 존재하지 않는 테이블에 대해 Export를 시도할 때와 같은 경우에 발생하며 이런 경우 tbExport 유틸리티는 경고 메시지를 출력한 후 해당 테이블을 건너뛰고 다음 객체에 대해 Export를 계속한다.
에러 메시지	Export가 실행되는 과정에서 에러가 발생하여 Export를 계속할 수 없는 경우에 출력된다. 시스템 메모리가 부족한 상황이나 Export에 필요한 뷰가 생성되지 않는 경우처럼 Export를 계속 실행할 수 없는 상황에 출력되며, 에러 메시지를 출력한 후 Export 세션을 종료한다.

tbExport 유틸리티의 특징은 다음과 같다.

- 논리적인 백업

  Tibero의 내부 스키마 및 데이터를 SQL 문장으로 추출한다.

- 서로 다른 시점의 데이터

  여러 개의 테이블을 Export할 때 추출한 각 테이블의 데이터는 동일 시점의 데이터가 아니라 Export 작업이 실행되는 시점의 순차적인 데이터이다.

- 테이블 정의를 저장

  데이터의 존재 여부에 상관 없이 테이블 정의(테이블의 DDL 스크립트)를 저장한다.

- 테이블의 재구성

  테이블 생성 후 수많은 DML 작업으로 인해 발생한 마이그레이션이 된 로우(migrated row)나 단편화(fragmentation)를 제거한다.

2.2. 빠른 시작

tbExport 유틸리티는 Tibero를 설치하는 과정에서 함께 설치되며, Tibero를 제거하면 함께 제거된다. 또한 Java 언어로 구현되어 있으며, JVM(Java Virtual Machine)이 설치되어 있는 어떤 플랫폼에서도 바로 운영할 수 있다.

2.2.1. 실행 전 준비사항

tbExport 유틸리티를 실행하기 전에 다음과 같은 사항을 준비해야 한다.

- JRE 1.4.2 이상 설치
- Tibero 데이터베이스 서버와 같은 플랫폼에 설치되어 있거나 네트워크로 연결된다.
- 실행에 필요한 클래스 라이브러리(기본 위치: $TB_HOME/client/lib/jar 디렉터리)
  - tbExport 클래스 : expimp.jar
  - 유틸리티 공통 라이브러리 : toolcom.jar
  - 유틸리티 공통 Logger 라이브러리 : msllogger-14.jar
  - JDBC 드라이버 : internal-jdbc-14.jar

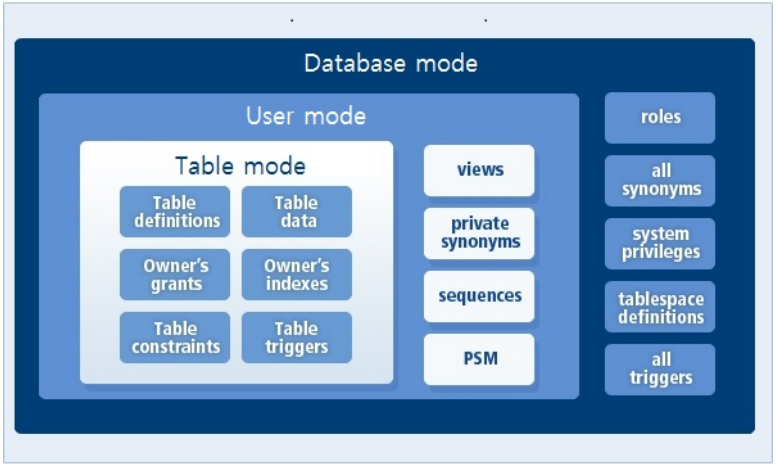
실행에 필요한 클래스 라이브러리는 Tibero를 설치하는 과정에 함께 설치되므로 추가적인 작업을 할 필요는 없다.

2.2.2. Export 모드

Export 모드에는 전체 데이터베이스 모드, 사용자 모드, 테이블 모드가 있다. 각 모드는 파라미터를 사용하여 지정할 수 있다.

다음은 모드별로 Export하는 스키마 객체의 포함 관계를 나타내는 그림이다.

[그림 2.1] Export 모드



전체 데이터베이스 모드

전체 데이터베이스 모드는 Tibero 데이터베이스 전체를 Export하기 위한 모드이다. SYS 사용자를 제외한 모든 사용자의 객체를 Export하기 위해 사용한다.

전체 데이터베이스 모드를 사용하려면 다음과 같이 FULL 파라미터를 Y로 설정한다.

```
FULL=Y
```

사용자 모드

사용자 모드는 SYS 사용자를 제외한 지정된 사용자가 소유한 모든 스키마 객체를 Export하는 모드이다. 지정한 사용자가 소유한 객체를 Export하기 위해 사용하며 DBA는 하나 이상의 사용자에게 이 모드를 사용할 수 있다.

사용자 모드를 사용하려면 다음과 같이 USER 파라미터를 USER=userlist 형태로 설정한다.

```
USER=SCOTT, USER1, …
```

테이블 모드

테이블 모드는 하나 이상의 테이블을 지정하여 그 테이블과 연관된 인덱스 등의 스키마 객체를 함께 Export하는 모드이다.

테이블 모드를 사용하려면 다음과 같이 TABLE 파라미터를 TABLE=tablelist 형태로 설정한다. 주의할 점은 SCOTT.EMP와 같이 테이블을 소유한 사용자를 반드시 명시해야 한다는 것이다.

```
TABLE=SCOTT.EMP, USER1.TABLE1, …
```

2.2.3. 실행

tbExport 유틸리티를 실행하려면 **$TB_HOME/client/bin** 디렉터리에서 **tbexport** 명령어를 입력한다.

다음은 전체 데이터베이스 모드로 실행하는 예이다.

[예 2.1] tbExport 유틸리티의 실행

```
$ tbexport username=tibero password=tmax sid=tibero file=export.dat full=y
$ tbexport cfgfile=export.cfg
```

2.3. tbExport 유틸리티

본 절에서는 명령 프롬프트에서 지정할 수 있는 tbExport 유틸리티의 파라미터를 설명한다.

사용자가 파라미터 값을 지정하지 않고 tbExport를 실행하면, 다음과 같이 명령 프롬프트에서 지정할 수 있는 파라미터의 목록과 사용법이 나타난다.

```
tbExport 7.0 102665 TmaxData Corporation Copyright (c) 2008-. All rights reserved.
Usage: tbexport [options] [parameter=value parameter=value ...]

Options:
  -h|--help        Display the more detailed information.
  -v|--version     Display the version information.
  -p|--patch       Display the binary's patch information.

Parameters:
  CFGFILE      Config file name
  COMPRESS     Compress Mode: Y/N, default: N
  CONSTRAINT   Export Constraint: Y/N, default: Y
  CONSISTENT   Consistent Mode: Y/N, default: N
  ENCRYPTION   Specifies a Encryption Mode
  ENCRYPTION_PASSWORD Specifies a password for encrypting data
                      in the export Dump File
  EXCLUDE      Limit the export to specific objects
  FILE         Export dump file name, default: default.dat
  FULL         Full Mode: Y/N, default: N
  GEOM_ASBYTES Export the geometry columns as bytes, default: N
  GRANT        Export Grant: Y/N, default: Y
  INDEX        Export Index: Y/N, default: Y
```

```
INLINE_CONSTRAINT Use the Inline Constraint: Y/N, default: N
             (this option is only supported for the not null)
IP           IP address, default: localhost
LOG          Export script log file name
LOGDIR       Export log directory
NO_PACK_DIR  Export unpacked dump files to specified directory.
             If this option is specified, FILE parameter will be ignored.
OVERWRITE    Overwrite datafile if same file name exists: Y/N, default: N
PACK_TYPE    Packing algorithm: TAR/ZIP, default: TAR
PASSWORD     User password
PORT         PORT number, default: 8629
QUERY        Where predicate: (Optional) to filter data to be exported
             (must be used with TABLE parameter.)
REMAP_TABLESPACE Remaps the objects from the source tablespace to the target
                 tablespace.
REMAP_TABLE  Remaps the objects from the source table to the target table.
ROWS         Export Table Rows: Y/N, default: Y
SAVE_CREDENTIAL Save your username and password to specified file
SCRIPT       LOG THE DDL SCRIPT: Y/N, default: N
SID          Database name
STATISTICS   Export Statistics: Y/N, default: N
TABLE        Table Mode: table name list
             Append :<Partition Name> to select a single partition (Optional)
TARGETDB     Target Server, default: TIBERO
TEMP_DIR     Directory for the temporary raw dump files.
THREAD_CNT   Thread Count, default: 4
USER         User Mode: user name list
USERNAME     Database user name
NOVALIDATE   NOVALIDATE: Y/N , default: N
INDEX_PARALLEL_DEGREE     Option of index parallel degree, default: 0 (NOPARALLEL)
```

다음은 명령 프롬프트에서 지정할 수 있는 tbExport 유틸리티의 파라미터이다.

항목	설명
CFGFILE	환경설정 파일의 이름이다.
COMPRESS	Export 수행과 동시에 Compress하는 기능이다. – Y : Compress 모드로 Export한다. – N : Compress 모드로 Export하지 않는다. (기본값) Compress 모드를 사용하는 경우 단일 스레드로 동작한다.
CONSISTENT	Export를 수행하는 시점을 기준으로 데이터를 Export하는 기능이다. – Y : Consistent 모드로 Export한다.

항목	설명
	– N : Consistent 모드로 Export하지 않는다. (기본값) flashback query에서 지원하지 않는 대상에 대해서 지원하지 않는다.
CONSTRAINT	Export를 수행할 때 제약조건의 Export 여부를 지정한다. – Y : 제약조건을 Export한다. (기본값) – N : 제약조건을 Export하지 않는다.
ENCRYPTION	Export할 때 생성되는 덤프 파일에 데이터를 암호화하여 저장할지 선택한다. (암호화 알고리즘: AES-128) 다음의 5가지 모드가 존재한다. – ALL : 데이터와 메타데이터 모두 암호화한다. – DATA_ONLY : 데이터만 암호화한다. – ENCRYPTED_COLUMNS_ONLY : 컬럼 암호화가 진행된 것만 특정하여 암호화한다. – METADATA_ONLY : 메타데이터만 암호화한다. – NONE : 어떠한 데이터도 암호화된 포맷으로 저장하지 않는다. (기본값)
ENCRYPTION_PASSWORD	Encryption Mode를 통해 덤프 파일에 적용한 암호에 대한 Password를 지정한다.
EXCLUDE	Export할 때 제외할 특정 user, table을 설정할 수 있다. 아래와 같은 방법으로 설정할 수 있다. – 스키마 및 schema.table 제외 `exclude=schema:"='TIBERO'/table:LIKE 'T%'"` `exclude=schema:"IN'USER1'"` – 여러 개 스키마 제외 `exclude=schema:"='TIBERO'"` `exclude=schema:"='USER1'"` – 여러 개 테이블 제외 `exclude=table:"LIKE'E%'"` `exclude=table:"IN('T2','T3')"`

항목	설명
FILE	Export를 수행할 때 생성되는 파일의 이름이다. (기본값: default.dat) 바이너리 파일의 형태로 운영체제에서 생성되며, 이름을 지정하지 않으면 기본값으로 생성된다.
FULL	전체 데이터베이스 모드로 Export를 수행할지 지정한다. – Y : 전체 데이터베이스 모드로 Export를 수행한다. – N : 사용자 또는 테이블 모드로 Export를 수행한다(둘 중 하나의 모드는 있어야 함). (기본값)
GEOM_ASBYTES	geometry 컬럼에 대해 WKB 또는 bytes로 얻어올지 여부를 결정한다. (기본값: N) – Tibero 6 이상 서버에서 Geometry 컬럼 데이터를 WKB 포맷으로 저장하기 때문에 export/import하는 경우 이 옵션을 사용하면 성능면에서 이점을 얻을 수 있다. export할 때 geom_asbytes 옵션을 'Y'로 설정하면 st_asbinary 같은 함수를 사용하지 않고 LOB 그대로 처리한다. – Tibero 5 SP1 이하 버전의 Tibero에서 export할 때 WKB 포맷으로 export하기 위해서는 geom_asbytes를 'Y'로 설정하지 않도록 주의해야 한다. geom_asbytes를 'N'로 설정하여 사용해야 하며, 이 때 내부적으로 st_asbinary()를 사용하여 WKB 포맷으로 데이터를 받아온다. 하위 버전의 Tibero 서버에서 Export 작업할 때 temp lob을 생성하여 처리해야 하는 성능상의 문제와 Import할 때에 DPL로 사용하는 경우 size가 큰 geometry 데이터에 대해서 처리할 수 없는 문제가 있을 수 있다.
GRANT	Export를 수행할 때 권한의 Export 여부를 지정한다. – Y : 권한을 Export한다. (기본값) – N : 권한을 Export하지 않는다.
INDEX	Export를 수행할 때 인덱스 정보의 Export 여부를 지정한다. – Y : 인덱스를 Export한다. (기본값) – N : 인덱스를 Export하지 않는다.
INLINE_CONSTRAINT	Export를 수행할 때 스크립트를 Inline Constraint로 출력할지 여부를 설정한다.

항목	설명
	– Y : Inline Constraint로 출력한다(Not Null에만 지원한다). – N : Out-of-line Constraint로 출력한다. (기본값)
IP	Export 대상 Tibero 서버의 IP 주소를 입력한다. (기본값: localhost)
LOG	Export된 오브젝트들의 스크립트가 기록될 로그 파일의 이름을 입력한다.
LOGDIR	Export의 수행 로그가 기록될 파일을 저장할 디렉터리 이름을 입력한다.
NO_PACK_DIR	압축을 해제한 덤프 파일이 저장되는 디렉터리이다. 이 옵션이 지정되면, FILE 파라미터에 설정된 값은 무시된다. **이 옵션으로 지정한 디렉터리에 파일이 존재하는 경우, 파일이 삭제될 수 있으므로 디렉터리를 지정할 때는 새로운 디렉터리나 비어있는 디렉터리 사용을 권장한다.**
OVERWRITE	Export를 수행할 때 생성되는 파일의 이름과 동일한 이름의 파일이 이미 존재하는 경우 파일을 덮어쓸지 지정한다. – Y : 파일을 덮어쓴다. – N : 파일을 덮어쓰지 않는다. (기본값) **기존 파일이 덮어씌워지므로 설정에 주의한다.**
PACK_TYPE	Export를 Compress(압축)로 수행하는 경우 packing algorithm 선택 옵션이다. – TAR : TAR 파일로 압축한다. (기본값) – ZIP : ZIP 파일로 압축한다.
PASSWORD	Export를 수행하는 사용자의 패스워드를 입력한다.
PACK_TYPE (hidden)	패키징에 사용할 알고리즘을 지정한다. – TAR : TAR 형식으로 패키징한다. (기본값) – ZIP : ZIP 형식으로 패키징한다.
PARALLEL_DEGREE (hidden)	테이블의 데이터를 Export하기 위해 수행하는 질의의 parallel hint를 입력한다. (기본값: 0 (NOT PARALLEL))
PORT	Export 대상 Tibero 서버의 포트 번호를 입력한다. (기본값: 8629)
QUERY	Export될 데이터에 필터 조건을 지정한다.

항목	설명
	– 모드에 상관없이 동작하지만, 원하지 않는 테이블에도 적용될 수 있으므로 주의한다. – Where 조건 앞과 뒤를 "\로 감싸주어야 한다. 단, 조건절의 내용을 String 처리를 해야한다면 "\"로 감싸주어야 한다. – 지정된 조건에 의해 SQL 문장에서 문법(Syntax) 에러가 발생할 경우 조건을 적용하지 않고 다시 시도한다.
REMAP_TABLESPACE	테이블 스페이스 이름을 변경할 수 있는 기능을 제공한다. 아래와 같은 방법으로 설정할 수 있다. – 테이블 스페이스 USR1에서 USR3으로 변경 `REMAP_TABLESPACE=usr1:usr3` – 여러 개 테이블 스페이스 변경 설정 `REMAP_TABLESPACE=usr1:usr3,usr2:usr4` – 대소문자 구분하여 설정 `REMAP_TABLESPACE=\"Usr1\":usr3`
REMAP_TABLE	테이블 이름을 변경할 수 있는 기능을 제공한다. 아래와 같은 방법으로 설정할 수 있다. – 테이블 TEST1에서 TEST2로 변경 `REMAP_TABLE=test1:test2` – 여러 개 테이블 변경 설정 `REMAP_TABLE=test1:test2,test3:test4` – Partition 테이블 변경 설정 `REMAP_TABLE=test1.partition_test1:test2.partition_test2`
ROWS	Export를 수행할 때 테이블의 데이터를 Export할지 여부를 지정한다. – Y : 테이블의 데이터를 Export한다. (기본값) – N : 테이블의 데이터를 Export하지 않는다.
SAVE_CREDENTIAL	암호화한 USERNAME과 PASSWORD를 사용할 때 설정한다.

항목	설명
	SAVE_CREDENTIAL 옵션을 사용하여 EXPIMP_WALLET 암호화 파일을 생성한다. ● 사용법 `SAVE_CREDENTIAL=[EXPIMP_WALLET_FILE_NAME]` 예) `SAVE_CREDENTIAL=/tmp/.expimp` `USERNAME=username` `PASSWORD=password` 다음은 EXPIMP_WALLET 파일의 환경변수를 설정하는 예이다. `export EXPIMP_WALLET=/tmp/.expimp` 다음은 위 설정의 인식 우선순위이다. 위의 설정들에 지정되어 있지 않으면 에러를 발생한다. 1. command line에 입력한 username, password 파라미터 2. cfgfile 내의 USERNAME과 PASSWORD 파일 3. EXPIMP_WALLET 파일의 USERNAME과 PASSWORD
SCRIPT	Export를 수행할 때 스키마 객체를 생성하는 DDL 스크립트의 표시 여부를 지정한다. – Y : 스키마 객체를 생성하는 DDL 스크립트를 표시한다. – N : 스키마 객체를 생성하는 DDL 스크립트를 표시하지 않는다. (기본값)
SERVER_VER (hidden)	Export의 대상이 되는 Tibero의 버전을 지정하여, 버전에 맞는 스크립트를 생성하도록 한다. Export 버전 상수이다. – 8 : Tibero 6, Tibero 7 (기본값) – 7 : Tibero 5 SP1 – 6 : Tibero 5
SID	Export 대상 Tibero 서버의 SID를 입력한다.

항목	설명
STATISTICS	Export 대상의 통계정보를 Export할지 여부를 지정한다. SAFE, RECALCULATE 등 통계정보 재계산은 지원하지 않는다. – Y : 대상 통계정보를 Export한다. – N : 대상 통계정보를 Export하지 않는다. (기본값)
TABLE	테이블 모드로 Export를 수행할 때 Export할 대상 테이블의 이름을 지정한다. 자세한 내용은 "테이블 모드"를 참고한다. ● 사용법 `TABLE=tablelist`
TEMP_DIR	Export를 수행할 때 사용되는 임시 덤프 파일들이 생성될 디렉터리를 지정한다.
THREAD_CNT	테이블의 데이터를 Export하기 위해 사용하는 스레드의 개수를 입력한다. (기본값: 4)
USER	사용자 모드로 Export를 수행할 때 Export될 객체의 소유자를 지정한다. 자세한 내용은 "사용자 모드"를 참고한다. ● 사용법 `USER=userlist`
USERNAME	Export를 수행하는 사용자의 계정을 입력한다.
NOVALIDATE	Constraint를 Export할 때 기존의 Row들이 Constraint 조건을 만족하는지 조사 여부를 지정한다. – Y : Constraint 조건을 만족하는지 조사하지 않는다. – N : Constraint 조건을 만족하는지 조사한다. (기본값) Export할 때 이 옵션을 사용한 경우에만 효과가 있으며, Import할 때에는 이 옵션의 사용 여부와 상관없이 Export할 때 사용했던 옵션대로 효과가 나타난다.
INDEX_PARALLEL_DEGREE	Export할 Index의 parallel degree를 입력한다. (기본값:0 (NOPARALLEL))

파라미터 값은 순서를 지정해서 입력하지 않아도 된다. 파라미터 값 중에 CFGFILE은 명령 프롬프트에서만 지정할 수 있지만, 나머지 파라미터 값은 환경설정 파일에서도 지정할 수 있다. 단, 환경설정 파일에 파라미터를 지정해서 사용하는 경우에는 파라미터 이름을 반드시 대문자로 입력해야 한다.

명령 프롬프트에 사용되는 파라미터를 환경설정 파일에 저장하여 관리하는 방법에는 다음과 같은 두 가지 형식이 있다. 두 번째 형식은 하나 이상의 파라미터 값을 지정하는 경우이다.

```
PARAMETER=value
PARAMETER=value1, ...
```

다음은 파라미터를 지정하는 예이다.

```
FULL=Y
FILE=EXPORT.DAT
GRANT=Y
INDEX=Y
CONSTRAINT=Y
```

2.4. 수행 예제

다음은 tbExport 유틸리티를 이용하여 Export를 수행하는 예이다.

[예 2.2] tbExport 유틸리티를 이용한 Export의 실행

```
tbExport 7.0 97668 TmaxData Corporation Copyright (c) 2008-. All rights reserved.
the entire database: Fri Feb 06 10:45:16 KST 2015
Export character set: MS949
    exporting tablespaces
    exporting roles
    exporting schema: "TIBERO"
        exporting tables
          [0] exporting table BONUS     no rows exported.
          [1] exporting table DEPT      4 rows exported.
          [2] exporting table EMP       10 rows exported.
          [3] exporting table SALGRADE  5 rows exported.
        exporting object privileges
        exporting indexes
        exporting sequences
        exporting views
        exporting synonyms
Packing the file...
Export completed successfully: Fri Feb 06 10:46:17 KST 2015
```

제3장 tbImport

본 장에서는 tbImport 유틸리티를 소개하고 사용 방법을 설명한다.

3.1. 개요

tbImport는 Tibero에서 제공하는 Import 유틸리티이다. 이 유틸리티를 통해 외부 파일에 저장된 스키마 객체를 Tibero 데이터베이스에 다시 저장하므로, tbExport 유틸리티와 함께 데이터베이스의 백업과 다른 머신 간의 데이터베이스 전송 등을 할 때 유용하다. tbImport 유틸리티는 기능에서 tbExport 유틸리티와 대칭적이거나 유사한 것이 많다.

하나의 스키마 객체를 저장하면 그와 연관된 스키마 객체가 자동으로 함께 저장된다. 필요에 따라서 연관된 일부 스키마 객체가 저장되지 않도록 지정할 수 있다.

Import 모드에는 tbExport에서와 같이 전체 데이터베이스 모드, 사용자 모드, 테이블 모드가 있다. DBA만이 사용할 수 있으며, DBA 권한을 줄 수 없을 경우 SELECT ANY DICTIONARY 권한을 부여하여 사용하는 것을 권장한다.

Import가 실행되는 과정에서 발생하는 로그는 LOG 파라미터를 사용하여 지정한다.

다음은 tbImport 유틸리티를 실행한 결과로 생성되는 완료, 경고 및 에러 메시지에 대한 설명이다.

항목	설명
완료 메시지	Import가 성공적으로 완료된 후에 출력된다.
경고 메시지	Import가 완료되었으나 경고가 발생한 경우에 출력된다.
에러 메시지	Import 과정에서 에러가 발생하여 Import를 계속할 수 없는 경우에 출력된다.

3.2. 빠른 시작

tbImport 유틸리티는 Tibero를 설치하는 과정에서 함께 설치되며, Tibero를 제거하면 함께 제거된다. 또한 Java 언어로 구현되어 있으며, JVM(Java Virtual Machine)이 설치되어 있는 어떤 플랫폼에서도 바로 운영할 수 있다.

3.2.1. 실행 전 준비사항

tbImport 유틸리티를 실행하기 전에 다음과 같은 사항을 준비해야 한다.

- JRE 1.4.2 이상 설치

- Tibero 데이터베이스 서버와 같은 플랫폼에 설치되어 있거나 네트워크로 연결된다.
- 실행에 필요한 클래스 라이브러리(기본 위치: $TB_HOME/client/lib/jar 디렉터리)
  - tbImport 클래스 : expimp.jar
  - 유틸리티 공통 라이브러리 : toolcom.jar
  - 유틸리티 공통 Logger 라이브러리 : msllogger-14.jar
  - JDBC 드라이버 : internal-jdbc-14.jar

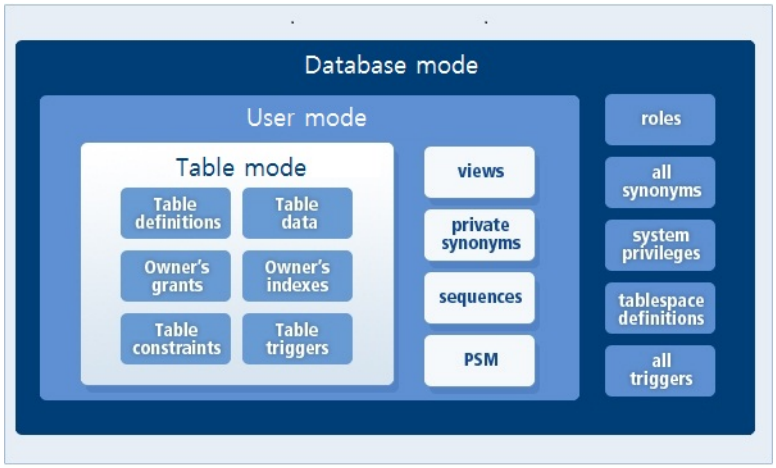
실행에 필요한 클래스 라이브러리는 Tibero를 설치하는 과정에 함께 설치되므로 추가적인 작업을 할 필요는 없다.

3.2.2. Import 모드

Import 모드는 Export 모드와 동일하게 전체 데이터베이스 모드, 사용자 모드, 테이블 모드가 있다. Export를 한 파일을 소스로 하여 각 모드의 특성에 맞게 데이터를 Import한다. 각 모드는 파라미터를 사용하여 지정할 수 있다.

다음은 모드별로 Import하는 스키마 객체의 포함 관계를 나타내는 그림이다.

[그림 3.1] Import 모드



전체 데이터베이스 모드

전체 데이터베이스 모드는 Export한 파일로부터 Tibero 데이터베이스 전체를 Import하기 위한 모드이다. SYS 사용자를 제외한 모든 사용자의 객체를 Import하기 위해 사용한다.

전체 데이터베이스 모드를 사용하기 위해서는 FULL 파라미터를 Y로 설정한다.

```
FULL=Y
```

사용자 모드

사용자 모드는 Export한 파일로부터 SYS 사용자를 제외한 지정된 사용자에 대해 그 사용자가 소유하고 있는 모든 스키마 객체를 Import하는 모드이다. DBA는 하나 이상의 사용자에게 이 모드를 사용할 수 있다.

사용자 모드를 사용하기 위해서는 USER 파라미터를 USER=userlist 형태로 설정한다.

```
USER=SCOTT, USER1, …
```

테이블 모드

테이블 모드는 Export한 파일로부터 하나 이상의 테이블을 지정하여 그 테이블과 연관된 인덱스 등의 스키마 객체를 함께 Import하는 모드이다.

테이블 모드를 사용하기 위해서는 TABLE 파라미터를 TABLE=tablelist 형태로 설정한다. 주의할 점은 SCOTT.EMP와 같이 테이블을 소유한 사용자를 반드시 명시해야 한다는 것이다.

```
TABLE=SCOTT.EMP, USER1.TABLE1, …
```

From User To User 모드

From User To User 모드는 Export한 파일로부터 FROMUSER 파라미터에 지정한 사용자에 대해서 TOUSER 파라미터에 지정한 사용자로 해당 스키마 객체의 소유자를 변경하여 Import하는 모드이다. DBA는 하나 이상의 사용자에게 이 모드를 사용할 수 있다.

From User To User 모드를 사용하기 위해서는 다음과 같은 형식으로 파라미터를 설정한다.

```
FROMUSER=SCOTT,USER1 TOUSER=USER2,USER3…
```

3.2.3. 실행

tbImport 유틸리티를 실행하려면 **$TB_HOME/client/bin** 디렉터리에서 **tbimport** 명령어를 입력한다.

다음은 전체 데이터베이스모드로 실행하는 예이다.

[예 3.1] tbImport 유틸리티의 실행

```
$ tbimport username=tibero password=tmax sid=tibero file=export.dat full=y
$ tbimport cfgfile=import.cfg
```

3.3. 수행 방법

본 절에서는 각 상황별 Import 수행 방법에 대해서 설명한다.

3.3.1. 제약조건이 있는 테이블의 Import

tbImport 유틸리티를 실행하는 중에 테이블에 제약조건이 설정되어 있는 경우 이 제약조건을 위반하는 로우는 저장되지 않는다.

만약 tbImport 유틸리티의 실행 순서로 테이블 데이터보다 테이블 제약조건을 먼저 저장한다면, 제약조건을 위반하여 저장되지 않는 로우가 생길 것이다. 예를 들어 참조 무결성 제약조건이 설정되어 있는 두 테이블에서 자식 테이블의 데이터를 부모 테이블보다 먼저 저장한다면, 자식 테이블에는 한 개의 로우도 저장되지 않을 것이다.

따라서 제약조건이 있는 테이블을 Import할 때에는 모든 테이블 데이터를 입력하고 나중에 테이블 제약조건을 설정한다.

3.3.2. 호환이 가능한 테이블의 Import

tbImport 유틸리티를 실행하기 전에 데이터베이스에 사용자가 직접 같은 이름의 테이블을 정의할 수 있다. 이때 새로 정의하는 테이블은 tbImport 유틸리티를 실행할 테이블과 호환성을 유지하는 한도에서 다르게 정의할 수 있다.

테이블 간의 호환성을 유지하려면 다음의 사항에 주의한다.

- tbImport 유틸리티를 실행할 테이블

  해당 테이블이 포함하는 컬럼을 모두 포함해야 한다.

- 데이터 타입

  상호 호환성이 있어야 하며 기본값 등을 변경하면 안 된다.

- 새로운 컬럼을 추가하는 경우

  NOT NULL 또는 기본 키 제약조건 등을 설정하지 않아야 한다.

3.3.3. 이미 존재하는 테이블에 데이터 Import

데이터베이스 내에 이미 존재하는 같은 이름의 테이블에 tbImport 유틸리티를 실행할 수도 있다. 이 때에도 앞에서 설명한 바와 같이 두 테이블 간에 호환성을 유지해야 한다.

tbImport 유틸리티를 실행하기 전에 사용자가 같은 이름의 테이블을 정의하거나 데이터베이스 내에 같은 이름의 테이블이 이미 존재하는 경우 테이블에 미리 설정되어 있는 제약조건에 의해 tbImport 유틸리티에 저장되지 않는 로우가 생길 수 있다.

이미 존재하는 테이블에 데이터를 Import할 때에는 다음의 두 가지 방법을 사용할 수 있다.

- 제약조건의 정지

  tbImport 유틸리티를 실행하는 동안에 제약조건을 잠시 정지시킨다.

- 실행 순서 조절

  tbImport 유틸리티를 실행하는 동안에 실행 순서를 조절한다. 예를 들어 참조 무결성 제약조건이 설정되어 있는 두 테이블에 대해 부모 테이블을 먼저 저장하고 나중에 자식 테이블을 저장한다.

테이블의 크기가 매우 큰 경우에는 순서 조절을 이용하는 후자의 해결 방법이 성능 면에서 유리하다. 제약조건을 잠시 정지시키는 방법의 경우 제약조건을 다시 허용하면 테이블 내의 모든 로우에 대해 제약조건의 성립 여부를 검사하기 때문에 성능 면에서 좋지 않은 영향을 줄 수 있다.

3.4. tbImport 유틸리티

본 절에서는 명령 프롬프트에서 지정할 수 있는 tbImport 유틸리티의 파라미터를 설명한다.

사용자가 파라미터 값을 지정하지 않고 tbImport 유틸리티를 실행하면, 다음과 같이 명령 프롬프트에서 지정할 수 있는 파라미터의 목록과 사용법이 나타난다.

```
tbImport 7.0 101902 TmaxData Corporation Copyright (c) 2008-. All rights reserved.
Usage: tbimport [options] [parameter=value parameter=value ...]

Options:
  -h|--help        Display the more detailed information.
  -v|--version     Display the version information.
  -p|--patch       Display the binary's patch information.

Parameters:
    BIND_BUF_SIZE     Specify the buffer size of DPL stream, default: 1M(1048576)
    CFGFILE           Config file name
    COMPRESS          Compress Mode: Y/N, default: N
    COMMIT            Commit after the insertion, default: N
    CONSTRAINT        Import Constraint: Y/N, default: Y
    DBLINK            Import DB Link: Y/N, default: Y
    DPL               Use Direct Path Load: Y/N, default: N
    ENCRYPTION_PASSWORD Specifies a password for accessing encrypted data
                        in the Dump File
    EXCLUDE_TABLE     Exclude Imported Table, default: None
    EXCLUDE_USER      Exclude Imported User, default: None
    EXP_SERVER_VER    Specify the exported server version, default: 8
    FILE              Import dump file name, default: default.dat
    FROMUSER          FromUser toUser Mode: user name list
                      (must be used with TOUSER parameter)
```

```
    FULL              Full Mode: Y/N, default: N
    GRANT             Import Grant: Y/N, default: Y
    GEOM_ASBYTES      Import the data to the geometry columns as bytes, default: N
    IGNORE            Ignore create error due to object existence: Y/N, default: N
    INDEX             Import Index: Y/N, default: Y
    IO_BUF_SIZE       Specify the buffer size of file I/O, default: 16M(16777216)
    IP                IP address, default: localhost
    LOG               Import script log file name
    LOGDIR            Import log directory
    NATIONAL_CHARSET  Specify the exported national character set,
                      default is the exported character set
    NOLOGGING         Import Table's NOLOGGING attribute: Y/N, default: N
    NO_PACK_DIR       Import unpacked dump files from specified directory.
                      If this option is specified, FILE parameter will be ignored.
    PACK_TYPE         Packing algorithm: TAR/ZIP, default: TAR
    PASSWORD          User password
    ROLE              Import Role: Y/N, default: Y
    PORT              PORT number, default: 8629
    PSM               Import PSM: Y/N, default: Y
    P_DPL             Use Parallel DPL: Y/N, default: N
    ROWS              Import Table Rows: Y/N, default: Y
    SAVE_CREDENTIAL   Save your username and password to specified file
    SCRIPT            LOG THE DDL SCRIPT: Y/N, default: N
    SEQUENCE          Import Sequence: Y/N, default: Y
    SID               Database name
    STATISTICS        Import Statistics: Y/N, default: N
    SYNONYM           Import Synonym: Y/N, default: Y
    TABLE             Table Mode: table name list
    TEMP_DIR          Directory for the temporary raw dump files.
    THREAD_CNT        Thread Count, default: 4
    TOUSER            FromUser toUser Mode: user name list
                      (must be used with FROMUSER parameter)
    TRIGGER           Import Trigger: Y/N, default: Y
    USER              User Mode: user name list
    USERNAME          Database user name
```

다음은 명령 프롬프트에서 지정할 수 있는 tbImport 유틸리티의 파라미터이다.

항목	설명
BIND_BUF_SIZE	Import를 DPL 모드로 실행할 때 stream에서 사용하는 bind buffer의 크기를 조절한다. (기본값: 1MB(1048576))
CFGFILE	환경설정 파일의 이름이다.
COMPRESS	Export할 때 Compress된 파일을 Import한다. – Y : Compress 모드로 Import한다. – N : Compress 모드로 Import하지 않는다. (기본값)

항목	설명
	Compress 모드를 사용하는 경우 단일 스레드로 동작한다.
COMMIT	insert 작업 후에 commit을 수행한다. (기본값: N) insert 작업의 단위는 아래와 같다. – CPL로 import할 때 기본적으로 bind insert buffer size인 1MB를 넘었을 때 commit을 수행한다. 만약 LONG, LONG RAW 컬럼이 있다면 row 단위로 commit을 수행한다. – DPL로 import할 때 BIND_BUF_SIZE로 지정된 크기를 넘었을 때 commit을 수행한다.
CONSTRAINT	Import를 수행할 때 제약조건의 Import 여부를 지정한다. – Y : 제약조건을 Import한다. (기본값) – N : 제약조건을 Import하지 않는다.
DPL	DPL 방법으로 Import할지 여부를 지정한다. – Y : DPL 방법을 사용한다. – N : DPL 방법을 사용하지 않는다. (기본값)
DBLINK	Import를 수행할 때 DBLink의 Import 여부를 지정한다. – Y : DBLink를 Import한다. (기본값) – N : DBLink를 Import하지 않는다.
ENCRYPTION_PASSWORD	Export할 때 사용했던 ENCRYPTION_PASSWORD를 입력하여 덤프 파일의 암호화된 데이터에 접근한다.
EXCLUDE_TABLE	Import할 때 특정 테이블을 제외하고 Import 한다. ● 사용법 `EXCLUDE_TABLE=tablelist` ● 예제 – 콤마(,)를 사용하여 다수의 테이블을 제외시킨다. 콤마를 사용하는 경우 콤마 앞뒤로 띄어쓰기가 있으면 안 된다. `EXCLUDE_TABLE=table1,table2` – username과 함께 tablename을 지정한다.

항목	설명
	``EXCLUDE_TABLE=user1.table1``` username 없이 tablename만 사용한다면 user 상관없이 동일한 tablename을 가지는 모든 테이블에 효과가 있다. – tablename과 username.tablename은 동시에 사용 가능하다. ```EXCLUDE_TABLE=table1,user1.table2``` – tablename이 중복되게 지정하는 것도 가능하다. 이 경우 타겟이 되는 모든 테이블이 exclude 대상이 된다. ```EXCLUDE_TABLE=user1.table1,user1.table1,table1```
EXCLUDE_USER	Import할 때 특정 User를 제외하고 Import 한다. ● 사용법 ```EXCLUDE_USER=userlist``` ● 예제 – 콤마(,)를 이용하여 다수의 User를 제외시킨다. 콤마를 사용하는 경우 콤마 앞뒤로 띄어쓰기가 있으면 안 된다. ```EXCLUDE_USER=user1,user2``` – username이 중복되게 지정하는 것도 가능하다. 이 경우 타겟이 되는 모든 user가 exclude 대상이 된다. ```EXCLUDE_USER=user1,user2,user1,user3```
EXP_SERVER_VER	Export한 서버의 버전을 설정한다. – 8 : Tibero 6, Tibero 7 (기본값) – 7 : Tibero 5 SP1 – 6 : Tibero 5
FILE	Import를 수행할 덤프 파일의 이름을 입력한다. (기본값: default.dat) 바이너리 파일의 형태로 운영체제에서 생성되며, 이름을 지정하지 않으면 기본값에서 Import한다.
FROMUSER	From to User 모드에서 사용하며 Export할 때 사용된 객체의 원래 소유자를 지정한다.

항목	설명
	● 사용법 다음의 형태로 사용할 수 있으며 이 때 Mapping 개수는 동일해야 한다. `FROMUSER=userlist` ● 예제 – multi user를 설정한다. `FROMUSER=user1,user2 TOUSER=user3,user4` – multi user의 object를 하나의 user로 설정할 수 있다. `FROMUSER=user1,user2 TOUSER=user3,user3` – FROMUSER에 동일한 user를 지정할 수 없다. `FROMUSER=user1,user1 TOUSER=user3,user4`
FULL	전체 데이터베이스 모드로 Import를 수행할지 여부를 지정한다. – Y : 전체 데이터베이스 모드로 Import를 수행한다. – N : 사용자 또는 테이블 모드로 Import를 수행한다(둘 중 하나의 모드는 있어야 함). (기본값)
GRANT	Import를 수행할 때 권한의 Import 여부를 지정한다. – Y : 권한을 Import한다. (기본값) – N : 권한을 Import하지 않는다.
GEOM_ASBYTES	geometry 컬럼에 대해 WKB 또는 bytes로 밀어넣을지 여부를 설정한다. (기본값: Y) – Tibero 6 이후에서 Geometry 컬럼을 WKB 포맷으로 저장하기 때문에 이 옵션을 사용할 필요가 없다. 이 옵션은 Tibero 5 SP1 이하 버전의 tibero를 export된 데이터를 import할 때 설정하여 사용해야 한다. – Tibero 5 SP1 이하 버전의 Tibero에서 export할 때, geom_asbytes를 'N'로 설정하여 Geometry 컬럼을 WKB 포맷으로 받았다면 geom_asbytes 옵션을 'N'로 설정해야한다. 내부적으로 st_geomfromwkb를 한다.

항목	설명
	만약 DPL로 넣으려면 서버 _DP_IMPORT_GEOM_FROM_OLD_FORMAT(기본값: N) iparam을 'Y'로 설정해야 한다.
IGNORE	Import를 수행할 때 이미 존재하는 스키마 객체로 인한 생성 에러를 무시 여부를 설정한다. – Y : 이미 존재하는 스키마 객체로 인한 생성 에러를 무시한다. – N : 이미 존재하는 스키마 객체로 인한 생성 에러를 무시하지 않는다. (기본값)
INDEX	Import를 수행할 때 인덱스 정보의 Import 여부를 지정한다. – Y : 인덱스를 Import한다. (기본값) – N : 인덱스를 Import하지 않는다.
IO_BUF_SIZE	Import를 실행할 때 파일의 입출력에 사용되는 버퍼의 크기를 조절한다. (기본값: 16MB(16777216))
IP	Import 대상 Tibero 서버의 IP 주소를 입력한다. (기본값: localhost)
LOG	Import된 오브젝트들의 스크립트가 기록될 파일의 이름을 입력한다.
LOGDIR	Import의 수행 로그가 기록될 파일을 저장할 디렉터리 이름을 입력한다.
NATIONAL_CHARSET	Export한 언어 셋을 설정한다. (기본값: Export한 문자 셋)
NOLOGGING	Import 대상 테이블에 대해 NOLOGGING 속성을 추가 여부를 지정한다. – Y : Import 대상 테이블에 대해 NOLOGGING 속성을 추가한다. – N : Import 대상 테이블에 대해 NOLOGGING 속성을 추가하지 않는다. (기본값)
NO_PACK_DIR	Import를 수행할 압축을 해제한 덤프 파일이 저장되는 디렉터리이다. 이 옵션이 지정되면, FILE 파라미터에 설정된 값은 무시된다.
PACK_TYPE	Compress Import를 수행하는 경우 packing algorithm 선택 옵션이다. – TAR : TAR Compress 파일을 Import한다. (기본값) – ZIP : ZIP Compress 파일을 Import한다.
PASSWORD	Import를 수행하는 사용자의 패스워드를 입력한다.
ROLE	Import를 수행할 때 ROLE의 Import 여부를 지정한다. – Y : ROLE을 Import한다. (기본값)

항목	설명
	– N : ROLE을 Import하지 않는다.
PORT	Import 대상 Tibero 서버의 포트 번호를 입력한다. (기본값: 8629)
PSM	Import를 수행할 때 PSM 오브젝트의 Import 여부를 지정한다. – Y : PSM 오브젝트를 Import한다. (기본값) – N : PSM 오브젝트를 Import하지 않는다.
P_DPL	병렬 DPL 방법으로 Import할지 여부를 지정한다. – Y : 병렬 DPL 방법을 사용한다. – N : 병렬 DPL 방법을 사용하지 않는다. (기본값)
ROWS	Import를 수행할 때 테이블의 데이터를 Import할지 여부를 지정한다. – Y : 테이블의 데이터를 Import한다. (기본값) – N : 테이블의 데이터를 Import하지 않는다.
SAVE_CREDENTIAL	암호화한 USERNAME과 PASSWORD를 사용할 때 설정한다. SAVE_CREDENTIAL 옵션을 사용하여 EXPIMP_WALLET 암호화 파일을 생성한다. ● 사용법 `SAVE_CREDENTIAL=[EXPIMP_WALLET_FILE_NAME]` 예) `SAVE_CREDENTIAL=/tmp/.expimp` `USERNAME=username` `PASSWORD=password` 다음은 EXPIMP_WALLET 파일의 환경변수를 설정하는 예이다. `export EXPIMP_WALLET=/tmp/.expimp` 다음은 위 설정의 인식 우선순위이다. 위의 설정들에 지정되어 있지 않으면 에러를 발생한다. 1. command line에 입력한 username, password 파라미터 2. cfgfile 내의 USERNAME과 PASSWORD 파일

항목	설명
	3. EXPIMP_WALLET 파일의 USERNAME과 PASSWORD
SCRIPT	Import를 수행할 때 스키마 객체를 생성하는 DDL 스크립트를 표시할지 여부를 지정한다. – Y : 스키마 객체를 생성하는 DDL 스크립트를 표시한다. – N : 스키마 객체를 생성하는 DDL 스크립트를 표시하지 않는다. (기본값)
SEQUENCE	Import를 수행할 때 Sequence의 Import 여부를 지정한다. – Y : Sequence를 Import한다. (기본값) – N : Sequence를 Import하지 않는다.
SID	Import 대상 Tibero 서버의 SID를 입력한다.
STATISTICS	Import 대상의 통계정보를 import할지 여부를 지정한다. SAFE, RECALCULATE 등 통계정보 재계산은 지원하지 않는다. – Y : 대상 통계정보를 Import한다. – N : 대상 통계정보를 Import하지 않는다. (기본값)
SYNONYM	Import를 수행할 때 Synonym의 Import 여부를 지정한다. – Y : Synonym을 Import한다. (기본값) – N : Synonym을 Import하지 않는다.
TABLE	테이블 모드로 Import를 수행할 때 Import할 대상 테이블의 이름을 지정한다. ● 사용법 `TABLE=tablelist`
TEMP_DIR	Import를 수행할 때 사용되는 임시 덤프 파일들이 생성될 디렉터리를 지정한다.
THREAD_CNT	테이블 데이터를 Import하기 위해 사용하는 스레드의 개수를 입력한다. (기본값: 4)
TOUSER	From to User 모드에서 사용하며 Import를 수행할 때 Import할 소유자를 지정한다.

항목	설명
	● 사용법 `TOUSER=userlist`
TRIGGER	Import를 수행할 때 Trigger의 Import 여부를 지정한다. – Y : Trigger를 Import한다. (기본값) – N : Trigger를 Import하지 않는다.
USER	사용자 모드로 Import를 수행할 때 Import될 객체의 소유자를 지정한다. ● 사용법 `USER=userlist`
USERNAME	Import를 수행하는 사용자의 계정을 입력한다.

파라미터 값은 순서를 지정해서 입력하지 않아도 된다. 파라미터 값 중에 CFGFILE 값은 명령 프롬프트에서만 지정할 수 있지만, 나머지 파라미터 값은 환경설정 파일에서도 지정할 수 있다. 단, 환경설정 파일에 파라미터를 지정해서 사용하는 경우에는 파라미터 이름을 반드시 대문자로 입력해야 한다.

명령 프롬프트에 사용되는 파라미터를 환경설정 파일에 저장하여 관리하는 방법에는 다음과 같은 두 가지 형식이 있다. 두 번째 형식은 하나 이상의 파라미터 값을 지정하는 경우이다.

```
PARAMETER=value
PARAMETER=value1, ...
```

다음은 파라미터를 지정하는 예이다.

```
FULL=Y
FILE=TBEXPORT.DAT
GRANT=Y
INDEX=Y
CONSTRAINT=Y
```

3.5. 수행 예제

tbImport 유틸리티로 Import를 수행하는 순서는 다음과 같다.

1. 테이블 정의

2. 테이블 데이터

3. 테이블 인덱스

4. 테이블 제약조건, 뷰, 프러시저 등

다음은 tbImport 유틸리티를 이용하여 Import를 수행하는 예이다.

[예 3.2] tbImport 유틸리티를 이용한 Import의 수행

```
tbImport 7.0 97819 TmaxData Corporation Copyright (c) 2008-. All rights reserved.
Unpacking the file...
the entire database: Mon Jul 14 01:07:43 KST 2014
Import character set: MS949
The version of this tbExport dump file is 5.0.
  importing schema: "TIBERO"
    importing tables
      [M] importing table BONUS     no rows imported.
      [0] importing table DEPT      4 rows imported.
      [0] importing table SALGRADE  5 rows imported.
      [1] importing table EMP       10 rows imported.
    importing index
    importing sequences
    importing views
    importing synonyms
Import completed successfully: Mon Jul 14 01:07:55 KST 2014
```

제4장 tbLoader

본 장에서는 tbLoader 유틸리티를 소개하고 사용 방법을 설명한다.

4.1. 개요

tbLoader는 대량의 데이터를 한번에 Tibero 데이터베이스에 저장하기 위한 유틸리티이다.이 유틸리티를 통해 SQL 문장을 일일이 작성하여 데이터베이스에 입력할 데이터를 입력하는 대신, 컬럼 데이터만 일반 텍스트 파일로 만들어서 한꺼번에 적재할 수 있다. 따라서 많은 데이터를 Tibero의 데이터베이스에 한 번에 저장할 때 유용한다.

tbLoader 유틸리티를 사용하면 사용자가 데이터 파일을 쉽게 작성할 수 있으며, 데이터를 적재하기 위한 시간을 단축할 수 있다.

4.2. 빠른 시작

tbLoader 유틸리티는 Tibero를 설치하는 과정에서 함께 설치되며, Tibero를 제거하면 함께 제거된다.

tbLoader 유틸리티는 명령 프롬프트에서 다음과 같은 형식으로 실행한다.

```
$ tbloader [options]
```

명령 프롬프트에서 지정할 수 있는 옵션에 대한 자세한 내용은 “4.9. tbLoader 유틸리티”를 참고한다.

다음은 tbLoader 유틸리티를 실행하는 예이다.

[예 4.1] tbLoader 유틸리티의 실행

```
$ tbloader userid=db_user/db_password@default
control=sample.ctl data=sample.data direct=Y
```

4.3. 입출력 파일

tbLoader 유틸리티는 컨트롤 파일(Control file)과 데이터 파일(Data file)을 입력으로 받아서 로그 파일(Log file)과 오류 파일(Bad file)을 출력한다. 컨트롤 파일과 데이터 파일은 입력 파일로써 사용자가 작성하며, 로그 파일과 오류 파일은 tbLoader 유틸리티에서 자동으로 생성되는 출력 파일이다. tbLoader 유틸리티의 입출력 파일은 모두 일반 텍스트 파일이므로 사용자가 입력 파일을 생성하기가 매우 용이하다.

본 절에서는 tbLoader 유틸리티를 사용할 때 필요한 입력 파일인 컨트롤 파일, 데이터 파일과 로드 결과로 출력되는 로그 파일, 오류 파일을 설명한다.

4.3.1. 컨트롤 파일

컨트롤 파일은 tbLoader 유틸리티의 실행을 위한 파라미터를 지정한 파일이다. 사용자는 컨트롤 파일에 읽어 올 데이터의 위치와 데이터를 읽어오는 구체적인 방법과 실제 데이터를 저장할 위치를 명시한다. 컨트롤 파일의 파라미터는 “4.8. 컨트롤 파일 옵션 지정”에서 자세히 설명한다.

4.3.2. 데이터 파일

데이터 파일은 데이터베이스의 테이블에 저장할 데이터가 들어있는 텍스트 파일이다. 데이터 파일은 tbSQL의 SPOOL 명령어를 이용하여 SQL 질의 결과를 저장하거나 일반 텍스트 편집기로 직접 작성한다.

사용자가 지정한 데이터 파일은 고정된 레코드 형태(Fixed Record Format), 분리된 레코드 형태(Separated Record Format)의 두 가지 형식으로 저장된다.

4.3.2.1. 고정된 레코드 형태

사용자가 컨트롤 파일에서 모든 컬럼에 대해서 POSITION 정보를 명시하였을 경우 적용되는 형식이다. 이 타입은 필드 사이에 구분자가 없는 대신 사용자가 명시한 컬럼의 위치에서 실제 데이터 값을 읽어 들인다. 단, 컬럼의 위치는 Byte의 길이에 의해 결정된다. 분리된 레코드 형태에 비해 유연성(flexibility)이 부족하지만 성능 면에서는 우수하다.

사용자는 레코드를 구분하기 위해서 고정된 길이의 레코드 사이즈를 명시하거나 시스템의 라인 끝(End of Line, 이하 EOL) 문자를 사용한다. 또한 시스템에서는 속도 향상을 위해 컨트롤 파일에 명시한 컬럼의 위치에서 데이터를 바로 읽어 들이므로 ESCAPED BY와 LINE STARTOR 파라미터 값도 무시한다.

따라서 “4.8. 컨트롤 파일 옵션 지정” 중에 FIELDS 모든 구문과 LINE STARTED BY 구문을 사용할 수 없고, 컬럼의 POSITION 구문을 사용해야 한다.

- LINES FIX 구문을 사용한 예제

  데이터 파일에 포함된 레코드의 길이가 고정된 길이(Byte 수)를 가질 때 사용하는 방식이다. 사용자는 컨트롤 파일에 LINES FIX 12와 같이 레코드의 사이즈를 명시해야 한다. 만약 LINES TERMINATED BY 구문이 사용되었다면 무시된다.

  다음은 고정된 길이의 레코드인 경우의 예이다.

```
example.ctl:
     LOAD DATA
     INFILE  'example.dat'
     LOGFILE 'example.log'
     BADFILE 'example.bad'
     APPEND
     INTO TABLE EMP
     LINES FIX 12
     (
         empno position (01:04),
         ename position (06:12)
```

```
    )

    example.dat:
    7922 MILLER 7777 KKS    7839 KING   7934 MILLER 7566 JONES
```

- LINES FIX 구문을 사용하지 않은 예제

  사용자가 컨트롤 파일에 라인 사이즈를 명시하지 않은 경우 EOL 문자('\n')를 사용하여 라인을 구분한다. 다음은 라인 구분자가 EOL 문자인 경우의 예이다.

```
example.ctl:
    LOAD DATA
    INFILE  'example.dat'
    LOGFILE 'example.log'
    BADFILE 'example.bad'
    APPEND
    INTO TABLE EMP
    TRAILING NULLCOLS
    (
        empno position (01:04),
        ename position (06:15),
        job   position (17:25),
        mgr   position (27:30),
        sal   position (32:39)
    )
```

```
example.dat:
    7922 MILLER     CLERK     7782 920.00
    7777 KKS        CHAIRMAN
    7839 KING       PRESIDENT      5500.0
    7934 MILLER     CLERK     7782 920.00
    7566 JONES      MANAGER   7839 3123.75
    7658 CHAN       ANALYST   7566 3450.00
    7654 MARTIN     SALESMAN  7698 1312.50
```

example.dat 데이터 파일의 두 번째 라인의 경우 마지막 두 컬럼에 해당하는 값이 없기 때문에 “4.8.19. TRAILING NULLCOLS 구문”이 없다면 오류가 발생한다.

4.3.2.2. 분리된 레코드 형태

사용자가 컨트롤 파일에서 모든 컬럼에 대해서 POSITION 정보를 명시하지 않았을 경우 적용되는 형식이다. 각각의 필드는 FIELD TERMINATOR로 구분되고, 각각의 레코드는 LINE TERMINATOR로 구분된다. 고정된 레코드 형태에 비해 성능 면에서는 부족하지만 사용자가 원하는 형태로 형식을 지정할 수 있어서 탁월한 유연성을 지녔다고 할 수 있다.

다음은 분리된 레코드 형태의 예이다.

```
example.ctl:
   LOAD DATA
   INFILE  'example.dat'
   LOGFILE 'example.log'
   BADFILE 'example.bad'
   APPEND
   INTO TABLE emp
   FIELDS TERMINATED BY ','
          OPTIONALLY ENCLOSED BY '"'
          ESCAPED BY '\\'
   LINES TERMINATED BY '\n'
   (
       empno,
       ename,
       job,
       mgr,
       hiredate,
       sal,
       comm,
       deptno
   )
```

```
example.dat:
   7654, "Martin", "Sales",7698,1981/10/28,1312.50,3,10
   7782, "\,Clark","Manager" ,7839, 1981/01/11 ,2572.50,10,20
   7839, "King",President,,1981/11/17,5500.00,,10
   7934,"Miller","Clerk",7782 ,1977/10/12,920.00,,10
   7566, "Jones",Manager" ,7839, 1981/04/02,3123.75,,20
   7658, "Chan", Ana lyst, 7566,1982/05/03,3450,,20
```

4.3.3. 로그 파일

tbLoader 유틸리티의 실행 과정을 기록한 파일이다. 사용자에게 입력할 컬럼의 기본적인 메타 데이터와 함께 실제 입력에 성공한 레코드, 실패한 레코드에 대한 통계가 제공된다. 또한 실패한 레코드에 대해서는 tbLoader 유틸리티가 판단한 실패의 이유가 제공된다.

4.3.4. 오류 파일

tbLoader 유틸리티를 실행할 때 로드에 실패한 레코드의 데이터를 기록한 파일이다. 사용자는 실패한 레코드를 포함하고 있는 오류 파일의 레코드를 수정하여 다시 로드한다.

4.4. 로드 방식

tbLoader 유틸리티에서 데이터를 로드하는 방식은 다음과 같이 두 가지가 있다.

- Conventional Path Load

  tbLoader 유틸리티에서 기본적으로 제공하는 데이터 로드 방법이다. 사용자가 지정한 데이터 파일을 읽어가며 컬럼 데이터를 읽어 컬럼 배열에 담고 일괄 처리(BATCH UPDATE) 방식으로 데이터를 데이터베이스 서버에 로드한다. Direct Path Load에 비해 성능이 약간 떨어지나, 별도의 제약 사항이 없다는 장점이 있다.

- Direct Path Load

  사용자가 지정한 데이터 파일을 읽어가며, 특정 컬럼의 데이터 타입에 맞게 데이터를 컬럼 배열(column array)형태로 만든다. 컬럼 배열 형태의 데이터는 블록 형식기(block formatter)을 거쳐서 Tibero의 데이터베이스 블록 형태에 맞게 만들어지고, 이 블록을 직접 Tibero의 데이터베이스에 쓰게 된다.

  Direct Path Load는 Conventional Path Load에 비해 훨씬 빠르다는 장점을 가지고 있으나, 다음과 같은 제약 사항이 있다.

  – CHECK 제약조건과 참조 키 제약조건(Referential Constraint)을 검사하지 않는다.

  – 기본 키 제약조건(Primary Key Constraint), 유일 키 제약조건(Unique Key Constraint), NOT NULL 제약조건은 검사한다.

  – 로딩 중에 입력 트리거(Insert trigger)가 작동하지 않는다.

4.5. 제약조건

본 절에서는 tbLoader 유틸리티의 제약조건을 설명한다.

4.5.1. 동일한 구분자의 사용

FIELD TERMINATED BY, ENCLOSED BY, ESCAPED BY, LINE TERMINATED BY 옵션 값에 대해 어느 하나라도 같은 값으로 지정된다면 데이터 파일로부터 정상적으로 읽을 수 없다.

예를 들어 다음과 같이 FIELD TERMINATED BY와 ENCLOSED BY 옵션 값을 같게 지정하면 에러가 발생한다.

```
FIELDS TERMINATED BY '"'
       OPTIONALLY ENCLOSED BY '"'
```

4.5.2. ESCAPED BY 옵션 값을 지정하지 않은 경우

만약 다음과 같이 ENCLOSED BY, FIELDS TERMINATED BY, LINES TERMINATED BY로 지정된 문자가 데이터 파일의 입력 필드 값으로 사용된 경우 ESCAPED BY를 지정하지 않으면 제대로 해석하지 못한다. 자세한 사항은 "4.8.15. FIELDS ESCAPED BY 구문"을 참고한다.

```
example.ctl:
    FIELDS TERMINATED BY ','
    (ID, NAME)

example.dat:
    7654,Martin, Kim
```

4.5.3. 테이블 owner와 수행 user가 다른 경우

load하려는 대상 테이블의 owner와 수행 user가 다른 경우 DPL 모드에서는 수행하려는 user에게 LOCK ANY TABLE 권한이 있어야 한다. 이는 DPL 방식의 빠른 속도를 보장하기 위해 작업 시작 전에 table lock을 잡아주어 다른 세션이 해당 테이블에 DML을 할 수 없게 하는 서버의 정책이다.

다음 구문으로 권한을 부여할 수 있다.

```
grant lock any table to 수행 user;
```

권한이 없을 경우에는 다음의 에러가 발생한다.

```
-17004: Permission denied
```

4.6. 공백 정책

본 절에서는 tbLoader 유틸리티에서 공백(Whitespace)을 처리하는 방법에 대해 설명한다.

4.6.1. 필드 값 전체가 공백인 경우

한 개의 레코드의 컬럼 값에 해당하는 입력 파일의 필드 값이 공백으로만 이루어져 있을 때 tbLoader 유틸리티는 입력 테이블의 컬럼의 데이터 타입에 따라 0 또는 NULL 값을 로딩한다.

4.6.2. 필드 값 일부가 공백인 경우

tbLoader 유틸리티는 빈 문자(' '), 탭 문자('\t') 및 EOL 문자('\n')를 공백으로 취급한다. 단, 해당 문자가 FIELD TERMINATED BY 구문이나 LINE TERMINATED BY 구문으로 선언되어 있는 경우는 공백으로 취급하지 않는다. 공백은 필드의 시작과 끝에 존재할 수 있다. 단, 필드의 중간에 존재하는 공백은 데이터의 한 부분으로 취급한다.

tbLoader 유틸리티는 공백에 대해서 데이터 파일의 형태에 따라서 다르게 취급한다.

고정된 레코드 형태인 경우

필드 값의 앞에 존재하는 공백은 실제 데이터로 취급하고, 뒤에 따르는 공백은 필요 없는 공백으로 간주하여 잘라낸다. 사용자가 필드 값의 앞에 존재하는 공백은 임의로 제거할 수 있지만, 뒤에 따르는 공백은 위치를 조정하기 위해 부가적으로 삽입하는 경우가 많기 때문이다.

다음과 같이 사용자가 필드 값의 앞과 뒤에 공백을 둔 경우 뒤쪽의 공백은 잘라낸다.

```
" aaa \t" -> " aaa"
```

분리된 레코드 형태인 경우

필드 값의 앞과 뒤에 존재하는 공백을 모두 필요 없는 것으로 간주하여 잘라낸다. 단, 사용자가 필드 값의 앞과 뒤에 공백을 삽입하고 싶다면, ENCLOSED BY 문자열로 감싸준 후에 공백을 삽입하면 공백은 필드의 데이터로 인식된다.

다음과 같이 사용자가 필드 값의 앞과 뒤에 공백을 둔 경우 앞과 뒤의 공백을 모두 잘라낸다.

```
" aaa \t" -> "aaa"
```

4.6.3. 필드 값의 공백을 데이터로 인식하려는 경우

tbLoader 유틸리티는 앞에서 언급한 방식으로 데이터를 잘라내지만, “4.8.9. PRESERVE BLANKS 구문”을 사용하여 필드의 앞과 뒤에 존재하는 공백을 강제로 데이터 값으로 입력할 수 있다.

4.7. 고급 기능

본 절에서는 tbLoader 유틸리티를 통해서 추가된 고급 기능에 대해 설명한다.

4.7.1. Parallel DPL

Direct Path Load 방식으로 전송하면 대상이 되는 테이블에 Lock이 걸리므로 동시에 같은 테이블에 DPL로 로딩할 수 없다. 이를 개선한 방법이 Parallel DPL이다.

Parallel DPL을 사용하면 load하는 데이터에 Parallel DPL flag가 설정되며, 서버는 병렬적으로 데이터를 받아 merge 작업을 수행한 후 저장한다. Parallel DPL을 사용하기 위해서는 tbloader 에서 direct를 'Y'로 설정해야 하고, parallel을 2 이상의 값으로 설정해주어야 한다.

다음은 Parallel DPL을 사용하는 예이다.

[예 4.2] Parallel DPL을 사용하여 tbLoader 실행 예

```
$ tbloader userid=db_user/db_password@default
control=sample.ctl data=sample.data direct=Y parallel=2
```

4.7.2. 접속 정보 암호화 기능

tbLoader 유틸리티는 데이터베이스 접속 정보(connect_string)를 암호화 파일(wallet)을 이용하여 사용하는 기능을 제공한다. 기능을 사용하기 위해서 먼저 다음 "암호화 파일 생성"을 참고하여 접속 정보를 저장하는 암호화 파일(wallet)을 생성해야 한다.

다음은 파일이 생성된 후 이를 이용하여 접속하는 예이다.

[예 4.3] 암호화 파일(wallet)을 사용하여 tbLoader 실행 예

```
$ export LR_WALLET_PATH=./wallet_file.dat

$ tbloader control=sample.ctl data=sample.data
```

참고

이 기능은 현재 UNIX 계열에서만 지원하며, Open SSL을 설치해야 사용 가능하다.

4.7.3. 메모리 보호 기능

유틸리티는 메모리 보호 기능을 제공한다. 기본 동작으로 메모리 보호 기능이 동작하고 있으며, 만약 사용을 중지하고 싶을 경우 환경변수를 추가해야 한다.

[예 4.4] 메모리 보호 기능 해제 예

```
$ export LD_MEM_PROTECT=N
```

4.7.4. 데이터 압축 전송 기능

tbLoader 유틸리티는 Direct Path Load 방식 데이터 전송에서 데이터 압축 기능을 제공한다. 데이터 압축 전송을 사용하기 위해서 tbLoader를 수행할 클라이언트에서 환경변수 TBCLI_DPL_COMPRESSION_LEVEL을 추가해야 한다. TBCLI_DPL_COMPRESSION_LEVEL의 값으로 압축 레벨을 설정한다.

다음은 압축 레벨에 대한 예제와 설명이다.

[예 4.5] 데이터 압축 전송 활성화 클라이언트

```
$ export TBCLI_DPL_COMPRESSION_LEVEL= -1
```

압축 레벨	설명
0	데이터를 압축하지 않는다. (기본값)
1	빠른 데이터 압축 전송을 한다.
9	고압축으로 데이터를 전송한다.
-1	기본 압축으로 데이터를 전송한다.

서버에서 압축 데이터를 받기 위해서 서버의 tip 파일에 _USE_DPL_MSG_COMPRESSION를 작성한다.

[예 4.6] 데이터 압축 전송 활성화 서버

```
_USE_DPL_MSG_COMPRESSION=Y (기본값 N)
```

4.8. 컨트롤 파일 옵션 지정

본 절에서는 컨트롤 파일 옵션을 지정하는 방법에 대해 설명한다.

사용자는 컨트롤 파일에서 다음의 정보를 지정할 수 있다.

- 데이터 파일에 포함된 문자 집합
- 데이터 파일의 byte ordering
- 데이터를 포함하는 데이터 파일
- 로드 수행 중에 발생한 로그를 기록할 로그 파일
- 로드에 실패한 데이터를 기록하는 오류 파일
- 오류 파일에 남기지 않을 에러 번호 지정
- 테이블에 존재하는 기존 데이터에 대한 처리 방법(APPEND|REPLACE|TRUNCATE|MERGE)
- 필드 종료자 및 기타 옵션(TERMINATOR, ENCLOSED BY STRING, ESCAPED BY STRING)
- 라인의 시작 문자열과 종료 문자열
- 데이터 파일에서 무시할 라인의 개수
- 테이블의 특정 컬럼에 대한 옵션

컨트롤 파일의 형식은 다음과 같다. 단, 컨트롤 파일에서 정한 아래 형식의 순서대로 사용해야 하며 대괄호([ ])에 포함된 내용은 생략할 수 있다.

```
LOAD DATA
  [CHARACTERSET characterset_name]
  [BYTEORDER endian_type]
  [INFILE data_file_name]
```

```
[LOGFILE log_file_name]
[BADFILE bad_file_name]
[DISCARDFILE discard_file_name]
[SKIP_ERRORS error_number, ...]
[APPEND|REPLACE|TRUNCATE|MERGE(column_name, .....)]
[PRESERVE BLANKS]
INTO TABLE table_name
[WHEN filter_expression [AND filter_expression...]]
[MULTI INSERT INDEXES|FAST BUILD INDEXES]
[FIELDS [TERMINATED BY field_terminator]
        [OPTIONALLY ENCLOSED BY enclosed_by_start_string
                          [AND enclosed_by_end_string]]
        [ESCAPED BY escaped_by_string]]
[LINES  [FIX number]
        [STARTED BY line_start_string]
        [TERMINATED BY line_terminator_string]]
[TRAILING NULLCOLS]
[IGNORE number LINES]
(column_name [FILLER]
             [POSITION(from:to)]
             [INTEGER EXTERNAL(size)              |
              FLOAT EXTERNAL(size)                |
              DOUBLE EXTERNAL(size)               |
              CHAR(size)                          |
              RAW(size)                           |
              DATE(size) date_fmt_string          |
              TIMESTAMP(size) timestamp_fmt_string |
              TIME(size) time_fmt_string]
             [OUTFILE]
             [CONSTANT constant_value]
             [NULL TERMINATED]
             [PRESERVE BLANKS]
             [sql_expression]
             [NULLIF cond_expression], ..........)
-- This line is comment.
```

다음은 컨트롤 파일의 설정 구문에 대한 설명이다. 상세한 내용은 각 절의 내용을 참고한다.

구문	설명
CHARACTERSET 구문	문자 집합을 지정한다.
BYTEORDER 구문	byte order type을 지정한다.
INFILE 구문	실제 데이터를 포함하고 있는 텍스트 파일인 데이터 파일의 경로와 이름을 지정한다.
LOGFILE 구문	데이터 로딩 과정에서 발생하는 로그를 기록할 로그 파일의 경로와 이름을 지정한다.

구문	설명
BADFILE 구문	데이터 로딩에 실패한 레코드를 기록할 오류 파일의 경로와 이름을 지정한다.
DISCARDFILE 구문	WHEN 조건절에 의해 생략된 레코드를 기록할 파일의 경로와 이름을 지정한다.
SKIP_ERRORS 구문	명시된 에러 번호에 대하여 오류 파일에 남기지 않도록 지정한다.
APPEND|REPLACE|TRUNCATE|MERGE 구문	사용자가 지정한 테이블에 기존에 데이터가 존재할 경우 그 데이터를 처리하는 방법을 지정한다.
PRESERVE BLANKS 구문	필드의 데이터에 포함된 공백을 잘라내지 않고 그대로 데이터베이스에 입력할 때 사용한다.
INTO TABLE 구문	데이터 파일의 내용을 입력하고자 하는 대상 테이블의 이름을 지정한다.
WHEN 구문	로딩할 데이터 파일의 필드 값에 대한 조건을 지정한다.
MULTI INSERT INDEXES|FAST BUILD INDEXES 구문	Direct Path Load 방식으로 대상 테이블에 데이터를 로드할 때 테이블에 존재하는 인덱스의 생성 방법을 지정한다.
FIELDS TERMINATED BY 구문	필드 종료자를 지정한다.
FIELDS OPTIONALLY ENCLOSED BY 구문	데이터 파일로부터 레코드를 읽어 올 때 필드의 값의 시작과 끝을 감쌀 문자열을 지정한다.
FIELDS ESCAPED BY 구문	지정된 문자열을 만나면 뒤따르는 문자의 의미를 확장시켜서 특수 문자 또는 문자열을 읽을 수 있다.
LINES FIX 구문	데이터 파일에서 한 라인의 길이를 지정한다.
LINES STARTED BY 구문	라인 시작 문자열을 지정한다.
LINES TERMINATED BY 구문	라인 종료 문자열을 지정한다.
TRAILING NULLCOLS 구문	데이터 파일의 레코드에 없는 마지막 컬럼의 데이터를 NULL로 취급한다.
IGNORE LINES 구문	데이터 파일의 처음부터 시작하여 지정한 수만큼의 라인을 로딩 대상에서 제외한다.
테이블 컬럼 속성	사용자가 데이터를 입력할 테이블의 컬럼 리스트를 설정한다.

4.8.1. CHARACTERSET 구문

문자 집합을 지정한다. 다양한 문자 집합의 데이터를 Tibero 서버에 로드할 수 있다. 지정된 값은 컨트롤 파일과 데이터 파일의 문자 집합에 영향을 준다. 지정하지 않을 경우 클라이언트의 문자 집합인 환경변수 TB_NLS_LANG 값이 데이터 파일의 기본 문자 집합으로 지정된다.

참고

TB_NLS_LANG의 기본값은 다음과 같다.

- Tibero 6 이전 : MSWIN949
- Tibero 7 이후 : UTF8

CHARACTERSET 구문 세부 내용은 다음과 같다.

- 문법

```
CHARACTERSET characterset_name
```

항목	설명
characterset_name	TB_NLS_LANG으로 사용할 수 있는 문자 집합들 중 하나를 지정한다. (기본값: 클라이언트의 기본 문자 집합)

- 예제

  현재 클라이언트의 문자 집합이 KSC5601(TB_NLS_LANG =EUCKR)인데 MSWIN949 데이터 파일을 서버에 로드하려는 경우 다음과 같이 컨트롤 파일에 지정한다.

```
CHARACTERSET MSWIN949
```

4.8.2. BYTEORDER 구문

BYTE ORDER을 지정한다. 바이너리 데이터를 읽을 때 필요하다.

BYTE ORDER 구문 세부 내용은 다음과 같다.

- 문법

```
BYTEORDER endian_type
```

항목	설명
endian_type	머신과 데이터 파일에 있는 byte order가 다를 경우에 데이터 파일의 byte order를 지정한다. (BIG ENDIAN or LITTLE ENDIAN)

- 예제

```
BYTEORDER BIG ENDIAN
```

4.8.3. INFILE 구문

데이터 파일을 지정한다. 실제 데이터를 포함하고 있는 텍스트 파일인 데이터 파일의 경로와 이름을 지정한다. 사용자는 절대 경로와 현재 디렉터리에 대한 상대 경로 방식을 모두 사용할 수 있다.

사용자가 명령 프롬프트와 컨트롤 파일에서 모두 경로를 지정하였다면, 명령 프롬프트에서 지정한 값이 우선시 된다.

INFILE 구문 세부 내용은 다음과 같다.

- 문법

```
INFILE data_file_name
```

항목	설명
data_file_name	데이터 파일의 경로와 이름을 지정한다.

- 예제

```
INFILE '/home/test/data.dat'
INFILE '../data.dat'
```

4.8.4. LOGFILE 구문

로그 파일을 지정한다. 데이터 로딩 과정에서 발생하는 로그를 기록할 로그 파일의 경로와 이름을 지정한다. 사용자는 절대 경로와 현재 디렉터리에 대한 상대 경로 방식을 모두 사용할 수 있다.

사용자가 명령 프롬프트와 컨트롤 파일에서 모두 경로를 지정하였다면, 명령 프롬프트에서 지정한 값이 우선시 된다.

LOGFILE 구문 세부 내용은 다음과 같다.

- 문법

```
LOGFILE log_file_name
```

항목	설명
log_file_name	로그 파일의 경로와 이름을 지정한다. (기본값: 컨트롤 파일명.log)

- 예제

```
LOGFILE '/home/test/control.log'
LOGFILE '../control.log'
```

4.8.5. BADFILE 구문

오류 파일을 지정한다. 데이터 로딩에 실패한 레코드를 기록할 오류 파일의 경로와 이름을 지정한다. 사용자는 절대 경로와 현재 디렉터리에 대한 상대 경로 방식을 모두 사용할 수 있다.

사용자가 명령 프롬프트와 컨트롤 파일에서 모두 경로를 지정하였다면, 명령 프롬프트에서 지정한 값이 우선한다.

BADFILE 구문 세부 내용은 다음과 같다.

- 문법

```
BADFILE bad_file_name
```

항목	설명
bad_file_name	오류 파일의 경로와 이름을 지정한다. (기본값: 데이터 파일명.bad)

- 예제

```
BADFILE '/home/test/data.bad'
BADFILE '../data.bad'
```

4.8.6. DISCARDFILE 구문

WHEN 조건절에 의해 생략된 레코드를 기록할 파일의 경로와 이름을 지정한다. 사용자는 절대 경로와 현재 디렉터리에 대한 상대 경로 방식을 모두 사용할 수 있다.

사용자가 명령 프롬프트와 컨트롤 파일에서 모두 경로를 지정하였다면, 명령 프롬프트에서 지정한 값이 우선시 된다.

DISCARDFILE 구문 세부 내용은 다음과 같다.

- 문법

```
DISCARDFILE discard_file_name
```

항목	설명
discard_file_name	파일의 경로와 이름을 지정한다.

- 예제

```
DISCARDFILE '/home/test/data.dsc'
DISCARDFILE '../data.dsc'
```

4.8.7. SKIP_ERRORS 구문

명시된 에러 번호에 대하여 오류 파일에 남기지 않도록 지정한다.

SKIP_ERRORS 구문 세부 내용은 다음과 같다.

- 문법

```
SKIP_ERRORS error_number, ...
```

항목	설명
error_number	오류 파일에 남기지 않을 에러 번호를 지정한다.

- 예제

```
SKIP_ERRORS -10007
SKIP_ERRORS -10007, -10005
```

4.8.8. APPEND|REPLACE|TRUNCATE|MERGE 구문

사용자가 지정한 테이블에 기존에 데이터가 존재할 경우 그 데이터를 처리하는 방법을 지정한다.

REPLACE와 TRUNCATE 옵션의 경우 테이블의 레코드가 삭제된 후에 자동 커밋되므로 tbLoader 유틸리티가 수행된 이후에 기존의 데이터를 복구할 수 없다.

기존 데이터의 처리 방법을 지정하는 세부 내용은 다음과 같다.

- 문법

```
APPEND|REPLACE|TRUNCATE|MERGE(column_name, .....)
```

항목	설명
APPEND	기존 데이터를 보존하면서 새로운 데이터를 추가한다. (기본값)
REPLACE	DELETE 명령으로 기존의 데이터를 삭제하고 새로운 데이터를 추가한다. 사용자는 DELETE에 대한 권한을 가지고 있어야 한다. DELETE에 대한 트리거가 존재하는 경우 해당 트리거가 수행된다. 단, 해당 테이블에 참조 무결성 제약조건이 있는 경우 조건을 불능화한 후에 DELETE 옵션을 사용해야 한다.
TRUNCATE	TRUNCATE 명령으로 기존의 데이터를 삭제하고 새로운 데이터를 추가한다. 단, 해당 테이블에 참조 무결성 제약조건이 있는 경우 조건을 불능화한 후에 TRUNCATE 옵션을 사용해야 한다.

항목	설명
MERGE(*column_name*, ...)	MERGE할 컬럼의 리스트를 명시한다. 유틸리티는 사용자가 지정한 컬럼의 리스트를 키 값으로 사용한다. 새로운 데이터가 기존의 데이터와 키 값이 같은 경우 기존의 데이터를 새로운 데이터 값으로 UPDATE하고, 그렇지 않으면 해당 데이터를 입력한다.

- 예제

```
control.ctl:
  LOAD DATA
  ...
  REPLACE
  ...
  (...)
```

```
control.ctl:
  LOAD DATA
  ...
  MERGE(EMPNO, EMPNM)
  ...
  (...)
```

4.8.9. PRESERVE BLANKS 구문

공백을 처리하는 방법을 설정한다. 필드의 데이터에 포함된 공백을 잘라내지 않고 그대로 데이터베이스에 입력할 때 사용한다.

PRESERVE BLANKS 구문 세부 내용은 다음과 같다.

- 문법

```
PRESERVE BLANKS
```

- 예제

```
control.ctl:
  LOAD DATA
  ...
  PRESERVE BLANKS
  ...
  (...)
```

4.8.10. INTO TABLE 구문

데이터 파일의 내용을 입력하고자 하는 대상 테이블의 이름을 지정한다.

INTO TABLE 구문 세부 내용은 다음과 같다.

- 문법

```
INTO TABLE table_name
```

항목	설명
table_name	대상 테이블의 이름을 지정한다. 이때 테이블 이름으로 PUBLIC synonym 지정이 가능하다.

- 예제

```
control.ctl:
  LOAD DATA
  ...
  INTO TABLE EMP
  ...
  (...)
```

4.8.11. WHEN 구문

로딩할 데이터 파일의 필드 값에 대한 조건을 지정한다.

WHEN 구문 세부 내용은 다음과 같다.

- 문법

```
WHEN filter_expression [AND filter_expression...]

filter_expression:
  {column_name|'('column_pos')'} operator {value_string|BLANKS}
```

항목	설명
column_name	비교할 컬럼 이름을 지정한다.
column_pos	비교할 컬럼 번호를 지정한다. 1부터 시작하며, 괄호로 감싸야 한다.
operator	비교 연산자를 지정한다. 등호(=)나 부등호(!=, <>)를 사용할 수 있다.
value_string	비교할 값을 지정한다. 작은따옴표 문자열 또는 BLANKS 키워드를 사용할 수 있다.

- 예제

```
control.ctl:
  LOAD DATA
  ...
  INTO TABLE EMP
  WHEN C1 = '1'
  ...
  (...)
```

4.8.12. MULTI INSERT INDEXES|FAST BUILD INDEXES 구문

Direct Path Load 방식으로 대상 테이블에 데이터를 로드할 때 테이블에 존재하는 인덱스의 생성 방법을 지정한다. MULTI INSERT 방식과 FAST BUILD 방식 중에 하나를 선택할 수 있다.

- **MULTI INSERT INDEXES**

  인덱스를 여러 개의 레코드 단위로 최적화하여 한번에 생성하는 방식이다.

- **FAST BUILD INDEXES**

  기존의 인덱스를 무시하고 데이터 파일의 데이터를 모두 로드하여 다시 생성하는 방식이다.

tbLoader 유틸리티를 사용하기 전에 대상 테이블에 기존 데이터가 많으면 MULTI INSERT 방식으로 인덱스를 생성하는 것이 유리하고, 그 외에는 FAST BUILD 방식이 유리하다. 단, tbLoader 유틸리티를 사용하여 DPL할 때 데이터 중복 등의 문제로 인해서 인덱스가 Unusable 상태로 변경될 수 있다.

인덱스 생성 방법을 지정하는 세부 내용은 다음과 같다.

- 문법

```
[MULTI INSERT INDEXES|FAST BUILD INDEXES]
```

항목	설명
MULTI INSERT INDEXES	MULTI INSERT 방식으로 인덱스를 생성한다.
FAST BUILD INDEXES	FAST BUILD 방식으로 인덱스를 생성한다.

- 예제

```
control.ctl:
  LOAD DATA
  ...
  MULTI INSERT INDEXES
  ...
  (...)
```

4.8.13. FIELDS TERMINATED BY 구문

필드 종료자를 지정한다. 데이터 파일로부터 레코드를 읽어 올 때 FIELD TERMINATED BY로 지정된 문자를 발견할 때까지 읽어서 하나의 필드로 취급한다.

FIELDS TERMINATED BY 구문 세부 내용은 다음과 같다.

- 문법

```
FIELDS TERMINATED BY field_terminator
```

항목	설명
field_terminator	ASCII 문자열로 지정한다.

- 예제

```
control.ctl:
  LOAD DATA
  ...
  FIELDS TERMINATED BY ','
  ...
  (...)
```

4.8.14. FIELDS OPTIONALLY ENCLOSED BY 구문

데이터 파일로부터 레코드를 읽어 올 때 필드의 값의 시작과 끝을 감쌀 문자열을 지정한다.

필드의 값에 빈 문자(' ', '\t', '\r', '\n')나 필드 종료자, 라인 종료자와 같은 메타 문자열을 포함하고 있는 경우 tbLoader 유틸리티는 해당 문자열을 데이터 값으로 인식한다. 단, ESCAPED BY 문자열은 ENCLOSED BY 문자열에 의해서 감싸져 있어도, 메타 문자열로 인식한다.

주의해야 할 점은 필드의 값이 ENCLOSED BY로 지정된 문자열과 같은 문자열이 포함된 경우 ESCAPED BY로 지정된 문자열이 접두사로 사용되어야 필드의 값으로 해석된다. 만약 ESCAPED BY로 지정된 문자열이 접두사로 사용되지 않았을 경우 해당 문자열은 ENCLOSED BY 문자로 해석되어 필드 값의 나머지 부분을 인식하지 못한다.

ENCLOSED BY 문자열을 지정하는 세부 내용은 다음과 같다.

- 문법

```
FIELDS OPTIONALLY ENCLOSED BY enclosed_by_start_string
```

항목	설명
enclosed_by_start_string	ASCII 문자열로 지정한다.

- 예제

  필드를 감쌀 시작 문자열과 종료 문자열이 같을 경우 다음과 같은 형식으로 지정한다.

```
control.ctl:
  LOAD DATA
  ...
  FIELDS OPTIONALLY ENCLOSED BY '"'
  ...
  (...)
```

  필드를 감쌀 시작 문자열과 종료 문자열이 다를 경우 다음과 같은 형식으로 지정한다.

```
control.ctl:
  LOAD DATA
  ...
  FIELDS OPTIONALLY ENCLOSED BY '{$' AND '$}'
  ...
  (...)
```

  마지막으로, 아래와 같이 파라미터 값이 지정된 경우 다음과 같은 형식으로 지정한다.

```
control.ctl:
  LOAD DATA
  ...
  FIELDS OPTIONALLY ENCLOSED BY '"'
         ESCAPED BY '\\'
  ...
  (...)
```

  필드의 값으로 ENCLOSED BY로 지정된 문자열과 같은 문자열이 포함되는 경우 ESCAPED BY로 지정된 문자열이 접두사로 사용되지 않으면 필드의 값을 정확하게 인식하지 못한다.

```
”quotation mark[\”]”,0001→(quotation mark["],0001)
”quotation mark[”]”,0001→오류 발생
```

4.8.15. FIELDS ESCAPED BY 구문

ESCAPED BY로 지정된 문자열을 만나면 뒤따르는 문자의 의미를 확장시켜서 특수문자 또는 문자열을 읽을 수 있다. 주의해야 할 점은 '\'를 지정하기 위해서는 2개의 '\\'를 사용해야 한다.

ESCAPED BY 문자열을 지정하는 세부 내용은 다음과 같다.

- 문법

```
FIELDS ESCAPED BY escaped_by_string
```

항목	설명
escaped_by_string	ASCII 문자열로 지정한다.

- 예제

```
FIELDS ESCAPED BY '\\'
FIELDS ESCAPED BY '$$!'
```

4.8.16. LINES FIX 구문

데이터 파일에서 한 라인의 길이를 지정한다. FIELDS 구문과 LINE TERMINATED BY, LINE STARTED BY 구문을 사용할 수 없다. COLUMN을 구분할 수 있도록 POSITION 구문을 함께 사용해야한다. POSITION 지정을 하지 않으면 NULL이 입력된다.

LINES FIX 구문 세부 내용은 다음과 같다.

- 문법

```
LINES FIX number
```

파라미터	설명
number	정수 값으로 지정한다. (단위: Byte)

- 예제

```
control.ctl:
  LOAD DATA
  ...
  LINES FIX 5
  ...
  (...)
```

위와 같이 파라미터를 지정하고, 아래의 내용이 저장된 데이터 파일을 읽는다고 가정해보자.

```
data.dat:
abcdefghijklmnopqrst
```

tbLoader 유틸리티는 위의 데이터 파일을 읽어서 다음과 같이 해석한다.

```
LINE #1: abcde
LINE #2: fghij
```

```
LINE #3: klmno
LINE #4: pqrst
```

4.8.17. LINES STARTED BY 구문

라인 시작 문자열을 지정한다. 데이터 파일로부터 라인 단위로 읽으면서 지정한 접두사 이후에 뒤따르는 데이터에 대해서만 로드의 대상으로 취급한다. 만약 접두사를 포함하지 않는 라인을 만나게 되면, 해당 라인 전체가 로드의 대상에서 제외된다. 단, 접두사는 성능향상을 위하여 동일한 문자를 반복해서 사용한다.

LINES STARTED BY 구문 세부 내용은 다음과 같다.

- 문법

```
LINES STARTED BY line_start_string
```

파라미터	설명
line_start_string	ASCII 문자열로 지정(최대 30Byte)한다.

- 예제

```
control.ctl:
  LOAD DATA
  ...
  LINES STARTED BY '$$$'
  ...
  (...)
```

위와 같이 파라미터를 지정하고, 아래의 내용이 저장된 데이터 파일을 읽는다고 가정해보자.

```
data.dat:
$$$0001,”SMITH”
…something … $$$0002,”DAVID”
```

tbLoader 유틸리티는 위의 데이터 파일을 읽어서 다음과 같이 해석한다.

```
(0001, “SMITH”), (0002, “DAVID”)
```

4.8.18. LINES TERMINATED BY 구문

라인 종료 문자열을 지정한다. 데이터 파일로부터 데이터를 읽어 올 때 LINE TERMINATED BY 구문으로 지정된 문자열을 발견하면 하나의 레코드가 완성된 것으로 취급한다.

LINES TERMINATED BY 구문 세부 내용은 다음과 같다.

- 문법

```
LINES TERMINATED BY line_terminator_string
```

항목	설명
line_terminator_string	ASCII 문자열로 지정한다. (기본값: '\n')

- 예제

다음은 LINE TERMINATOR 문자열을 2개의 ASCII 문자로 구성된 '|\n'으로 지정하는 예이다.

```
control.ctl:
  LOAD DATA
  ...
  LINES TERMINATED BY '|\n'
  ...
  (...)
```

참고

Windows에서 작성된 데이타 파일일 경우에는 '|\r\n'과 같이 입력해야 한다.

4.8.19. TRAILING NULLCOLS 구문

데이터 파일의 레코드에 없는 마지막 컬럼의 데이터를 NULL로 취급한다. 만약 이 절을 추가하지 않으면, 레코드에 없는 NULL 컬럼의 데이터를 에러로 본다.

TRAILING NULLCOLS 구문의 세부 내용은 다음과 같다.

- 문법

```
TRAILING NULLCOLS
```

- 예제

다음은 첫 번째 레코드의 job 컬럼의 값이 NULL로 설정되는 예이다.

```
control.ctl:
  LOAD DATA
  ...
  TRAILING NULLCOLS
  ...
  (
```

```
      id,
      name,
      job
  )
```

```
data.dat:
    10 SMITH
```

4.8.20. IGNORE LINES 구문

데이터 파일의 처음부터 시작하여 지정한 수만큼의 라인을 로딩 대상에서 제외한다. 음수값을 입력할 경우 데이터 파일의 마지막에서 역으로 지정한 수만큼의 라인을 로딩 대상에서 제외한다.

로딩 대상을 지정하는 세부 내용은 다음과 같다.

- 문법

```
IGNORE number LINES
```

항목	설명
number	정수 값으로 지정한다. (기본값: 0)

- 예제

  데이터 파일의 첫 번째 라인이 컬럼의 이름을 나타낼 경우 다음과 같이 파라미터를 지정하여 첫 번째 라인만 제외한다.

```
control.ctl:
  LOAD DATA
  ...
  IGNORE 1 LINES
  ...
  (...)
```

4.8.21. UNORDERED 구문

고정된 레코드 형태의 데이터를 로딩할 때, 컨트롤 파일에 기재된 각 컬럼에 해당하는 데이터의 위치가 데이터 파일의 앞에서부터 순서대로 나열되지 않을 때 사용하는 옵션이다.

UNORDERED 구문 세부 내용은 다음과 같다.

- 문법

```
UNORDERED
```

- 예제

  다음과 같이 position 뒤에 입력되는 숫자가 오름차순이 아닐 때는 해당 옵션을 주어야 에러를 출력하지 않는다.

```
control.ctl: LOAD DATA ...
          UNORDERED ... ( c1 position (03:04), c2 position (01:02)
          )
```

4.8.22. 테이블 컬럼 속성

사용자가 데이터를 입력할 테이블의 컬럼 리스트를 설정한다. 단, 데이터 파일의 컬럼 순서와 동일하게 작성해야 한다. 파라미터 column_name에 대상 테이블의 컬럼 이름을 명시한다.

대상 컬럼과 속성을 지정하는 세부 내용은 다음과 같다. 컬럼의 속성은 각 절에서 설명한다.

```
(column_name [FILLER]
             [POSITION(from:to)]
             [INTEGER EXTERNAL(size)               |
              FLOAT EXTERNAL(size)                 |
              DOUBLE EXTERNAL(size)                |
              CHAR(size)                           |
              RAW(size)                            |
              DATE(size) date_fmt_string           |
              TIMESTAMP(size) timestamp_fmt_string |
              TIME(size) time_fmt_string]
             [OUTFILE]
             [CONSTANT constant_value]
             [NULL TERMINATED]
             [PRESERVE BLANKS]
             [sql_expression]
             [NULLIF cond_expression], ..........)
```

속성	설명
FILLER 구문	데이터 파일에서 제외시킬 데이터에 해당하는 컬럼을 지정한다.
POSITION 구문	데이터 파일의 한 라인에서 해당 컬럼에 해당하는 값의 시작 위치와 마지막 위치를 지정한다.
데이터 타입	특정한 데이터 타입과 버퍼 크기를 설정한다.
OUTFILE 구문	데이터를 별도에 파일에서 읽어 올 수 있도록 OUTFILE 속성을 지정한다.
CONSTANT 구문	특정 컬럼의 값을 데이터 파일의 값에 상관없이 상수 값으로 지정한다.
컬럼의 공백 보존	컬럼의 공백을 처리하는 방법을 설정한다.

속성	설명
SQL 표현	특정 컬럼의 값을 표현하기 위해서 SQL에서 지원하는 표현을 지정한다.
NULLIF 구문	특정 컬럼의 값을 NULL로 치환하기 위해 조건식을 지정한다.

4.8.22.1. FILLER 구문

데이터 파일에서 제외시킬 데이터에 해당하는 컬럼을 지정한다.

컬럼의 FILLER 구문의 세부 내용은 다음과 같다.

- 문법

```
FILLER
```

- 예제

  고정된 레코드 형태인 경우 기존 문법 앞에 FILLER를 명시한다.

```
control.ctl:
  LOAD DATA
  ...
  (
      empno          position (01:04),
      ename          position (06:15),
      job     filler position (17:25),
      mgr            position (27:30),
      sal            position (32:39),
      comm           position (41:48),
      deptno         position (50:51)
  )
```

  분리된 레코드 형태인 경우 컬럼 이름 뒤에 FILLER를 명시한다.

```
control.ctl:
  LOAD DATA
  ...
  (
      empno,
      ename,
      job filler,
      mgr,
      sal,
      comm,
      deptno
  )
```

4.8.22.2. POSITION 구문

데이터 파일의 한 라인에서 해당 컬럼에 해당하는 값의 시작 위치와 마지막 위치를 지정한다.

POSITION 구문에서 지정하는 세부 내용은 다음과 같다.

- 문법

```
POSITION(from:to)
```

항목	설명
from	데이터 파일의 한 라인에서 해당 컬럼의 시작 위치를 지정한다. 라인의 위치는 1부터 시작한다.
to	데이터 파일의 한 라인에서 해당 컬럼의 마지막 위치를 지정한다.

- 예제

고정된 레코드 형태인 경우 다음과 같이 컬럼의 목록과 함께 정확한 위치를 명시해야 한다.

```
control.ctl:
  LOAD DATA
  ...
  (
      empno   position (01:04),
      ename   position (06:15),
      job     position (17:25),
      mgr     position (27:30),
      sal     position (32:39),
      comm    position (41:48),
      deptno  position (50:51)
  )
```

분리된 레코드 형태인 경우 POSITION 구문을 반드시 사용할 필요는 없다.

```
control.ctl:
  LOAD DATA
  ...
  (
      empno,
      ename,
      job,
      mgr,
      sal,
      comm,
      deptno
  )
```

4.8.22.3. 데이터 타입 구문

tbLoader 유틸리티는 특정한 데이터 타입을 제공한다. 데이터 타입별로 기본값이 다르거나 컬럼의 데이터를 다른 방식으로 바인딩한다.

제공하는 데이터 타입은 다음과 같다.

- 숫자 데이터 타입
- 문자열 데이터 타입
- 바이너리 데이터 타입
- 날짜 데이터 타입

숫자 데이터 타입

문자열 데이터를 Tibero 서버의 숫자형 컬럼에 로드하기 위해 사용한다. 컬럼에 NULL을 명시하면, tbLoader 유틸리티는 기본값으로 0을 바인딩하여 서버에 로드한다. 이와는 반대로 0이 아닌 NULL로 바인딩하려면, 문자열 데이터 타입을 선언하여 사용하면 된다.

숫자 데이터 타입(NUMERIC EXTERNAL)의 컬럼의 형식을 지정하는 세부 내용은 다음과 같다.

- 문법

```
INTEGER EXTERNAL(size)|FLOAT EXTERNAL(size)|DOUBLE EXTERNAL(size)
```

항목	설명
INTEGER EXTERNAL(*size*)	문자열 데이터가 정수 형태일 때 사용한다.
FLOAT EXTERNAL(*size*)	문자열 데이터가 실수 형태일 때 사용한다.
DOUBLE EXTERNAL(*size*)	문자열 데이터가 2배 정밀도(double precision) 실수 형태일 때 사용한다.

참고

size는 "데이터 버퍼 크기"를 참고해서 지정한다.

문자열 데이터 타입

문자열 데이터를 Tibero 서버의 문자형(CHAR, VARCHAR, CLOB, NCLOB) 컬럼에 로드하기 위해 사용한다. 컬럼에 NULL을 명시하면, tbLoader 유틸리티는 기본값으로 NULL을 바인딩하여 서버에 로드한다. 이와는 반대로 NULL이 아닌 0으로 바인딩하려면, 숫자형 데이터 타입을 선언하여 사용하면 된다.

문자열 데이터 타입의 컬럼 형식을 지정하는 세부 내용은 다음과 같다.

- 문법

```
CHAR(size)
```

항목	설명
CHAR(*size*)	문자열 데이터에 사용한다. size는 "데이터 버퍼 크기"를 참고해서 지정한다.

바이너리 데이터 타입

바이너리 데이터를 Tibero 서버의 대용량 객체형(RAW, BLOB, LONGRAW) 컬럼에 로드하기 위해 사용한다. 컬럼에 NULL을 명시하면, tbLoader 유틸리티는 기본값으로 NULL을 바인딩하여 서버에 로드한다.

바이너리 데이터 타입의 컬럼 형식을 지정하는 세부 내용은 다음과 같다.

- 문법

```
RAW(size)
```

항목	설명
RAW(size)	바이너리 데이터에 사용한다. size는 "데이터 버퍼 크기"를 참고해서 지정한다.

날짜 데이터 타입

문자열 데이터를 Tibero 서버의 날짜형(DATE, TIME, TIMESTAMP) 컬럼에 로드하기 위해 사용한다. 컬럼에 NULL을 명시하면, tbLoader 유틸리티는 기본값으로 NULL을 바인딩하여 서버에 로드한다.

Tibero의 클라이언트는 TB_NLS_DATE_FORMAT, TB_NLS_TIMESTAMP_FORMAT으로 DATE와 TIMESTAMP 타입의 컬럼 형식을 지정한다. 하지만 사용자가 tbLoader 유틸리티의 컨트롤 파일에서 컬럼별로 DATE, TIMESTAMP, TIME 타입의 컬럼의 형식으로 직접 지정할 수도 있다. 단, 반드시 큰따옴표(" ")로 감싸야 한다.

날짜 데이터 타입의 컬럼 형식을 지정하는 세부 내용은 다음과 같다.

- 문법

```
DATE date_fmt_string|TIMESTAMP timestamp_fmt_string|TIME time_fmt_string
```

항목	설명
date_fmt_string	DATE 타입인 컬럼의 형식을 지정한다.
timestamp_fmt_string	TIMESTAMP 타입인 컬럼의 형식을 지정한다.
time_fmt_string	TIME 타입인 컬럼의 형식을 지정한다.

- 예제

다음의 예는 tbLoader 유틸리티의 컨트롤 파일이다. 본 예제에서는 data.dat 파일 내부에 있는 컬럼 hiredate에 'YYYYMMDD'에 맞는 형식의 날짜를 명시한다.

```
control.ctl:
  LOAD DATA
  .....
  INFILE 'data.dat'
  .....
  (
      empno,
      ename,
      hiredate DATE "YYYYMMDD"
  )
```

```
data.dat:
1111, JOHN, 19981112
2222, TOM, 20070802
```

데이터 버퍼 크기

tbLoader 유틸리티는 데이터 파일의 데이터를 읽어 데이터 버퍼에 저장하고, 해당 데이터를 Tibero 서버의 데이터 타입으로 변환한 후 로딩한다.

tbLoader 유틸리티는 기본적으로 분리된 레코드 형태의 데이터를 읽기 위해서 데이터베이스 내의 스키마 정보를 이용하여 적정한 크기의 버퍼를 할당한다. 반면에 고정된 레코드 형태의 데이터를 읽기 위해서 POSITION의 시작과 끝의 정보를 이용하여 버퍼를 할당한다.

만약 사용자가 데이터의 길이 정보를 정확히 안다면, 데이터 버퍼의 크기를 명시할 수 있다. 또한 BLOB, CLOB, LONG, LONG RAW와 같은 대용량 데이터를 읽기 위해 원하는 만큼의 데이터 버퍼의 크기(단위: Byte)를 명시할 수 있다.

- 예제

  컨트롤 파일의 형식이 다음과 같다면 empno, hiredate 컬럼에 각각 5bytes, 10bytes의 데이터 버퍼를 할당한다.

```
control.ctl:
  LOAD DATA
  ....
  (
      empno INTEGER EXTERNAL(5),
      hiredate DATE(10) "YYYY/MM/DD"
  )
```

4.8.22.4. OUTFILE 구문

사용자는 BLOB, CLOB, LONG, LONG RAW 타입처럼 대용량 데이터나, RAW와 같은 바이너리 데이터를 읽어 올 때 데이터 파일이 아닌 별도에 파일에서 읽어 올 수 있도록 **OUTFILE**속성을 지정할 수 있다. 단, 사용자는 데이터 파일에 해당 파일의 경로를 명시해야 한다.

사용자가 OUTFILE 속성을 지정하면, tbLoader 유틸리티는 내부적으로 파일의 데이터를 버퍼링하여 여러 번 서버로 로딩한다. 이와는 반대로 OUTFILE 속성을 지정하지 않으면, tbLoader 유틸리티는 대용량 데이터를 데이터 파일에서 직접 읽어 오고, 이를 위해 기본적으로 32KB의 데이터 버퍼를 할당한다. 따라서 이보다 큰 데이터를 데이터 파일에서 읽고 싶다면 직접 "데이터 버퍼 크기"를 지정한다.

- 예제

  컨트롤 파일의 형식이 다음과 같다면,

```
control.ctl:
  LOAD DATA
  .....
  INFILE 'data.dat'
  .....
  (
      empno,
      ename,
      resume OUTFILE
  )
```

  data.dat 내부에 resume 컬럼의 위치에 다음과 같이 파일 경로를 입력하면, 해당 파일로부터 데이터를 읽어 들인다.

```
data.dat
1111, JOHN, ./clobdata.txt
2222, TOM, /home/test/clobdata.txt
```

4.8.22.5. CONSTANT 구문

CONSTANT 속성을 이용하여 특정 컬럼의 값을 데이터 파일의 값에 상관없이 상수 값으로 지정할 수 있다. 단, 상수 값에 빈 문자열도 포함될 수 있으므로, 반드시 큰따옴표(" ")로 감싸야 한다.

- 예제

  컨트롤 파일의 형식이 다음과 같다면 data.dat 파일 내부의 데이터 값에 상관없이 컬럼 empno 값을 숫자 1234로 지정한다.

```
control.ctl:
  LOAD DATA
  .....
  INFILE 'data.dat'
  .....
  (
      empno constant "1234",
      ename
  )
```

4.8.22.6. 공백 보존

"4.8.9. PRESERVE BLANKS 구문"과 같은 기능을 하지만 컬럼의 속성으로 사용하여 해당 컬럼에만 적용되도록 할 수 있다.

- 예제

  컨트롤 파일의 형식이 다음과 같다면 컬럼 empno에 해당하는 첫 번째 필드의 모든 공백은 데이터 값으로 유지된다.

```
control.ctl:
  LOAD DATA
  ...
  (
      empno PRESERVE BLANKS,
      ename
  )
```

4.8.22.7. SQL 표현

특정 컬럼의 값을 표현하기 위해서 SQL에서 지원하는 표현을 사용할 수 있다. SQL 표현은 반드시 큰따옴표(" ")로 감싸야 한다. SQL 표현 안에서는 콜론(:) 기호와 컬럼 이름을 사용하여 컬럼의 데이터 값을 바인딩할 수 있다.

- 예제

  컨트롤 파일의 형식이 다음과 같다면 컬럼 empno에 TO_CHAR 함수를 이용하여 SYSDATE 값을 'YYYYMMDD' 포맷으로 문자열로 만들어 저장한다. 컬럼 ename에는 데이터 파일의 컬럼 empno 필드 값을 데이터 값으로 바인딩한다.

```
control.ctl:
  LOAD DATA
  ...
  (
      empno "TO_CHAR(SYSDATE, 'YYYYMMDD')",
      ename ":empno"
  )
```

4.8.22.8. NULLIF 구문

특정 컬럼의 값을 NULL로 치환하기 위해 조건식을 사용할 수 있다. 조건식의 좌항은 컬럼 이름이나 컬럼 번호를 사용할 수 있고, 우항은 작은따옴표(' ') 문자열 또는 BLANKS 키워드를 사용 할 수 있다.

NULLIF 구문의 세부 내용은 다음과 같다.

- 문법

```
NULLIF {column_name|'('column_pos')'} operator {value_string|BALNKS}
```

항목	설명
column_name	비교할 컬럼 이름을 지정한다.
column_pos	비교할 컬럼 번호를 지정한다. 1부터 시작하며, 괄호로 감싸야 한다.
operator	비교 연산자를 지정한다. 등호(=)나 부등호(!=, <>)를 사용할 수 있다.
value_string	비교할 값을 지정한다. 작은따옴표 문자열 또는 BLANKS 키워드를 사용할 수 있다.

- 예제

  컨트롤 파일의 형식이 다음과 같다면 컬럼 empno의 값이 0인 경우 ename 컬럼을 NULL로 치환한다.

```
control.ctl:
  LOAD DATA
  ...
  (
      empno CHAR,
      ename CHAR NULLIF empno = '0'
  )
```

4.8.23. 주석 삽입

컨트롤 파일의 해당 라인이 주석으로 처리된다.

- 예제

```
-- This is comment for control file.
```

4.9. tbLoader 유틸리티

본 절에서는 명령 프롬프트에서 지정할 수 있는 tbLoader 유틸리티의 파라미터를 설명한다.

tbLoader 유틸리티를 실행할 때 파라미터를 지정하지 않으면, 다음과 같이 명령 프롬프트에서 지정할 수 있는 파라미터의 목록과 사용 방법이 나타난다.

```
$ tbloader
tbLoader 7
TmaxData Corporation Copyright (c) 2008-. All rights reserved.

Usage: tbloader [options] [controls]
```

```
Options:
  -h|--help       Display the more detailed information.
  -c|--charset    Display usable character sets in this version.
  -v|--version    Display the version information.

Controls:
  userid          Username/password@dbname
  control         The name of the control file
  data            The name of the data file
  log             The name of the log file
  bad             The name of the bad file
  discard         The name of the discard file
  skip            The count of line to skip in data file       (Default: 0)
  errors          The count of error to allow                  (Default: 50)
  rows            The count of row for each commit
                  (Default: commit after the end)
  message         The count of records to process to print out (Default: 0)
  readsize        The size of buffer to read from the data file
                  (Default: 65536)
  disable_idx     Whether to disable indexes before loading    (Default: N)
  direct          Whether to use the Direct Path Loading       (Default: N)
  dpl_log         Whether to log about the Direct Path Loading (Default: N)
  parallel        The count of threads to execute Parallel Direct Path Loading
                  (Default: 1)
  bindsize        The size of buffer to read in the Direct Path Loading
                  (Default: 65536)
  syntax          Control file syntax (Default : tibero)

  dpl_parallel    Deprecated
  multithread     Deprecated

Example:
  tbloader userid=userid/passwd@dbname control=sample.ctl bindsize=1000000
```

다음은 명령 프롬프트에서 지정할 수 있는tbLoader 유틸리티의 파라미터이다.

항목	설명
userid	Tibero의 데이터베이스 사용자명과 패스워드 및 데이터베이스명을 다음의 형식으로 지정한다. `userid=userid/passwd@databasename`
control	파라미터 정보를 포함하는 컨트롤 파일의 경로와 이름을 지정하는 파라미터이다. 절대 경로와 현재 디렉터리에 대한 상대 경로 방식을 모두 사용할 수 있다.

항목	설명
data	실제 데이터를 포함하고 있는 텍스트 파일의 경로와 이름을 지정하는 파라미터이다. 절대 경로와 현재 디렉터리에 대한 상대 경로 방식을 모두 사용할 수 있다. 사용자가 명령 프롬프트와 컨트롤 파일에서 모두 경로를 지정하였다면, 명령 프롬프트에서 지정한 값을 우선시한다.
log	데이터 로딩 과정에서 발생하는 로그를 기록할 파일의 경로와 이름을 지정하는 파라미터이다. (기본값: 컨트롤 파일명.log) 절대 경로와 현재 디렉터리에 대한 상대 경로 방식을 모두 사용할 수 있다. 사용자가 명령 프롬프트와 컨트롤 파일에서 모두 경로를 지정하였다면, 명령 프롬프트에서 지정한 값을 우선시한다.
bad	데이터 로딩에 실패한 레코드를 기록할 파일에 대한 경로와 이름을 지정하는 파라미터이다. (기본값: 데이터 파일명.bad) 절대 경로와 현재 디렉터리에 대한 상대 경로 방식을 모두 사용할 수 있다. 사용자가 명령 프롬프트와 컨트롤 파일에서 모두 경로를 지정하였다면, 명령 프롬프트에서 지정한 값을 우선시한다.
discard	WHEN 조건절에 의해 생략된 레코드를 기록할 파일에 대한 경로와 이름을 지정하는 파라미터이다. 별도로 명시하지 않을 경우 생략된 레코드는 기록되지 않는다. 절대 경로와 현재 디렉터리에 대한 상대 경로 방식을 모두 사용할 수 있다. 사용자가 명령 프롬프트와 컨트롤 파일에서 모두 경로를 지정하였다면, 명령 프롬프트에서 지정한 값을 우선시한다.
skip	데이터 파일의 처음부터 지정한 수만큼의 라인을 로드의 대상에서 제외하는 파라미터이다. (기본값: 0) 컨트롤 파일 옵션 중 “4.8.20. IGNORE LINES 구문”과 같은 기능을 한다.
errors	데이터를 업로드할 때 최대로 허용할 에러의 개수를 지정하는 파라미터이다. (기본값: 50) tbLoader 유틸리티는 사용자가 지정한 에러의 개수를 넘지 않는 범위 내에서 데이터를 업로드한다. 만약, 지정한 개수를 만날 경우 데이터의 업로드를 중지한다. – ERRORS는 -1과 0부터 Integer의 최댓값(2147483647) 중의 한 값이다. – ERRORS를 0으로 지정하면 단 하나의 에러도 허용하지 않는다.

항목	설명
	– ERRORS를 양의 정수 N으로 지정하면 N+1개의 에러가 발생한 경우 데이터의 업로드를 중지한다. 즉, N개의 에러까지만 허용한다. – ERRORS를 -1로 지정하면 모든 에러를 건너띄고 에러가 발생하지 않는것만 업로드한다.
rows	사용자가 대용량 데이터를 업로드할 때 커밋을 수행할 레코드 개수를 지정하는 파라미터이다. 단, tbLoader는 성능을 고려하여 지정한 레코드의 개수를 정확히 맞추어서 서버로 데이터를 보내지는 않는다.
message	tbLoader 유틸리티가 현재 처리하고 있는 논리적인 레코드의 개수를 화면에 출력하는 파라미터이다. 별도로 명시하지 않을 경우 화면에 진행 상황을 출력하지 않는다. 단, 너무 작은 값을 명시할 경우 성능에 영향을 미칠 수 있다.
readsize	tbLoader는 데이터 파일의 내용을 버퍼링을 사용하여 읽어들인다. 이때 사용하는 읽기 전용 버퍼 크기(READ BUFFER SIZE)를 지정하는 파라미터이다. (단위: Byte, 기본값: 65536(64KB), 최댓값: 2,097,152(2MB))
disable_idx	데이터를 로딩하기 전 테이블에 정의된 모든 인덱스를 비활성화할지를 지정하는 파라미터이다. – Y : 모든 인덱스를 UNUSABLE 상태로 변경한다. – N : 인덱스 상태를 변경하지 않는다. (기본값)
direct	사용자가 데이터를 로드할 때 Conventional Path Load 또는 Direct Path Load 방법 중 하나를 지정하는 파라미터이다. – Y : Direct Path Load로 지정한다. 이 외의 값이나 빈 문자를 입력한 경우는 Conventional Path Load로 데이터를 로드한다. – N : Conventional Path Load로 데이터를 로드한다. (기본값)
dpl_log	Direct Path Load 방법으로 데이터를 로드할 때 서버의 로그 파일에 로그를 남길지를 지정하는 파라미터이다. – Y : 데이터를 업로드할 때 서버의 로그 파일에 로그를 남긴다. 장애가 발생하는 경우 복구가 가능하나 로딩할 때 성능이 저하되는 단점이 있다. – N : 데이터를 업로드할 때 서버의 로그 파일에 로그를 남기지 않는다. 장애가 발생했을 때 복구할 수 없다. (기본값)

항목	설명
parallel	Direct Path Load 방법으로 데이터를 로드할 때 사용할 스레드의 개수를 지정하는 파라미터이다. (기본값: 1) tbLoader 유틸리티는 지정한 개수만큼 데이터를 읽는 스레드와 서버로 로딩하는 스레드를 생성하여 동시에 수행한다.
bindsize	Direct Path Load 방법으로 데이터를 로드할 때 클라이언트에서 사용하는 Direct Path Stream의 크기를 지정하는 파라미터이다. Tibero 클라이언트는 데이터가 지정한 크기만큼 바인딩이 되기 전까지는 서버로 업로드가 되지 않는다. 따라서, 대용량의 데이터를 업로드할 때 효율적으로 사용할 수 있다. (단위: Byte, 기본값: 65536(64KB), 최댓값: 10,485,760(10MB))
syntax	컨트롤 파일이 작성된 구문을 지정한다. – Tibero : tbLoader유틸리티로 작성된 컨트롤 파일을 사용한다. (기본값) – Oracle : SQL*Loader 유틸리티로 작성된 컨트롤 파일을 사용한다.
dpl_parallel	더 이상 사용되지 않는다.
multithread	더 이상 사용되지 않는다.

참고

1. 사용자는 명령 프롬프트에서 파라미터나 컨트롤 파일에 필요한 메타 데이터를 입력할 수 있다. 단, userid, control, direct, message, readsize, bindsize, errors, rows 파라미터는 명령 프롬프트에서만 지정할 수 있다.

2. Options의 -c는 하위 호환성을 위하여 현재 클라이언트 버전에서 사용 가능한 character sets 정보를 보여준다.

다음은 명령 프롬프트에서 tbLoader 유틸리티의 파라미터를 설정한 예이다.

```
userid=userid/passwd@dbname

control=/home/test/control.ctl
control=../control.ctl

log=/home/test/control.log
log=../control.log

data=/home/test/data.dat
data=../data.dat
```

```
bad=/home/test/data.bad
bad=../data.bad

discard=/home/test/data.dsc
discard=../data.dsc

skip=1
direct=Y
dpl_log=Y
message=10000
readsize=65536
bindsize=65536
errors=100
rows=100
parallel=2
```

4.10. 수행 예제

본 절에서는 분리된 레코드 형태, 고정된 레코드 형태, BLOB과 CLOB 타입과 같은 대용량 데이터가 존재하는 세 가지 경우에 대해 다음과 같은 순서로 데이터를 로드하는 예제를 살펴본다.

1. 테이블을 생성한다(모든 예제에서 공통 사항이다).

2. 컨트롤 파일을 작성한다.

3. 데이터 파일을 작성한다.

4. tbLoader 유틸리티를 실행한다.

5. 로그 파일과 오류 파일을 확인한다.

다음은 공통으로 사용할 테이블을 생성하는 예이다. 본 예제에서는 데이터베이스 이름을 default라 하고, 테이블의 소유자는 loader/loader_pw라고 가정한다.

```
CREATE TABLE MEMBER
(
    ID        NUMBER(4) NOT NULL,
    NAME      VARCHAR2(10),
    JOB       VARCHAR2(9),
    BIRTHDATE DATE,
    CITY      VARCHAR2(10),
    PICTURE   BLOB,
    AGE       NUMBER(3),
    RESUME    CLOB
);
```

```
CREATE TABLE CLUB
(
    ID       NUMBER(6) NOT NULL,
    NAME     VARCHAR2(10),
    MASTERID NUMBER(4)
);
```

4.10.1. 분리된 레코드 형태

Conventional Path Load 방법을 이용하여 분리된 레코드 형태의 데이터를 Tibero 서버로 로드한다.

컨트롤 파일 작성

컨트롤 파일인 control.ctl의 내용은 다음과 같다.

```
LOAD DATA
INFILE './data.dat'
APPEND
INTO TABLE club
FIELDS TERMINATED BY ','
       OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '|\n'
IGNORE 1 LINES
(
 id integer external,
 name,
 masterid integer external
)
```

다음은 컨트롤 파일에 지정된 정보이다.

- 데이터 파일

  현재 디렉터리에 위치한 data.dat 파일이다.

- 로그 파일

  사용자가 명령 프롬프트에서 로그 파일명을 명시하지 않는다면, 기본적으로 컨트롤 파일명으로 설정한 control.log 파일을 생성한다.

- 오류 파일

  사용자가 명령 프롬프트에서 오류 파일명을 명시하지 않는다면, 기본적으로 데이터 파일명으로 설정한 data.bad 파일을 생성한다. 단, 오류 레코드가 존재하지 않을 때는 오류 파일을 생성하지 않는다.

- 대상 테이블

  사용자는 club 테이블의 id, name, masterid 컬럼의 데이터를 로드하려고 한다.

- FIELDS TERMINATED BY 문자열, FIELDS OPTIONALLY ENCLOSED BY 문자열, FIELDS ESCAPED BY 문자열, LINES TERMINATED BY 문자열

  FIELDS TERMINATED BY 문자로써 콤마(,)를 사용하고, FIELDS OPTIONALLY ENCLOSED BY 문자열로써 큰따옴표(" ")를 사용하고, LINES TERMINATED BY 문자열로 '|\n'을 사용한다. FIELDS ESCAPED BY 문자열은 지정하지 않았으므로 기본값으로 '\\'를 사용한다.

- 로딩 대상

  IGNORE LINES 구문이 있기 때문에 데이터 파일의 첫 번째 줄의 레코드는 무시한다.

데이터 파일 작성

데이터 파일인 data.dat의 내용은 다음과 같다.

```
id     name      masterid|
111111,FC-SNIFER,2345|
dkkkkkkkkkk|
111112,"DOCTOR CLUBE ZZANG",2222|
111113,"ARTLOVE",3333|
111114,FINANCE,1235|
111115,"DANCE MANIA",2456|
111116,"MUHANZILZU",2378|
111117,"INT'L",5555
```

tbLoader 유틸리티 실행

다음과 같이 tbLoader 유틸리티를 실행한다.

```
tbloader userid=loader/loader_pw@default control=./control.ctl
```

로그 파일과 오류 파일 확인

tbLoader 유틸리티를 실행한 후 생성된 로그 파일과 오류 파일을 확인한다.

tbLoader 유틸리티의 실행 과정을 기록한 로그 파일의 내용은 다음과 같다.

```
tbLoader 7
TmaxData Corporation Copyright (c) 2008-. All rights reserved.

Data File : ./data.dat
Bad File : ./data.bad

Table 'CLUB' was loaded from the data file.
```

```
Column Name                     Position DataType
------------------------------ ---------- ------------
ID                                    1 NUMERIC EXTERNAL
NAME                                  2 CHARACTER
MASTERID                              3 NUMERIC EXTERNAL

Record 2 was rejected.
  TBR-80053 : Some relatively positioned columns are not present in the record.

Record 3 was rejected.
  TBR-80025 : Column data exceeds data buffer size.

Record 6 was rejected.
  TBR-80025 : Column data exceeds data buffer size.

Table 'CLUB'
----------------
8 Rows were requested to load.
5 Rows were loaded successfully.
3 Rows were failed to load because of some errors

Elapsed time was: 00:00:00.407089
```

tbLoader 유틸리티가 로드에 실패한 오류 데이터를 기록한 오류 파일은 다음과 같다. 사용자는 오류가 발생한 데이터를 수정하여 Tibero의 데이터베이스에 다시 업로드를 한다.

```
dkkkkkkkkkk|
111112,"DOCTOR CLUBE ZZANG",2222|
111115,"DANCE MANIA",2456|
```

4.10.2. 고정된 레코드 구분자가 EOL 문자인 경우

Direct Path Load 방법을 이용하여 고정된 레코드 형태의 데이터를 Tibero 서버로 로드하려고 한다.

컨트롤 파일 작성

컨트롤 파일인 control.ctl의 내용은 다음과 같다.

```
LOAD DATA
INFILE '/home/test/data.dat'
APPEND
INTO TABLE MEMBER
(
    id        position (01:04) integer external,
```

```
    name      position (06:15),
    job       position (17:25),
    birthdate position (27:36),
    city      position (38:47),
    age       position (49:51) integer external
)
```

다음은 컨트롤 파일에 지정된 정보이다.

- 데이터 파일

  /home/test 디렉터리에 위치한 data.dat 파일이다.

- 로그 파일

  사용자가 명령 프롬프트에서 로그 파일명을 명시하지 않는다면, 기본적으로 컨트롤 파일명으로 설정한 control.log인 로그 파일을 생성한다.

- 오류 파일

  사용자가 명령 프롬프트에서 오류 파일명을 명시하지 않는다면, 기본적으로 데이터 파일명으로 설정한 data.bad인 오류 파일을 생성한다. 단, 오류 레코드가 존재하지 않을 때는 오류 파일을 생성하지 않는다.

- 대상 테이블

  사용자는 member 테이블의 id, name, job, birthdate, city, age 컬럼의 데이터를 로드하려고 한다.

- FIELDS TERMINATED BY 문자열, FIELDS OPTIONALLY ENCLOSED BY 문자열, FIELDS ESCAPED BY 문자열, LINES TERMINATED BY 문자열

  FIELDS TERMINATED BY, FIELDS OPTIONALLY ENCLOSED BY, FIELDS ESCAPED BY 문자열들은 고정된 레코드 형태이기 때문에 사용하지 않는다. LINES FIX 구문을 사용하지 않았기 때문에 LINES TERMINATED BY 문자로 EOL을 사용한다.

- 로딩 대상

  데이터 파일의 첫 번째 라인부터 데이터를 로딩한다.

데이터 파일 작성

데이터 파일인 data.dat의 내용은 다음과 같다.

tbLoader 유틸리티가 고정된 레코드 형태의 데이터를 입력받기 때문에, 사용자는 정확한 위치에 컬럼의 데이터를 입력해야 한다.

```
7777 KKS        CHAIRMAN  1975-11-18 SEOUL      33
7839 KING       PRESIDEN             DAEGU      45
7934 MILLER     CLERK     1967-01-24 BUSAN      37
7566 JONES      MANAGER
7499 ALLEN      SALESMAN  ddddddddd  KYUNG-JU
```

```
aaaa7654 MARTIN SALESMAN
7648 CHAN       ANALYST   1979-10-11 INCHON     28
```

tbLoader 유틸리티 실행

다음과 같이 tbLoader 유틸리티를 실행한다.

```
tbloader userid=loader/loader_pw@default control=./control.ctl direct=Y
```

로그 파일 및 오류 파일 확인

tbLoader 유틸리티를 실행한 후 생성된 로그 파일과 오류 파일을 확인한다.

tbLoader 유틸리티의 실행 과정을 기록한 로그 파일의 내용은 다음과 같다.

```
tbLoader 7
TmaxData Corporation Copyright (c) 2008-. All rights reserved.

Data File : ./data.dat
Bad File : ./data.bad

Table 'MEMBER' was loaded from the data file.

Column Name                     Position DataType
------------------------------ ---------- ------------
ID                                      1 NUMERIC EXTERNAL
NAME                                    2 CHARACTER
JOB                                     3 CHARACTER
BIRTHDATE                               4 DATE
CITY                                    5 CHARACTER
AGE                                     6 NUMERIC EXTERNAL

Record 4 was rejected.
  TBR-80053 : Some relatively positioned columns are not present in the record.

Record 5 was rejected.
  TBR-80053 : Some relatively positioned columns are not present in the record.

Record 6 was rejected.
  TBR-80053 : Some relatively positioned columns are not present in the record.

Table 'MEMBER'
----------------
7 Rows were requested to load.
4 Rows were loaded successfully.
3 Rows were failed to load because of some errors
```

```
Elapsed time was: 00:00:00.338089
```

위의 로그 파일의 내용을 보면, 로드되지 않은 3개의 레코드에 마지막 컬럼의 값이 존재하지 않는다는 것을 알 수 있다. “4.8.19. TRAILING NULLCOLS 구문”을 이용하면 해당 레코드의 마지막 컬럼의 값을 NULL 문자로 바인딩할 수 있다.

tbLoader 유틸리티가 로드에 실패한 오류 데이터를 기록한 오류 파일은 다음과 같다. 오류가 발생한 데이터는 수정해서 Tibero의 데이터베이스에 다시 업로드를 해야 한다.

```
7566 JONES      MANAGER
7499 ALLEN      SALESMAN  ddddddddd  KYUNG-JU
aaaa7654 MARTIN SALESMAN
```

4.10.3. 고정된 길이의 레코드인 경우

Conventional Path Load 방법을 이용하여 고정된 길이의 레코드 형태를 가진 데이터를 Tibero 서버로 로드하려고 한다.

컨트롤 파일 작성

컨트롤 파일인 control.ctl의 내용은 다음과 같다.

```
LOAD DATA
INFILE './data.dat'
APPEND
INTO TABLE MEMBER
LINES FIX 51
TRAILING NULLCOLS
(
    id        position (01:04) integer external,
    name      position (06:15),
    job       position (17:25),
    birthdate position (27:36),
    city      position (38:47),
    age       position (49:50) integer external
)
```

다음은 컨트롤 파일에 지정된 정보이다.

- 데이터 파일

  tbloader 명령을 수행하는 현재 디렉터리에 위치한 data.dat 파일이다.

- 로그 파일

사용자가 명령 프롬프트에서 로그 파일명을 명시하지 않는다면, 기본적으로 컨트롤 파일명으로 설정한 control.log인 로그 파일을 생성한다.

- 오류 파일

  사용자가 명령 프롬프트에서 오류 파일명을 명시하지 않는다면, 기본적으로 데이터 파일명으로 설정한 data.bad인 오류 파일을 생성한다. 단, 오류 레코드가 존재하지 않을 때는 오류 파일을 생성하지 않는다.

- 대상 테이블

  사용자는 member 테이블의 id, name, job, birthdate, city, age 컬럼의 데이터를 로드하려고 한다.

- FIELDS TERMINATED BY 문자열, FIELDS OPTIONALLY ENCLOSED BY 문자열, FIELDS ESCAPED BY 문자열, LINES TERMINATED BY 문자열

  FIELDS TERMINATED BY, FIELDS OPTIONALLY ENCLOSED BY, FIELDS ESCAPED BY 문자열들은 고정된 레코드 형태이기 때문에 사용하지 않는다. LINES FIX 구문을 사용했기 때문에 LINES TERMINATED BY 문자열은 사용하지 않는다.

- 로딩 대상

  데이터 파일의 첫 번째 라인부터 데이터를 로딩한다.

- “4.8.19. TRAILING NULLCOLS 구문”

  데이터 파일의 레코드의 값이 존재하지 않는 경우 NULL 문자로 바인딩한다.

데이터 파일 작성

데이터 파일인 data.dat의 내용은 다음과 같다.

tbLoader 유틸리티가 고정된 레코드 형태의 데이터를 입력받기 때문에 사용자는 정확한 위치에 컬럼의 데이터를 입력해야 한다. 아래의 내용은 편의상 라인 구분을 하였지만, 실제 사용하는 경우에는 한 줄로 기술하여 사용해야 한다.

```
7777 KKS        CHAIRMAN  1975-11-18 SEOUL      33
7839 KING       PRESIDENT 1963-04-13         DAEGU
7934 MILLER     CLERK     1967-01-24 BUSAN      41
7566 JONES      MANAGER         1955-08-20      53
7499 ALLEN      SALESMAN  1977-12-28
7648 CHAN       ANALYST   1979-10-11 ULSAN      28
```

tbLoader 유틸리티 실행

다음과 같이 tbLoader 유틸리티를 실행한다.

```
tbloader userid=loader/loader_pw@default control=./control.ctl
```

로그 파일 및 오류 파일 확인

tbLoader 유틸리티를 실행한 후 생성된 로그 파일과 오류 파일을 확인한다.

다음은 tbLoader 유틸리티의 실행 과정을 기록한 로그 파일의 내용이다.

```
tbLoader 7
TmaxData Corporation Copyright (c) 2008-. All rights reserved.

Data File : ./data.dat
Bad File : ./data.bad

Table 'MEMBER' was loaded from the data file.

Column Name                    Position DataType
------------------------------ ---------- ------------
ID                                    1 NUMERIC EXTERNAL
NAME                                  2 CHARACTER
JOB                                   3 CHARACTER
BIRTHDATE                             4 DATE
CITY                                  5 CHARACTER
AGE                                   6 NUMERIC EXTERNAL

Record 2 was rejected.
  TBR-5074: Given string does not represent a number in proper format.

Record 4 was rejected.
  TBR-5045: Only TIME format can be omitted.

Table 'MEMBER'
----------------
6 Rows were requested to load.
4 Rows were loaded successfully.
2 Rows were failed to load because of some errors

Elapsed time was: 00:00:00.022404
```

tbLoader 유틸리티가 로드에 실패한 오류 데이터를 기록한 오류 파일은 다음과 같다. 사용자는 오류가 발생한 데이터를 수정하여 Tibero 의 데이터베이스에 다시 업로드를 한다.

```
7839 KING       PRESIDENT 1963-04-13         DAEGU
7566 JONES      MANAGER        1955-08-20       53
```

4.10.4. 대용량 객체형 데이터를 포함하는 경우

Conventional Path Load 방법을 이용하여 분리된 레코드 형태의 데이터를 Tibero 서버로 로드하려고 한다. 본 예제는 대용량 바이너리 데이터를 저장하는 데이터 타입인 BLOB 컬럼을 포함하는 경우 사용자가 입력 파일을 작성하는 방법을 보여준다. 대용량 텍스트 데이터를 저장할 수 있는 데이터 타입인 CLOB의 경우도 이와 유사하다.

컨트롤 파일 작성

컨트롤 파일인 control.ctl의 내용은 다음과 같다.

```
LOAD DATA
INFILE  './data.dat'
LOGFILE './logfile.log'
BADFILE './badfile.bad'
APPEND
INTO TABLE member
FIELDS TERMINATED BY ','
       OPTIONALLY ENCLOSED BY '"'
       ESCAPED BY '\\'
LINES TERMINATED BY '\n'
TRAILING NULLCOLS
IGNORE 2 LINES
(
    id integer external,
    name,
    job,
    birthdate,
    city,
    age integer external,
    picture outfile
)
```

다음은 컨트롤 파일에 지정된 정보이다.

- 데이터 파일

  현재 디렉터리에 위치한 data.dat 파일이다.

- 로그 파일

  사용자가 명령 프롬프트에서 로그 파일명을 명시하지 않는다면, 현재 디렉터리에 이름이 logfile.log인 로그 파일을 생성한다.

- 오류 파일

  사용자가 명령 프롬프트에서 오류 파일명을 명시하지 않는다면, 현재 디렉터리에 이름이 badfile.bad인 오류 파일을 생성한다. 단, 오류 레코드가 존재하지 않을 때는 오류 파일을 생성하지 않는다.

- 대상 테이블

  사용자는 member라는 테이블의 id, name, job, birthdate, city, age, picture 컬럼의 데이터를 로드하려고 한다.

- FIELDS TERMINATED BY 문자열, FIELDS OPTIONALLY ENCLOSED BY 문자열, FIELDS ESCAPED BY 문자열, LINES TERMINATED BY 문자열

  FIELDS TERMINATED BY로써 콤마(,), FIELDS OPTIONALLY ENCLOSED BY 문자열로써 큰따옴표(" "), FIELDS ESCAPED BY 문자열은 '\\'를 LINES TERMINATED BY 문자열로 '\n'를 사용한다.

- 로딩 대상

  IGNORE LINES 구문이 있기 때문에 데이터 파일의 세 번째 라인부터 데이터를 로딩한다.

- “4.8.19. TRAILING NULLCOLS 구문”

  데이터 파일의 레코드의 값이 존재하지 않는 경우 NULL 문자로 바인딩한다.

데이터 파일 작성

데이터 파일인 data.dat의 내용은 다음과 같다.

BLOB 컬럼에 바이너리 데이터를 업로드하기 위해서 해당 바이너리 파일이 위치한 경로를 입력한다.

```
첫 번째 라인 무시
7782,"Clark","Manager",1981-01-11 , DAEGU,26,./blob.jpg
7839,"King",President,1960/11/17,SEOUL,47
7934,"Miller","Clerk",1977/10/12,BUSAN,30,./blob2.jpg
7566,"Jones",Manager\, ,1981/04/02,31
7499, "Allen", "Salesman" a,1981:02/20,26
7654, "Martin", "Sale smn", 1981/10/28,26
7658, "Chan",Ana lyst, 1982/05/03,25,25
```

tbLoader 유틸리티 실행

다음과 같이 tbLoader 유틸리티를 실행한다.

```
tbloader userid=loader/loader_pw@default control=./control.ctl
```

로그 파일 및 오류 파일 확인

tbLoader 유틸리티를 실행한 후 생성된 로그 파일과 오류 파일을 확인한다.

tbLoader 유틸리티의 실행 과정을 기록한 로그 파일의 내용은 다음과 같다.

```
tbLoader 7
TmaxData Corporation Copyright (c) 2008-. All rights reserved.

Data File : ./data.dat
Bad File : ./badfile.bad

Table 'MEMBER' was loaded from the data file.

Column Name                     Position DataType
------------------------------ ---------- ------------
ID                                      1 NUMERIC EXTERNAL
NAME                                    2 CHARACTER
JOB                                     3 CHARACTER
BIRTHDATE                               4 DATE
CITY                                    5 CHARACTER
AGE                                     6 NUMERIC EXTERNAL
PICTURE                                 7 RAW

Record 4 was rejected.
  TBR-80025 : Column data exceeds data buffer size.

Table 'MEMBER'
---------------
6 Rows were requested to load.
5 Rows were loaded successfully.
1 Rows were failed to load because of some errors

Elapsed time was: 00:00:00.055475
```

tbLoader 유틸리티가 로드에 실패한 오류 데이터를 기록한 오류 파일은 다음과 같다. 오류가 발생한 데이터는 수정해서 Tibero의 데이터베이스에 다시 업로드를 해야 한다.

```
7499, "Allen", "Salesman" a,1981:02/20,26
```

제5장 tbdv

본 장에서는 tbdv 유틸리티를 소개하고 사용 방법을 설명한다.

5.1. 개요

tbdv는 Tibero 데이터베이스의 데이터 파일에 대한 정합성을 검사하는 간단한 유틸리티이다. 이 유틸리티를 통해 데이터베이스가 오프라인일 때도 데이터파일에 대한 기본적인 정합성 검사를 수행할 수 있다.

tbdv 유틸리티는 데이터 블록 각각에 대해 다음의 사항을 체크한다.

- 블록에 DBA가 올바로 적혀있는가?
- 블록의 체크섬이 일치하는가?
- 블록의 가용공간과 실제 사용한 공간을 더한 것이 블록 크기와 일치하는가?
- 블록에 들어가는 row-piece들이 서로의 영역을 침범하지 않고 들어가 있는가?

이러한 사항들 중 한 개라도 정합성 검사에 실패한다면 미디어 장애로 판단할 수 있으며 media recovery를 수행하여 데이터베이스를 복구해야 한다.

5.2. 빠른 시작

tbdv 유틸리티는 Tibero를 설치하는 과정에서 함께 설치되며, Tibero를 제거하면 함께 제거된다.

tbdv 유틸리티는 명령 프롬프트에서 다음과 같은 형식으로 실행한다.

```
$ tbdv [-s BLKSIZE] [-l CHECK_LENGTH] /path/to/datafile
```

명령 프롬프트에서는 옵션으로 블록 크기를 지정할 수 있으며 기본은 8192byte이다.

파일의 앞쪽 일부분만 체크하고 싶은 경우 CHECK_LENGTH를 지정하는 옵션을 사용할 수 있다. 지정하지 않은 경우 파일 전체에 대해서 체크를 진행하게 된다. RAW Device의 경우 명시적으로 RAW Device의 크기를 입력해야 하며, 입력하지 않은 경우 에러를 발생시키게 된다.

다음은 tbdv 유틸리티를 실행하는 예이다.

[예 5.1] tbdv 유틸리티의 실행

```
$ tbdv df1.dtf
================================================================
= Database Verifier (DV) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
```

```
Verifying 'df1.dtf'...



Verifying complete.

Total blocks: 1152
Processed blocks: 1063
Empty blocks: 89
Corrupt blocks: 0
```

5.3. 수행 예제

본 절에서는 정상적인 데이터 파일 및 정합성이 깨진 블럭이 있는 데이터 파일들에 대해 각각 tbdv 유틸리티가 어떻게 정합성 오류를 발견하는지 살펴본다.

데이터 블록의 dba가 적혀있는 부분에 장애를 일으키면 dv는 다음과 같은 결과를 출력한다.

[예 5.2] DBA가 잘못됐을 때 dv의 출력

```
$ tbdv df1.dtf
================================================================
= Database Verifier (DV) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
Verifying 'df1.dtf'...

block #2351 is misplaced.dba differs (expected=16779567, real=16779551)

Verifying complete.

Total blocks: 2433
Processed blocks: 2343
Empty blocks: 90
Corrupt blocks: 1
```

데이터 파일에서 fractured block을 발견했을때는 데이터 블록이 consistent하지 않다는 결과를 출력한다.

[예 5.3] Fractured block(Inconsistent block)을 발견했을 때 dv의 출력

```
$ tbdv df1.dtf
================================================================
= Database Verifier (DV) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
```

```
Verifying 'df1.dtf'...

block #2311 isn't consistent.

Verifying complete.

Total blocks: 2433
Processed blocks: 2343
Empty blocks: 90
Corrupt blocks: 1
```

데이터 블록의 남은 공간을 기재하는 부분이 블록 크기에서 실제로 사용한 공간의 차이와 불일치할 경우 dv의 출력은 다음과 같다.

[예 5.4] 블록의 빈 공간이 실제 사용한 공간과 정합성이 맞지 않는 경우 dv의 출력

```
$ tbdv df1.dtf
================================================================
= Database Verifier (DV) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
Verifying 'df1.dtf'...

block #2004 has incorrect freespace.

Verifying complete.

Total blocks: 2433
Processed blocks: 2343
Empty blocks: 90
Corrupt blocks: 1
```

참고

tbdv 유틸리티는 수행과정에서 zero-out된 블록은 포맷이 안되었다고 간주하기 때문에 실제로 디스크에 장애가 발생하여 데이터가 써진 블록이 zero-out되는 현상이 발생하여도 tbdv는 블록이 포맷되지 않았다고 판단하고(즉, 아직 데이터 파일에서 할당되지 않은 부분) 에러를 출력하지 않으므로 사용에 주의한다.

제6장 유틸리티 API

Tibero에서는 유틸리티와 관련된 C와 C++ 함수를 제공하고 있다. 애플리케이션 프로그램 개발자는 유틸리티 API를 사용하여 기존에 명령 프롬프트에서 수행하던 유틸리티를 애플리케이션 프로그램에서 호출할 수 있다.

6.1. 헤더 파일

다음은 유틸리티 API가 사용하는 헤더 파일이다.

헤더 파일명	설명
tbutil.h	유틸리티 API를 선언하고 이와 관련된 구조체를 정의한다. 정의할 구조체는 다음과 같다. – struct sqlstr – struct TBExpImpMeta – struct TBExportIn – struct TBExportOut – struct TBExportStruct – struct TBImportIn – struct TBImportOut – struct TBImportStruct
sqlca.h	유틸리티 내부에서 발생한 에러 정보를 애플리케이션 프로그램에 전달하는 구조체를 선언한다. 선언할 구조체는 다음과 같다. – struct sqlca
sqlcli.h	ODBC 표준인 헤더 파일이며, 표준 API와 매크로를 정의한다.

6.2. 구조체

다음은 유틸리티 API에서 정의할 수 있는 구조체이다.

- sqlstr 구조체

필드	용도	타입	설명
length	입력	SQLSMALLINT	문자열의 길이를 명시한다.
data	입력	SQLCHAR *	문자열이 저장된 포인터를 명시한다.

- TBExpImpMeta 구조체

필드	용도	타입	설명
fieldTerm	입력	SQLCHAR *	컬럼(필드)의 구분자로, 사용할 문자열을 명시한다.
fieldTermLen	입력	SQLSMALLINT	컬럼(필드)의 구분자로, 사용할 문자열의 길이를 명시한다.
lineTerm	입력	SQLCHAR *	레코드의 구분자로, 사용할 문자열을 명시한다.
lineTermLen	입력	SQLSMALLINT	레코드의 구분자로, 사용할 문자열의 길이를 명시한다.
enclStart	입력	SQLCHAR *	컬럼의 데이터를 감쌀 시작 문자열을 명시한다. 시작 문자열을 큰따옴표(" ")로 명시하면, 컬럼의 데이터 맨 앞에 큰따옴표가 붙게 된다.
enclStartLen	입력	SQLSMALLINT	컬럼의 데이터를 감쌀 시작 문자열의 길이를 명시한다.
enclEnd	입력	SQLCHAR *	컬럼의 데이터를 감쌀 종료 문자열을 명시한다. 종료 문자열을 큰따옴표(" ")로 명시하면, 컬럼의 데이터 맨 뒤에 큰따옴표가 붙게 된다.
enclEndLen	입력	SQLSMALLINT	컬럼의 데이터를 감쌀 종료 문자열의 길이를 명시한다.
escape	입력	SQLCHAR *	컬럼의 데이터를 해석할 때 필요한 ESCAPE 문자열을 명시한다. 단, TBImport 구조체에서만 사용한다.
escapeLen	입력	SQLSMALLINT	컬럼의 데이터를 해석할 때 필요한 ESCAPE 문자열의 길이를 명시한다. 단, TBImport 구조체에서만 사용한다.

- TBExportIn 구조체

필드	용도	타입	설명
iMeta	입력	TBExpImpMeta *	TBExpImpMeta 구조체의 포인터이다. 데이터를 추출할 때 필요한 입력 메타데이터를 명시한다.

- TBExportOut 구조체

필드	용도	타입	설명
oRowsExported	출력	SQLINTEGER	데이터 파일에 추출된 레코드의 개수가 기록된다.

- TBExportStruct 구조체

필드	용도	타입	설명
piDataFileName	입력	SQLCHAR *	추출된 데이터가 저장될 파일의 경로명을 명시한다.
iDataFileNameLen	입력	SQLSMALLINT	데이터 파일명의 길이를 명시한다. 파일명이 NULL 문자로 종료된 경우 SQL_NTS를 명시한다.
piActionString	입력	sqlstr *	SELECT 문을 이용하여 대상 테이블 또는 뷰의 데이터를 지정한다. SELECT 문에 명시한 컬럼의 순서대로 추출된 데이터가 저장된다.
iFileType	입력	SQLSMALLINT	현재는 컬럼 구분자를 사용하는 형태인 SQL_DEL만 사용할 수 있다.
piMsgFileName	입력	SQLCHAR *	데이터를 추출할 때 발생하는 에러와 경고 그리고 유용한 정보를 기록하는 메시지 파일명을 명시한다.
iMsgFileNameLen	입력	SQLSMALLINT	메시지 파일명의 길이를 명시한다. 파일명이 NULL 문자로 종료된 경우 SQL_NTS를 명시한다.
piExportInfoIn	입력	TBExportIn *	TBExportIn 구조체의 포인터이다. 데이터를 추출할 때 필요한 입력 메타데이터를 명시한다.
piExportInfoOut	입력	TBExportOut *	TBExportOut 구조체의 포인터이다. 데이터를 추출한 후에 발생한 결과정보를 저장한다.

- TBImportIn 구조체

필드	용도	타입	설명
iMeta	입력	TBExpImpMeta *	TBExpImpMeta 구조체의 포인터이다. 데이터를 로드할 때 필요한 입력 메타데이터를 명시한다.
iRowCount	입력	SQLINTEGER	로드할 레코드의 개수를 명시한다.

필드	용도	타입	설명
			사용자가 값을 0으로 지정한 경우 데이터 파일의 모든 레코드를 로드한다.
iSkipCount	입력	SQLINTEGER	데이터 파일에서 로드할 대상을 제외하기 위해 건너뛸 레코드의 개수를 명시한다.
iCommitCount	입력	SQLINTEGER	데이터를 로드 중에 한 번씩 커밋하려고 할 때 그 개수를 명시한다. 단, 성능적인 이슈로 사용자가 명시한 개수 단위로 커밋이 일어나지는 않는다.
iErrorCount	입력	SQLINTEGER	사용자가 허용하는 에러 레코드의 개수이다. 데이터를 로드 중에 에러 레코드 개수가 허용치보다 많으면, 로드를 중단한다.

- TBImportOut 구조체

필드	용도	타입	설명
oRowsRead	출력	SQLINTEGER	데이터 파일에서 읽은 레코드의 개수를 기록한다.
oRowsSkipped	출력	SQLINTEGER	데이터 파일에서 건너뛴 레코드의 개수를 기록한다.
oRowsInserted	출력	SQLINTEGER	해당 테이블 또는 뷰에 입력(INSERT 연산)된 레코드의 개수를 기록한다.
oRowsUpdated	출력	SQLINTEGER	해당 테이블 또는 뷰에 수정(UPDATE 연산)된 레코드의 개수를 기록한다.
oRowsRejected	출력	SQLINTEGER	데이터 로드에 실패한 레코드의 개수를 기록한다.
oRowsCommitted	출력	SQLINTEGER	커밋에 성공한 레코드의 개수를 기록한다.

- TBImportStruct 구조체

필드	용도	타입	설명
piDataFileName	입력	SQLCHAR *	로드할 데이터 파일의 경로명을 명시한다.
iDataFileNameLen	입력	SQLSMALLINT	데이터 파일명의 길이를 명시한다. 파일명이 NULL 문자로 종료된 경우 SQL_NTS를 명시한다.
piActionString	입력	sqlstr *	로드할 대상 테이블과 컬럼을 명시하고, 상세한 메타데이터를 명시한 문자열이다. tbLoader 유틸리티의 컨트롤 파일에서 사용하는 문자열과 동일한 문법이다. 이 문법에 대한 내용은 tbLoader 유틸리티의 컨트롤 파일의 형식을 참고한다.

필드	용도	타입	설명
iFileType	입력	SQLSMALLINT	현재는 컬럼 구분자를 사용하는 형태인 SQL_DEL만 사용할 수 있다.
piMsgFileName	입력	SQLCHAR *	데이터를 로드할 때 발생하는 에러와 경고 그리고 유용한 정보를 기록하는 메시지 파일명을 명시한다.
iMsgFileNameLen	입력	SQLSMALLINT	메시지 파일명의 길이를 명시한다. 파일명이 NULL 문자로 종료된 경우 SQL_NTS를 명시한다.
piBadFileName	입력	SQLCHAR *	데이터를 로드할 때 에러가 발생한 데이터를 모아서 기록하는 오류 데이터 파일이다.
iBadFileNameLen	입력	SQLSMALLINT	오류 데이터 파일명의 길이를 명시한다. 파일명이 NULL 문자로 종료된 경우 SQL_NTS를 명시한다.
iDPL	입력	BOOL	데이터를 로드할 때 Direct Path Load 방식의 사용 여부를 지정한다.
iTrailNullCols	입력	BOOL	데이터 파일의 레코드에 없는 마지막 컬럼의 데이터를 NULL로 취급 여부를 지정한다.
piImportInfoIn	입력	TBImportIn *	TBImportIn 구조체의 포인터이다. 데이터를 로드할 때 필요한 입력 메타데이터를 지정한다.
poImportInfoOut	입력	TBImportOut *	TBImportOut 구조체의 포인터이다. 데이터를 로드한 후에 발생한 결과 정보를 로드한다.

6.3. 유틸리티 API 목록

다음은 유틸리티 API 목록이다.

함수 타입	함수	헤더 파일
데이터베이스 연결	TBConnect	tbutil.h
데이터베이스연결 해제	TBDisconnect	tbutil.h
데이터 추출	TBExport	tbutil.h
데이터 로드	TBImport	tbutil.h

6.3.1. TBConnect

데이터베이스 연결 정보(SID, 사용자명, 패스워드)를 이용하여 Tibero 서버에 연결하는 함수이다.

TBConnect 함수의 세부 내용은 다음과 같다.

- 문법

```
SQLRETURN SQL_API
TBConnect(SQLCHAR *dnsname, SQLCHAR *username, SQLCHAR *pwd,
          struct sqlca *pSqlca);
```

- 파라미터

파라미터	용도	설명
dnsname	입력	tbdsn.tbr(또는 tbnet_alias.tbr) 파일에 정의된 SID이다.
username	입력	사용자명을 지정한다.
pwd	입력	패스워드를 지정한다.
pSqlca	출력	반환 코드가 SQL_SUCCESS가 아닌 경우 유틸리티 내부에서 발생한 에러 정보를 포함한다.

- 반환 코드

반환 코드	설명
SQL_SUCCESS	함수가 성공적으로 완료된 상태
SQL_SUCCESS_WITH_INFO	함수가 성공적으로 완료되었으나, 경고 메시지가 있는 상태
SQL_ERROR	치명적인 에러가 발생한 상태

- 관련 함수

  TBDisconnect

6.3.2. TBDisconnect

데이터베이스 연결정보(SID)에 해당하는 Tibero 서버와의 연결을 해제하는 함수이다.

TBDisconnect 함수의 세부 내용은 다음과 같다.

- 문법

```
SQLRETURN SQL_API
TBDisconnect(SQLCHAR *dnsname, struct sqlca *pSqlca);
```

- 파라미터

파라미터	용도	설명
dnsname	입력	tbdsn.tbr(또는 tbnet_alias.tbr)파일에 정의된 SID이다.
pSqlca	출력	반환 코드가 SQL_SUCCESS가 아닌 경우 유틸리티 내부에서 발생한 에러 정보를 포함한다.

- 반환 코드

반환 코드	설명
SQL_SUCCESS	함수가 성공적으로 완료된 상태
SQL_SUCCESS_WITH_INFO	함수가 성공적으로 완료되었으나, 경고 메시지가 있는 상태
SQL_ERROR	치명적인 에러가 발생한 상태

- 관련 함수

  TBConnect

6.3.3. TBExport

데이터베이스에 존재하는 데이터를 외부 파일로 추출하는 함수이다. 외부 파일은 추출된 데이터를 텍스트 형태로 표시하며, 컬럼 구분자와 레코드 구분자를 이용하여 컬럼과 레코드를 추출한다. 이렇게 추출된 외부 파일은 tbLoader 유틸리티의 데이터 파일 형태를 취한다. 사용자는 SELECT 문을 이용하여 추출할 데이터를 선정할 수 있다. 단, 대상 테이블 또는 뷰의 SELECT 권한이 있어야 한다.

`TBExport` 함수의 세부 내용은 다음과 같다.

- 문법

```
SQLRETURN SQL_API
TBExport(SQLINTEGER versionNumber, TBExportStruct *pParamStruct,
         struct sqlca *pSqlca);
```

- 파라미터

파라미터	용도	설명
versionNumber	입력	유틸리티 라이브러리의 버전 번호이다. 이 파라미터는 유틸리티 라이브러리가 변경될 것에 대비하여 하위 버전과의 호환성을 위해 존재한다. 현재 버전 번호는 1이다.
pParamStruct	입출력	이 파라미터를 사용하여 추출할 대상 정보를 명시하고, 추출 후에는 결과 정보를 받는다. 자세한 내용은 "TBExportStruct 구조체"를 참고한다.
pSqlca	출력	반환 코드가 SQL_SUCCESS가 아닌 경우 유틸리티 내부에서 발생한 에러 정보를 포함한다.

- 반환 코드

반환 코드	설명
SQL_SUCCESS	함수가 성공적으로 완료된 상태
SQL_SUCCESS_WITH_INFO	함수가 성공적으로 완료되었으나, 경고 메시지가 있는 상태
SQL_ERROR	치명적인 에러가 발생한 상태

- 예제

```
#include "tbutil.h"

#define DNS_NAME  "DEFAULT"
#define USER_NAME "SYS"
#define PWD       "tibero"

int main(int argc, char *argv[]) {
    SQLRETURN      rc = SQL_SUCCESS;
    SQLINTEGER     versionNumber = 1;
    SQLCHAR        dataFileName[256];
    SQLCHAR        actionString[256];
    SQLCHAR        msgFileName[256];
    SQLCHAR        fieldTerm[5];
    SQLCHAR        lineTerm[5];
    SQLCHAR        enclStart[5];
    SQLCHAR        enclEnd[5];

    struct sqlca   ca           = {"\0", 0, 0, {0, "\0"}, "\0",
                                   {0, 0, 0, 0, 0, 0}, "\0", "\0"};
    TBExportStruct exportStruct = {NULL, 0, NULL, NULL, NULL, 0, NULL,
                                   0, NULL, NULL};
    TBExportIn     exportIn     = {{NULL, 0, NULL, 0, NULL, 0, NULL, 0, NULL, 0}};

    TBExportOut    exportOut    = {0};

    rc= TBConnect((SQLCHAR *)DNS_NAME, (SQLCHAR *)USER_NAME, (SQLCHAR *)PWD, &ca);


    if (rc != SQL_SUCCESS) return -1;

    strcpy((char *)dataFileName, "./all_tables.dat");
    strcpy((char *)actionString, "select * from all_tables");
    strcpy((char *)msgFileName, "./all_tables.log");
    strcpy((char *)fieldTerm, ",");
    strcpy((char *)lineTerm, "\n");
    strcpy((char *)enclStart, "\"");
    strcpy((char *)enclEnd, "\"");

    /* setting data file name */
```

```
    exportStruct.piDataFileName   = dataFileName;
    exportStruct.iDataFileNameLen = strlen((char *)dataFileName);

    /* setting action String */
    exportStruct.piActionString = calloc(1, sizeof(sqlstr));
    exportStruct.piActionString->data   = actionString;
    exportStruct.piActionString->length = strlen((char *)actionString);

    /* setting file type */
    exportStruct.iFileType = SQL_DEL;

    /* setting message file name */
    exportStruct.piMsgFileName   = msgFileName;
    exportStruct.iMsgFileNameLen = strlen((char *)msgFileName);

    /* setting field term, line term etc.. */
    exportIn.iMeta.fieldTerm    = fieldTerm;
    exportIn.iMeta.fieldTermLen = strlen((char *)fieldTerm);
    exportIn.iMeta.lineTerm     = lineTerm;
    exportIn.iMeta.lineTermLen  = strlen((char *)lineTerm);
    exportIn.iMeta.enclStart    = enclStart;
    exportIn.iMeta.enclStartLen = strlen((char *)enclStart);
    exportIn.iMeta.enclEnd      = enclEnd;
    exportIn.iMeta.enclEndLen   = strlen((char *)enclEnd);

    /* setting export input, output information */
    exportStruct.piExportInfoIn  = &exportIn;
    exportStruct.poExportInfoOut = &exportOut;

    /* setting file type */
    rc = TBExport(versionNumber, &exportStruct, &ca);
    if (rc != SQL_SUCCESS) return -1;

    /* disconnect */
    rc= TBDisconnect((SQLCHAR *)DNS_NAME, &ca);
    if (rc != SQL_SUCCESS) return -1;

    return 1;
}
```

6.3.4. TBImport

외부 파일에 존재하는 데이터를 데이터베이스에 로드하는 함수이다. 여기서 외부 파일은 tbLoader 유틸리티가 지원하는 고정된 레코드 형태와 분리된 레코드 형태를 지원한다. 단, 대상 테이블 또는 뷰의 INSERT 권한이 있어야 한다.

TBImport 함수의 세부 내용은 다음과 같다.

- 문법

```
SQLRETURN SQL_API TBImport(SQLINTEGER versionNumber,
                           TBImportStruct *pParamStruct,
                           struct sqlca *pSqlca);
```

- 파라미터

파라미터	용도	설명
versionNumber	입력	유틸리티 라이브러리의 버전 번호이다. 이 파라미터는 유틸리티 라이브러리가 변경될 것에 대비하여 하위 버전과의 호환성을 위해 존재한다. 현재 버전 번호는 1이다.
pParamStruct	입출력	이 파라미터를 사용하여 로드할 대상 정보를 명시하고, 로드 후에는 결과 정보를 받는다. 자세한 내용은 "TBImportStruct 구조체"를 참고한다.
pSqlca	출력	반환 코드가 SQL_SUCCESS가 아닌 경우 유틸리티 내부에서 발생한 에러 정보를 포함한다.

- 반환 코드

반환 코드	설명
SQL_SUCCESS	함수가 성공적으로 완료된 상태
SQL_SUCCESS_WITH_INFO	함수가 성공적으로 완료되었으나, 경고 메시지가 있는 상태
SQL_ERROR	치명적인 에러가 발생한 상태

- 예제

```
#include "tbutil.h"

#define DNS_NAME  "DEFAULT"
#define USER_NAME "SYS"
#define PWD       "tibero"

int main(int argc, char *argv[]) {
 SQLRETURN      rc = SQL_SUCCESS;
 SQLINTEGER     versionNumber = 1;
 SQLCHAR        dataFileName[256];
 SQLCHAR        actionString[256];
 SQLCHAR        msgFileName[256];
 SQLCHAR        badFileName[256];
 SQLCHAR        fieldTerm[5];
 SQLCHAR        lineTerm[5];
 SQLCHAR        enclStart[5];
 SQLCHAR        enclEnd[5];
```

```
SQLCHAR        escape[5];

struct sqlca   ca           = {"\0", 0, 0, {0, "\0"}, "\0",
                               {0, 0, 0, 0, 0, 0}, "\0", "\0"};
TBImportStruct importStruct = {NULL, 0, NULL, NULL, NULL, 0, NULL,
                                0, NULL, 0, 0, NULL, NULL};
TBImportIn     importIn     = {{NULL, 0, NULL, 0, NULL, 0, NULL, 0, NULL, 0},
                                0, 0, 0, 0};
TBImportOut    importOut    = {0, 0, 0, 0, 0, 0};

rc= TBConnect((SQLCHAR *)DNS_NAME, (SQLCHAR *)USER_NAME, (SQLCHAR *)PWD, &ca);
if (rc != SQL_SUCCESS) return -1;

strcpy((char *)actionString, "LOAD DATA "
                             "APPEND "
                             "INTO TABLE DEPT "
                             "MULTI INSERT INDEXES "
                            "(position, deptno, dname CONSTANT \"co dep\", loc)");

strcpy((char *)dataFileName, "./test.dat");
strcpy((char *)msgFileName, "./test.log");
strcpy((char *)badFileName, "./test.bad");
strcpy((char *)fieldTerm,   ",b");
strcpy((char *)lineTerm,    "abbb\n");
strcpy((char *)enclStart,   "{$");
strcpy((char *)enclEnd,     "$}");
strcpy((char *)escape,      "XX");

/* setting data file name */
importStruct.piDataFileName   = dataFileName;
importStruct.iDataFileNameLen = strlen((char *)dataFileName);

/* setting action String */
importStruct.piActionString = calloc(1, sizeof(sqlstr));
importStruct.piActionString->data   = actionString;
importStruct.piActionString->length = strlen((char *)actionString);

/* setting file type */
importStruct.iFileType = SQL_DEL;

/* setting message file name */
importStruct.piMsgFileName   = msgFileName;
importStruct.iMsgFileNameLen = strlen((char *)msgFileName);

/* setting bad data file name */
importStruct.piBadFileName   = badFileName;
importStruct.iBadFileNameLen = strlen((char *)badFileName);
```

```
 /* turn on DPL mode */
 importStruct.iDPL            = 2;

 /* setting field term, line term etc.. */
 importIn.iMeta.fieldTerm    = fieldTerm;
 importIn.iMeta.fieldTermLen = strlen((char *)fieldTerm);
 importIn.iMeta.lineTerm     = lineTerm;
 importIn.iMeta.lineTermLen  = strlen((char *)lineTerm);
 importIn.iMeta.enclStart    = enclStart;
 importIn.iMeta.enclStartLen = strlen((char *)enclStart);
 importIn.iMeta.enclEnd      = enclEnd;
 importIn.iMeta.enclEndLen   = strlen((char *)enclEnd);
 importIn.iMeta.escape       = escape;
 importIn.iMeta.escapeLen    = strlen((char *)escape);

 importIn.iRowcount          = 0;
 importIn.iSkipcount         = 0;
 importIn.iCommitcount       = 2;
 importIn.iErrorcount        = 50;

 /* setting export input, output information */
 importStruct.piImportInfoIn  = &importIn;
 importStruct.poImportInfoOut = &importOut;

 /* setting file type */
 rc = TBImport(versionNumber, &importStruct, &ca);
 if (rc != SQL_SUCCESS) return -1;

   fprintf(stdout, "oRowsRead[%ld]", importOut.oRowsRead);
   fprintf(stdout, "oRowsSkipped[%ld]", importOut.oRowsSkipped);
   fprintf(stdout, "oRowsInserted[%ld]", importOut.oRowsInserted);
   fprintf(stdout, "oRowsUpdated[%ld]", importOut.oRowsUpdated);
   fprintf(stdout, "oRowsRejected[%ld]", importOut.oRowsRejected);
   fprintf(stdout, "oRowsCommitted[%ld]", importOut.oRowsCommitted);

   /* disconnect */
   rc= TBDisconnect((SQLCHAR *)DNS_NAME, &ca);
   if (rc != SQL_SUCCESS) return -1;

   return 1;
}
```

색인

L

M

N

O

P

Q

R

S

T

U

V

W

ㄱ

ㄷ

ㄹ

ㅁ

ㅂ

ㅇ

ㅈ

ㅋ

ㅌ