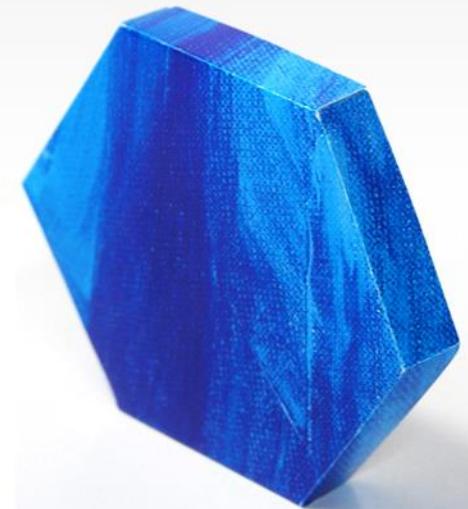




Tibero SQL for Developers



Update History

Contents (1/4)

● Chapter 1장. Tibero 소개 및 기능

1. Tibero Architecure

● Chapter 2장. 설치 및 Directory.

1. 설치
2. Directory

● Chapter 3장. tbSQL

1. OverView
2. 실행
3. 데이터베이스 접속
4. 인터페이스
5. 환경설정
6. 종료
7. 시스템 변수
8. 기본 기능
9. 고급 기능

10. 명령어

11. 컬럼 포맷

● Chapter 4장. tbAdmin

1. OverView
2. 설치 및 실행
3. 기능
4. Export / Import 기능

● Chapter 5장. Object

1. Object 개요
2. 테이블 관리
3. 인덱스 관리
4. 파티션(Partition) 테이블 관리
5. 파티션(Partition) 인덱스 관리
6. 뷰(View) 관리
7. 시퀀스(Sequence) 관리
8. 시노ним(Synonym) 관리

Contents (2/4)

● Chapter 6장. SQL 문장의 구성요소

1. SQL 문장의 구성요소
2. 데이터 타입
3. 리터럴
4. 형식 문자열
5. 의사 컬럼
6. NULL
7. 주석
8. 힌트

● Chapter 7장. 데이터 조회.

1. SQL 정의
2. SQL 문장 분류
3. SELECT를 이용한 데이터 조회
4. WHERE절 이용
5. AND
6. OR
7. BETWEEN AND
8. IN
9. IS NULL/NOT
10. LIKE
11. 연산자
12. ORDER BY

13. GROUP BY
14. HAVING
15. 기타
16. SQL 작성시 유의 사항

● Chapter 8장. 함수 사용.

1. 함수 정의 / 종류
2. 숫자 함수 (ROUND, MOD...)
3. 문자 함수 (SUBSTR, LOWER...)
4. 날짜 함수 (ADD_MONTH ...)
5. 변환 함수 (TO_CHAR, TO_DATE...)
6. 기타 함수 (DECODE, NVL...)
7. 집합 함수 (AVG, RANK, SUM, MIN...)
8. 분석 함수

Contents (2/4)

● Chapter 9장. 데이터 조회 고급 활용

1. 조인의 정의
2. SELECT FROM을 이용한 조인
3. JOIN 절을 이용한 조인
4. 3개 이상의 테이블 조인
5. 여러 형태의 조인
6. SUBQUERY(하위질의)
7. 스칼라 SUBQUERY
8. INLINE VIEW
9. Top-N 쿼리
10. ROLLUP
11. CUBE
12. GROUPING SETS
13. 계층 질의
14. 조인방식
15. 기타

● CHAPTER 10장. DML 활용

1. DML 정의
2. INSERT 문
3. UPDATE 문
4. DELETE 문
5. MERGE 문
6. TRIGGER
7. Transaction 이란?
8. LOCK

Contents (3/4)

● CHAPTER 11장. tbPSM

1. OverView
2. 구성요소
3. 프로그램 구조
4. tbPSM 문장의 구성요소
5. tbPSM의 데이터 타입
6. 데이터 타입의 변환
7. 데이터 변수의 선언과 참조 영역
8. 연산자
9. 제어 구조
10. 복합 타입
11. 서브프로그램
12. Package
13. SQL 문장의 실행
14. 에러 처리

● CHAPTER 12장. tbJDBC 활용

1. tbJDBC
2. JDK 설치
3. JDBC의 표준 기능
4. 개발 과정
5. 기본 프로그래밍

● Chapter 13장. tbCLI 를 활용하자.

1. OverView
2. 구성요소
3. Program 구조
4. DATA TYPE
5. 함수
6. tbCLI Message
7. tbCLI와 ODBC
8. tbCLI와 ODBC 연동

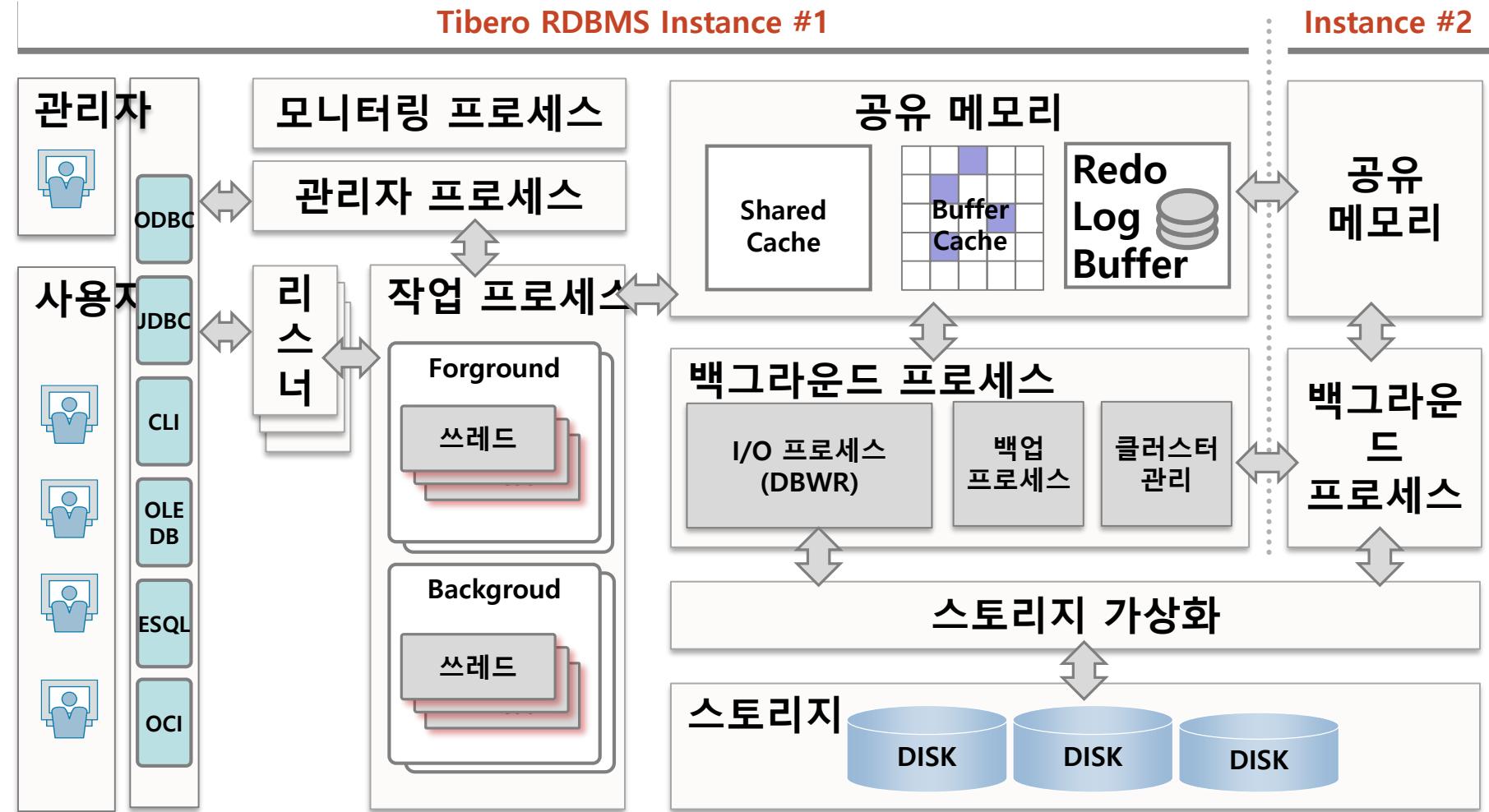
Contents (4/4)

- CHAPTER 14장. 유필리티

1. TBEXPORT
2. TBIMPORT

Chapter 1장 Tibero 소개 및 기능

● Tibero 전체 구조



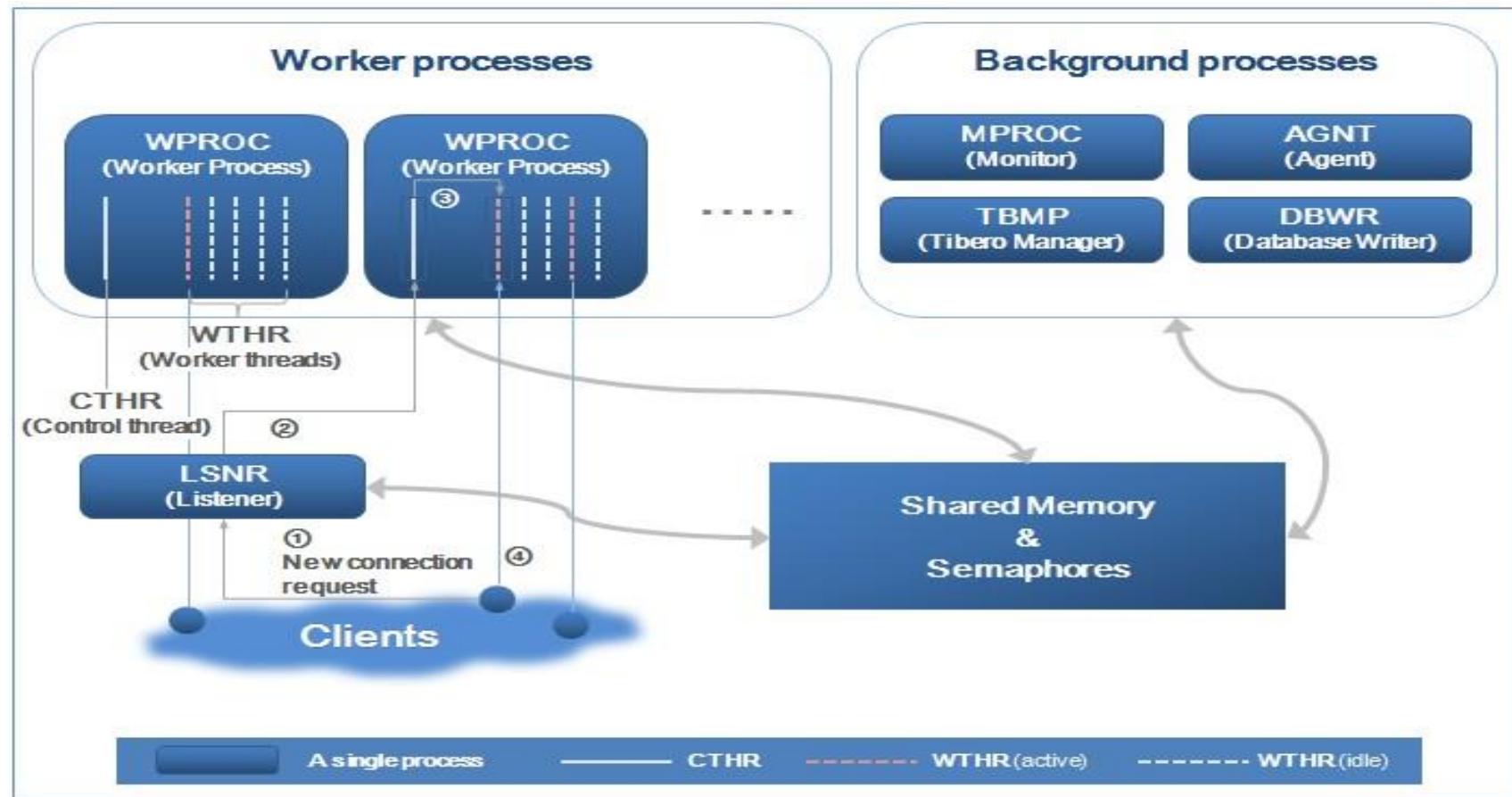
Tibero Process 구조

CHAPTER 1장. Tibero 소개 및 기능.

● Tibero 프로세스 구조

➤ 대규모 사용자 접속을 수용하는 다중 프로세스 및 다중 스레드 기반의 아키텍쳐 구조

- Listener, 워커프로세스 (Working Process 또는 Foregroud Process) , Background Process



● Listener

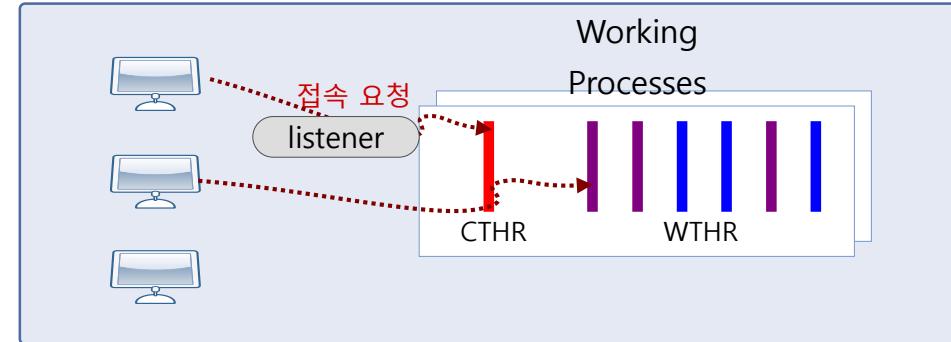
- 클라이언트의 새로운 접속 요청을 받아 이를 유휴한 워커 프로세스에 할당
- 클라이언트와 워커 프로세스간에 중계 역할을 담당하며, 별도의 실행 파일인 `tblistener`를 사용하여 작업
- 모니터링 프로세스에 의해서 생성되며 외부에서 강제 종료하더라도 재 시작 가능
- Listener Multi-Port 지정 가능

```
alter system listener add port 8799;  
alter system listener delete port 8799;  
  
<$TB_SID.tip>  
EXTRA_LISTENER_PORTS=8799;8800;
```

- 클라이언트의 새로운 접속 요청이 이루어지는 순서
 - 1) Client가 접속 요청
 - 2) Listener는 현재 빈 WTHR이 있는 프로세스를 찾아서 이 사용자의 접속 요청을 CTHR에게 넘겨줌.
 - 3) 요청을 받은 CTHR은 자기 자신의 WTHR 상태를 체크해서 일하지 않는 WTHR에게 할당
 - 4) WTHR은 Client와 인증 절차를 걸쳐 세션 시작

● 워커프로세스 (Working Processes 또는 Foregrouud Process)

- 클라이언트와 실제 통신을 하며, 사용자 요구 사항을 처리 하는 프로세스
- Foregrouud Worker Process는 리스너를 통해 들어온 온라인 요청 처리
- Backgrouud Worker Process는 Internal Task나 Job Scheduler에 등록된 배치 작업을 수행 (MAX_BG_SESSION_COUNT으로 설정하며, MAX_SESSION_COUNT작게 설정, _WTHR_PER_PROC 값의 배수로 설정)
- CTHR (Control Thread)
 - 각 Working Process마다 하나씩 생성. 서버 시작 시에 지정된 개수의 Worker Thread를 생성.
 - 시그널 처리 담당.
 - I/O Multiplexing을 지원하며, 필요한 경우 워커 스레드 대신 메시지 송/수신 역할 수행.



➤ WTHR (Working Thread).

- 각 Working Process마다 여러 개 생성.
- Client 가 보내는 메시지를 받아 처리하고 그 결과를 리턴.
- SQL Parsing, 최적화, 수행 등 DBMS가 해야 하는 대부분의 일 처리.

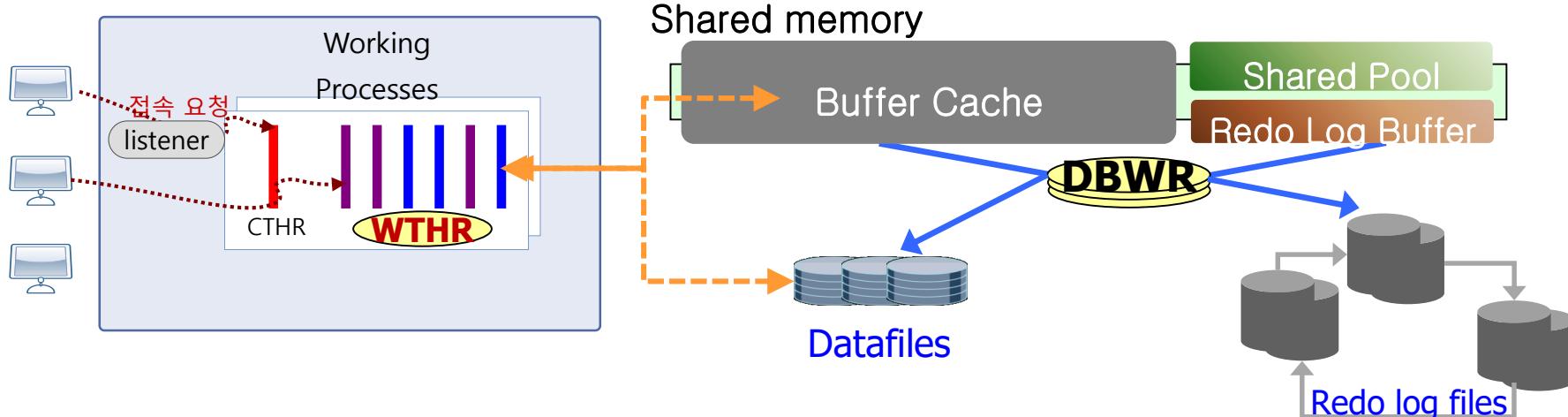
● Working Thread 의 동작

➤ Get DB blocks

- WTHR가 필요한 block을 직접 disk에 Read.
- 읽어온 block들은 shared memory에 두고 (buffer cache), 자신은 물론 다른 WTHR가 필요할 때 재사용 (I/O reduction).

➤ Modify DB blocks

- DB block의 내용을 수정 (insert/update/delete) 하기 전, 해당 block을 반드시 buffer cache에 올리고, memory상에서 수정.
- 이 과정에서 동시에 redo log를 redo buffer에 Write.
- 수정된 (dirty) block은 DBWR (BLKW)가 나중에 필요에 따라 적는다.



● Background Processes

➤ 사용자의 요청을 직접 받아들이지는 않고, 워커스레드나 다른 배경 프로세스가 요청할 때, 혹은 정해진 주기에 따라 동작하며 주로 시간이 오래 걸리는 디스크 작업 담당

➤ 독립된 프로세스로서, 사용자의 요청과 비동기적으로 동작.

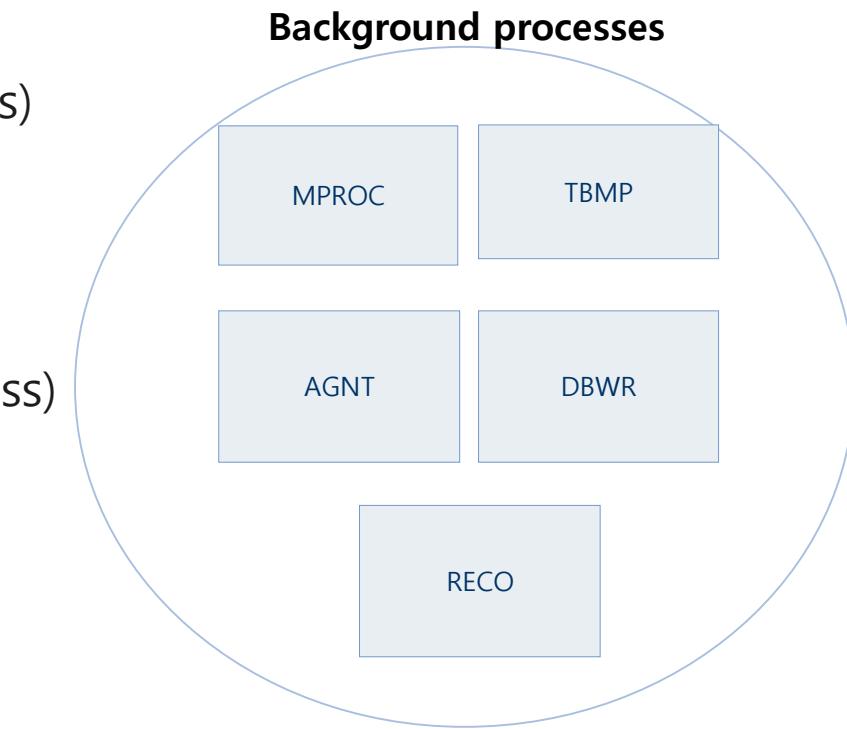
➤ 감시 프로세스(MPROC: monitor process)

➤ Tibero 매니저 프로세스(TBMP)

➤ 에이전트 프로세스(AGNT : agent process)

➤ 데이터베이스 쓰기 프로세스(DBWR)

➤ 복구 프로세스(RECO)



- 감시 프로세스(MPROC: monitor process)

- 하나의 독립된 프로세스
- Tibero 기동 시 최초로 생성되는 종료 시에도 마지막에 종료
- Tibero 기동 시, 리스너를 포함한 다른 프로세스를 생성, 주기적으로 각 프로세스 상태 점검
- 교착 상태 (Deadlock)도 검사

```
[tibero@T1:/tibero]$ ps -ef | egrep -e 'tbsvr|tblistener' | grep -v grep
tibero  6008  1  0 14:13 pts/1    00:00:05 tbsvr          -t NORMAL -SVR_SID tibero
tibero  6009  6008  0 14:13 pts/1  00:00:00 /tibero/cls/tibero6/bin/tblistener -n 11 -t NORMAL -SVR_SID tibero
tibero  6010  6008  0 14:13 pts/1  00:00:00 tbsvr_TBMP      -t NORMAL -SVR_SID tibero
tibero  6011  6008  0 14:13 pts/1  00:01:10 tbsvr_WP000     -t NORMAL -SVR_SID tibero
tibero  6012  6008  0 14:13 pts/1  00:00:08 tbsvr_WP001     -t NORMAL -SVR_SID tibero
tibero  6013  6008  0 14:13 pts/1  00:00:00 tbsvr_PEP000    -t NORMAL -SVR_SID tibero
tibero  6014  6008  0 14:13 pts/1  00:00:00 tbsvr_PEP001    -t NORMAL -SVR_SID tibero
tibero  6015  6008  0 14:13 pts/1  00:00:00 tbsvr_PEP002    -t NORMAL -SVR_SID tibero
tibero  6016  6008  0 14:13 pts/1  00:00:00 tbsvr_PEP003    -t NORMAL -SVR_SID tibero
tibero  6017  6008  0 14:13 pts/1  00:00:14 tbsvr_AGNT     -t NORMAL -SVR_SID tibero
tibero  6018  6008  0 14:13 pts/1  00:00:07 tbsvr_DBWR     -t NORMAL -SVR_SID tibero
tibero  6019  6008  0 14:13 pts/1  00:00:00 tbsvr_REC0     -t NORMAL -SVR_SID tibero
```

● Tibero 매니저 프로세스(TBMP)

- 시스템 관리 용도 프로세스
- 관리자의 접속 요청을 받아 이를 시스템 관리 용도로 예약된 워커 스레드에 접속을 할당
- 기본적으로 워커 프로세스와 동일한 역할을 수행하지만 **리스너를 거치지 않고, 스페셜 포트를 통해 직접 접속**.
- **SYS 계정만** 접속이 허용, LOCAL에서만 접속 가능

● 에이전트 프로세스(AGNT : agent process)

- 시스템 유지를 위해 주기적으로 처리해야 하는 Tibero 내부의 작업을 담당
- SEQW라는 명칭을 사용했으나 Tibero 6부터 AGNT로 명칭이 변경
- 다중 스레드(multi-threaded) 기반 구조로 동작하며, 서로 다른 용도의 업무를 스레드별로 나누어 수행

● 데이터베이스 쓰기 프로세스(DBWR)

- 데이터베이스에서 변경된 내용을 디스크에 기록하는 일과 연관된 스레드들이 모여 있는 프로세스
- 사용자가 변경한 블록을 디스크에 주기적으로 기록하는 스레드
- 리두 로그를 디스크에 기록하는 스레드
- 두 스레드를 통해 데이터베이스의 체크포인트 과정을 관할하는 체크포인트 스레드

● Recovery 프로세스(RECO)

- 복구 전용 프로세스
- Crash / Instance Recovery 수행 (기존 WT001에서 하던 작업)

Chapter 2장 설치 및 Directory.

- Tibero Windows 자동설치, 수동설치, Linux 자동 설치 부분은 Manual 참조
- Tibero 바이너리 설치

- 티베로에 접속하여 다음과 같이 티베로 압축 파일을 푼다.

```
$ cd /tibero  
$ tar -xvzf tibero_binary.tar.gz  
tibero6/bin/nbboot  
tibero6/bin/alterdd.sh  
.....
```

```
tibero6      → $TB_HOME  
|--- bin  
|--- client  
|--- config  
|--- scripts
```

```
$ cd $TB_HOME/config  
$ ./gen_tip.sh
```

● 초기 환경파일 생성

- \$TB_SID.tip 파일 생성 : Tibero 파라미터 파일
- tbdsn.tbr 파일 생성 : Tibero Client 접속 설정 파일
- psm_commands 파일 생성 : psm compile을 위한 command 파일 생성

● 라이센스 적용

- Tibero를 설치한 Hostname에 맞는 license.xml 파일을 발급(티베로 공급사의 엔지니어를 통해 제공)
- \$TB_HOME/license 디렉토리 생성 후 라이센스 파일을 이 디렉토리에 복사

- Tibero 파라미터 파일(\$TB_HOME/config/\$TB_SID.tip) 수정

```
DB_NAME=tibero
LISTENER_PORT=8629
CONTROL_FILES="/tibero/tbdata/c1.ctl","/tibero/tbdata/c2.ctl"
DB_CREATE_FILE_DEST=/tibero/tbdata
LOG_ARCHIVE_DEST=/tibero/tbarch
MAX_SESSION_COUNT=20
TOTAL_SHM_SIZE=600M
MEMORY_TARGET=1G
```

- Network 설정 파일(\$TB_HOME/client/config/tbdsn.tbr) 수정 가능

- Client가 Tibero에 접속 시 필요한 통신 환경을 설정하는 파일

```
tibero=(
    (INSTANCE=(HOST=localhost)          # SID name
     (PORT=8629)                      # 접속할 Tibero IP
     (DB_NAME=tibero)                 # 접속할 Tibero Port
   )
)
```

- Tibero 데이터베이스 생성

- Tibero를 nomount 모드로 부팅

```
$ tbboot nomount  
Listener port = 8629
```

Tibero 6

```
TmaxData Corporation Copyright (c) 2008-. All rights reserved.  
Tibero instance started up (NOMOUNT mode).
```

```
$ tbsql sys/tibero
```

tbSQL 6

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

Connected to Tibero.

```
SQL>
```

- Database 생성 스크립트

```
SQL> CREATE DATABASE
      USER sys IDENTIFIED BY tibero
      MAXINSTANCES 2
      MAXDATAFILES 50
      CHARACTER SET MSWIN949
      LOGFILE GROUP 1 '/tibero/cls/tbdata/tibero/log001' SIZE 10M,
              GROUP 2 '/tibero/cls/tbdata/tibero/log002' SIZE 10M,
              GROUP 3 '/tibero/cls/tbdata/tibero/log003' SIZE 10M
      MAXLOGGROUPS 10
      MAXLOGMEMBERS 2
      NOARCHIVELOG
              DATAFILE '/tibero/cls/tbdata/tibero/system01.dtf' SIZE 300m
              AUTOEXTEND ON NEXT 64M MAXSIZE 500m
      DEFAULT TEMPORARY TABLESPACE temp
              TEMPFILE '/tibero/cls/tbdata/tibero/temp01.dtf' SIZE 100m
              AUTOEXTEND ON NEXT 64M MAXSIZE 300m
      DEFAULT TABLESPACE usr
              DATAFILE '/tibero/cls/tbdata/tibero/usr01.dtf'    SIZE 50m
              AUTOEXTEND ON NEXT 64m MAXSIZE 200m
      UNDO TABLESPACE undo
              DATAFILE '/tibero/cls/tbdata/tibero/undo01.dtf' SIZE 300m
              AUTOEXTEND ON NEXT 64M MAXSIZE 300m
/

```

- Tibero 재기동

```
$ tbdown  
Tibero instance terminated (NORMAL mode).
```

```
$ tbboot  
Listener port = 8629
```

Tibero 6

```
TmaxData Corporation Copyright (c) 2008-. All rights reserved.  
Tibero instance started up (NORMAL mode).
```

- Data Dictionary 및 system 패키지 생성

```
$ cd $TB_HOME/scripts  
$ sh system.sh
```

Enter SYS password:

Enter SYSCAT password:

Dropping agent table...

Creating text packages table ...

Creating the role DBA...

Create default system users & roles?(Y/N):

Skip creating default system users & roles

Creating virtual tables(1)...

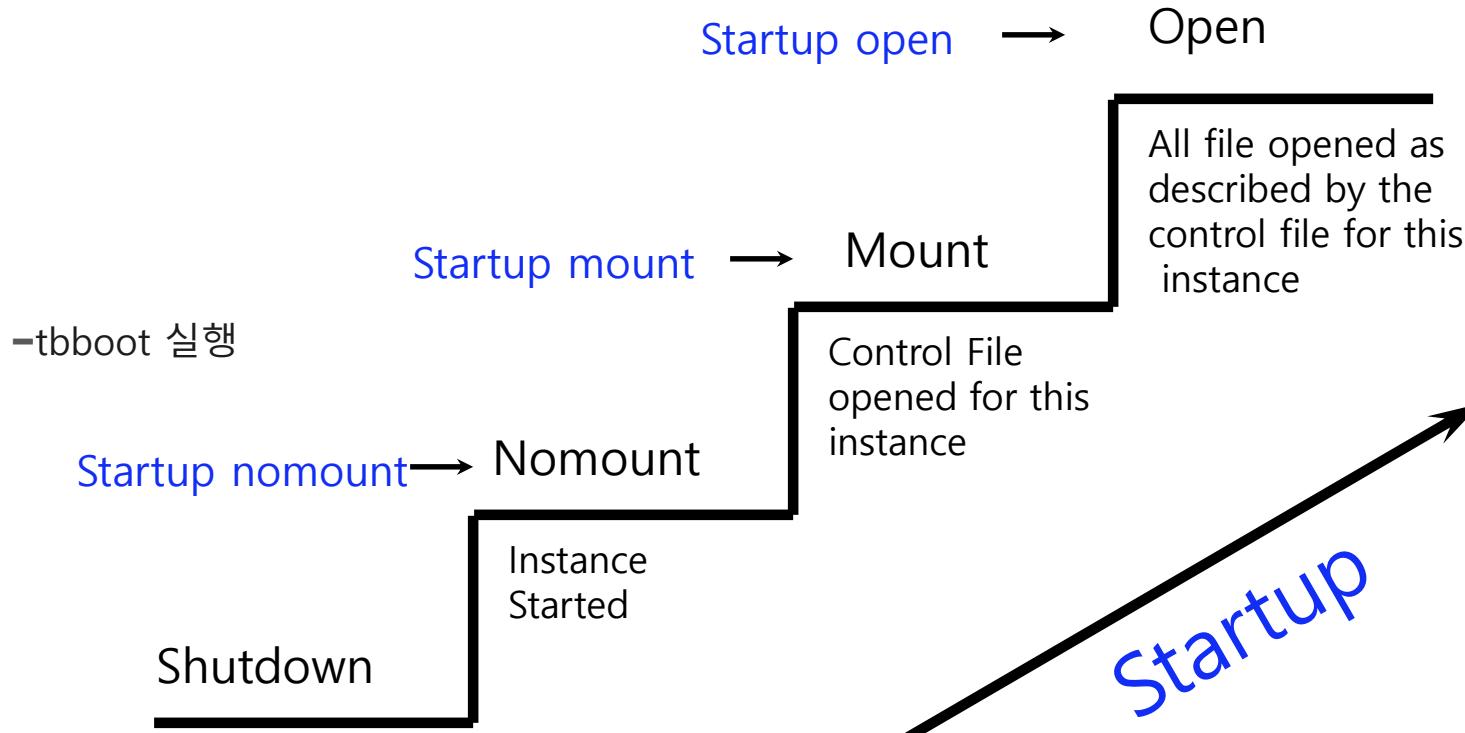
❖ 주의사항

~ **sh system.sh** 실행시 2회의 패스워드 입력을 받으며,
다음과 같이 입력해야 함.

- ✓ 첫번째 패스워드 입력 → tibero
- ✓ 두번째 패스워드 입력 → syscat

- Tibero 기동

- Start Process



● Tibero 기동

- tbboot 사용법

```
$ tbboot -h
```

Usage: tbboot [-h] [-v] [-l] [-C] [-t BOOTMODE]

-h: show this help.

-v: show RDBMS version.

-l: show license information.

-C: show available character set list.

-c: No replication mode.

BOOTMODE: one of NOMOUNT MOUNT RECOVERY NORMAL RESETLOGS ALTERDD READONLY FAILOVER

If no bootmode is set, default bootmode is 'NORMAL'.

```
$ tbboot
```

Listener port = 8629

Tibero 6

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

Tibero instance started up (NORMAL mode).

● Tibero 종료

- ttdown은 현재 동작 중인 Tibero를 종료하는 역할을 수행한다.

```
$ ttdown -h
```

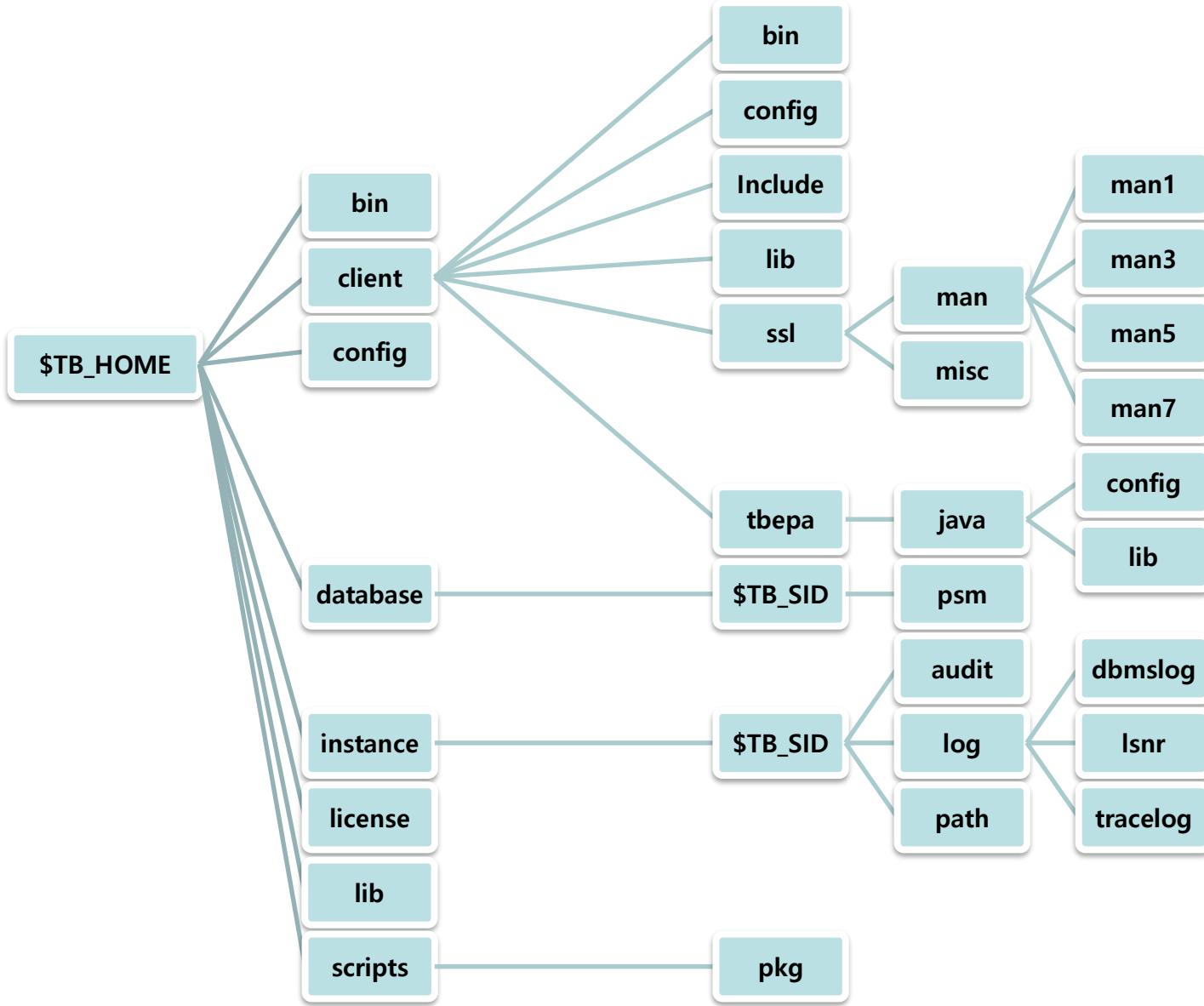
Usage: ttdown [-h] [-t DOWNMODE]

DOWNMODE : NORMAL, POST_TX, IMMEDIATE, ABORT, SWITCHOVER, ABNORMAL

옵션	설명										
	옵션이 없는 경우 Tibero를 정상 모드로 종료하는 옵션이다.										
-h	ttdown 을 사용하기 위한 간단한 도움말을 보여주는 옵션이다.										
-t	Tibero 서버를 종료할 수 있는 옵션이다. 이 옵션은 생략이 가능하다. <table border="1"><thead><tr><th>다운 모드</th><th>설명</th></tr></thead><tbody><tr><td>NORMAL</td><td>일반적인 종료 모드이다.</td></tr><tr><td>POST_TX</td><td>모든 트랜잭션이 끝날 때까지 기다리고 나서 Tibero 를 종료하는 모드이다.</td></tr><tr><td>IMMEDIATE</td><td>현재 수행 중인 모든 작업을 강제로 중단시키며, 진행중인 트랜잭션을 롤백하고 Tibero 를 종료하는 모드이다.</td></tr><tr><td>ABORT</td><td>Tibero 의 프로세스를 강제로 종료하는 모드이다.</td></tr></tbody></table>	다운 모드	설명	NORMAL	일반적인 종료 모드이다.	POST_TX	모든 트랜잭션이 끝날 때까지 기다리고 나서 Tibero 를 종료하는 모드이다.	IMMEDIATE	현재 수행 중인 모든 작업을 강제로 중단시키며, 진행중인 트랜잭션을 롤백하고 Tibero 를 종료하는 모드이다.	ABORT	Tibero 의 프로세스를 강제로 종료하는 모드이다.
다운 모드	설명										
NORMAL	일반적인 종료 모드이다.										
POST_TX	모든 트랜잭션이 끝날 때까지 기다리고 나서 Tibero 를 종료하는 모드이다.										
IMMEDIATE	현재 수행 중인 모든 작업을 강제로 중단시키며, 진행중인 트랜잭션을 롤백하고 Tibero 를 종료하는 모드이다.										
ABORT	Tibero 의 프로세스를 강제로 종료하는 모드이다.										

Directory

CHAPTER 2장. 설치 및 Directory.



● Directory 구성

구분	Data Type															
bin	<p>Tibero의 실행 파일과 서버 관리를 위한 유ти리티가 위치한 디렉터리이다. 이 디렉터리에 속한 파일 중에서 tbsvr과 tblistener는 Tibero를 구성하는 실행 파일이며, tbboot와 ttdown은 각각 Tibero를 기동하고 종료하는 역할을 담당한다.</p> <p>tbsvr과 tblistener 실행 파일은 반드시 tbboot 명령어를 이용하여 실행되어야 하며, 절대로 직접 실행해서는 안 된다.</p>															
client/bin	<table border="1"><thead><tr><th>유ти리티</th><th>설명</th></tr></thead><tbody><tr><td>tbSQL</td><td>기본적인 클라이언트 프로그램으로 사용자가 직접 SQL 질의를 하고 그 결과를 확인할 수 있는 유ти리티이다.</td></tr><tr><td>T-Up</td><td>다른 데이터베이스의 내용을 Tibero의 데이터베이스로 옮기는 것을 지원하는 유ти리티이다.</td></tr><tr><td>tbExport</td><td>논리적 백업이나 데이터베이스 간에 데이터 이동을 위해 데이터베이스의 내용을 외부 파일로 저장하는 유ти리티이다.</td></tr><tr><td>tblImport</td><td>외부 파일에 저장된 내용을 데이터베이스로 가져오는 유ти리티이다.</td></tr><tr><td>tbLoader</td><td>대량의 데이터를 데이터베이스로 한꺼번에 읽어 들이는 유ти리티이다.</td></tr><tr><td>tbpc</td><td>C 언어로 작성된 프로그램 안에서 내장 SQL(Embedded SQL)을 사용하는 프로그램을 개발할 때 이를 C 프로그램으로 변환하는 유ти리티이다. 이렇게 변환된 프로그램을 C 컴파일러를 통해 컴파일할 수 있도록 도와주는 역할도 담당한다.</td></tr></tbody></table>		유ти리티	설명	tbSQL	기본적인 클라이언트 프로그램으로 사용자가 직접 SQL 질의를 하고 그 결과를 확인할 수 있는 유ти리티이다.	T-Up	다른 데이터베이스의 내용을 Tibero의 데이터베이스로 옮기는 것을 지원하는 유ти리티이다.	tbExport	논리적 백업이나 데이터베이스 간에 데이터 이동을 위해 데이터베이스의 내용을 외부 파일로 저장하는 유ти리티이다.	tblImport	외부 파일에 저장된 내용을 데이터베이스로 가져오는 유ти리티이다.	tbLoader	대량의 데이터를 데이터베이스로 한꺼번에 읽어 들이는 유ти리티이다.	tbpc	C 언어로 작성된 프로그램 안에서 내장 SQL(Embedded SQL)을 사용하는 프로그램을 개발할 때 이를 C 프로그램으로 변환하는 유ти리티이다. 이렇게 변환된 프로그램을 C 컴파일러를 통해 컴파일할 수 있도록 도와주는 역할도 담당한다.
유ти리티	설명															
tbSQL	기본적인 클라이언트 프로그램으로 사용자가 직접 SQL 질의를 하고 그 결과를 확인할 수 있는 유ти리티이다.															
T-Up	다른 데이터베이스의 내용을 Tibero의 데이터베이스로 옮기는 것을 지원하는 유ти리티이다.															
tbExport	논리적 백업이나 데이터베이스 간에 데이터 이동을 위해 데이터베이스의 내용을 외부 파일로 저장하는 유ти리티이다.															
tblImport	외부 파일에 저장된 내용을 데이터베이스로 가져오는 유ти리티이다.															
tbLoader	대량의 데이터를 데이터베이스로 한꺼번에 읽어 들이는 유ти리티이다.															
tbpc	C 언어로 작성된 프로그램 안에서 내장 SQL(Embedded SQL)을 사용하는 프로그램을 개발할 때 이를 C 프로그램으로 변환하는 유ти리티이다. 이렇게 변환된 프로그램을 C 컴파일러를 통해 컴파일할 수 있도록 도와주는 역할도 담당한다.															

● Directory 구성

구분	Data Type								
client/config	Tibero의 클라이언트 프로그램을 실행하기 위한 설정 파일이 위치하는 디렉터리이다.								
client/include	Tibero의 클라이언트 프로그램을 작성할 때 필요한 헤더 파일이 위치하는 디렉터리이다.								
client/lib	Tibero의 클라이언트 프로그램을 작성할 때 필요한 라이브러리 파일이 위치하는 디렉터리이다.								
client/ssl	서버 보안을 위한 인증서와 개인 키를 저장하는 디렉터리이다.								
client/tbepa	External Procedure와 관련된 설정 파일과 로그 파일이 있는 디렉터리이다.								
config	Tibero의 환경설정 파일이 위치하는 디렉터리이다. 이 위치에 존재하는 \$TB_SID.tip 파일이 Tibero의 환경설정을 결정한다.								
database/\$TB_SID	Tibero의 데이터베이스 정보를 별도로 설정하지 않는 한, 모든 데이터베이스 정보가 이 디렉터리와 그 하위 디렉터리에 저장된다. (이 디렉터리에는 데이터 자체에 대한 메타데이터 뿐만 아니라 다음과 같은 종류의 파일이 있다.) <table border="1"><thead><tr><th>파일</th><th>설명</th></tr></thead><tbody><tr><td>컨트롤 파일</td><td>다른 모든 파일의 위치를 담고 있는 파일이다.</td></tr><tr><td>데이터 파일</td><td>실제 데이터를 저장하고 있는 파일이다.</td></tr><tr><td>로그 파일</td><td>데이터 복구를 위해 데이터에 대한 모든 변경 사항을 저장하는 파일이다.</td></tr></tbody></table>	파일	설명	컨트롤 파일	다른 모든 파일의 위치를 담고 있는 파일이다.	데이터 파일	실제 데이터를 저장하고 있는 파일이다.	로그 파일	데이터 복구를 위해 데이터에 대한 모든 변경 사항을 저장하는 파일이다.
파일	설명								
컨트롤 파일	다른 모든 파일의 위치를 담고 있는 파일이다.								
데이터 파일	실제 데이터를 저장하고 있는 파일이다.								
로그 파일	데이터 복구를 위해 데이터에 대한 모든 변경 사항을 저장하는 파일이다.								

● Directory 구성

구분	Data Type											
database/\$TB_SID/ /psm	tbPSM 프로그램을 컴파일드 모드(Compiled mode)로 컴파일하는 경우, 컴파일된 파일이 저장되는 디렉터리이다. 하지만, 현재 Tibero에서는 인터프리터 모드만을 지원하고 있다. 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.											
instance/\$TB_SID/ audit	데이터베이스 사용자가 시스템 특권 또는 스키마 객체 특권을 사용하는 것을 감시(AUDIT)한 내용을 기록한 파일이 저장되는 디렉터리이다.											
instance/\$TB_SID/ log	Tibero의 로그 파일이 저장되는 디렉터리이다. <table border="1"><thead><tr><th>파일</th><th>설명</th></tr></thead><tbody><tr><td>트레이스 로그 파일 (Tibero6 FS06 버전부터 slog)</td><td>디버깅을 위한 파일이다. 서버가 하는 모든 일이 자세하게 기록되는 파일이며, 서버 성능이 저하되는 원인을 찾거나 Tibero 자체의 버그를 해결하는 데 사용될 수 있다.</td></tr><tr><td>DBMS 로그 파일 (Tibero6 FS06 버전부터 dlog)</td><td>트레이스 로그 파일에 기록되는 정보보다 좀 더 중요한 정보가 기록되는 파일이며, 서버 기동 및 종류, DDL 문장의 수행 등이 기록되는 파일이다.</td></tr><tr><td>EVENT 로그 파일 (Tibero6 FS06 버전부터 ilog)</td><td>설정된 event에 대한 트레이스 로그가 기록되는 파일이며, event 로그를 보려면 tbev를 이용해야한다.</td></tr><tr><td>LISTENER 로그 파일</td><td>Listener의 디버깅을 위한 파일이다. 리스너에서 일어난 중요한 일이 기록되는 파일이며, 리스너의 버그를 해결하는 데 사용될 수 있다.</td></tr></tbody></table>		파일	설명	트레이스 로그 파일 (Tibero6 FS06 버전부터 slog)	디버깅을 위한 파일이다. 서버가 하는 모든 일이 자세하게 기록되는 파일이며, 서버 성능이 저하되는 원인을 찾거나 Tibero 자체의 버그를 해결하는 데 사용될 수 있다.	DBMS 로그 파일 (Tibero6 FS06 버전부터 dlog)	트레이스 로그 파일에 기록되는 정보보다 좀 더 중요한 정보가 기록되는 파일이며, 서버 기동 및 종류, DDL 문장의 수행 등이 기록되는 파일이다.	EVENT 로그 파일 (Tibero6 FS06 버전부터 ilog)	설정된 event에 대한 트레이스 로그가 기록되는 파일이며, event 로그를 보려면 tbev를 이용해야한다.	LISTENER 로그 파일	Listener의 디버깅을 위한 파일이다. 리스너에서 일어난 중요한 일이 기록되는 파일이며, 리스너의 버그를 해결하는 데 사용될 수 있다.
파일	설명											
트레이스 로그 파일 (Tibero6 FS06 버전부터 slog)	디버깅을 위한 파일이다. 서버가 하는 모든 일이 자세하게 기록되는 파일이며, 서버 성능이 저하되는 원인을 찾거나 Tibero 자체의 버그를 해결하는 데 사용될 수 있다.											
DBMS 로그 파일 (Tibero6 FS06 버전부터 dlog)	트레이스 로그 파일에 기록되는 정보보다 좀 더 중요한 정보가 기록되는 파일이며, 서버 기동 및 종류, DDL 문장의 수행 등이 기록되는 파일이다.											
EVENT 로그 파일 (Tibero6 FS06 버전부터 ilog)	설정된 event에 대한 트레이스 로그가 기록되는 파일이며, event 로그를 보려면 tbev를 이용해야한다.											
LISTENER 로그 파일	Listener의 디버깅을 위한 파일이다. 리스너에서 일어난 중요한 일이 기록되는 파일이며, 리스너의 버그를 해결하는 데 사용될 수 있다.											
	트레이스 로그 파일과 DBMS 로그 파일은 데이터베이스를 사용할수록 계속 누적되어 저장된다. 또한, 전체 디렉터리의 최대 크기를 지정할 수 있으며, Tibero는 그 지정된 크기를 넘어가지 않도록 오래된 파일을 삭제한다.											

● Directory 구성

구분	Data Type	
instance/\$TB_SID/log	파라미터	설명
	LOG_DEFAULT_DEST	로그 생성 기본 경로
	DBMS_LOG_FILE_SIZE	DBMS 로그 파일 하나의 최대 크기를 설정한다.
	DBMS_LOG_TOTAL_SIZE_LIMIT	DBMS 로그 파일이 저장된 디렉터리의 최대 크기를 설정한다.
	TRACE_LOG_FILE_SIZE	트레이스 로그 파일 하나의 최대 크기를 설정한다.
	TRACE_LOG_TOTAL_SIZE_LIMIT	트레이스 로그 파일이 저장된 디렉터리의 최대 크기를 설정한다.
	EVENT_TRACE_FILE_SIZE	Event 로그 파일 하나의 최대 크기를 설정한다.
	EVENT_TRACE_TOTAL_SIZE_LIMIT	Event 로그 파일이 저장된 디렉터리의 최대 크기를 설정한다.
	LSNR_LOG_FILE_SIZE	Listener 로그 파일 하나의 최대 크기를 설정한다.
	LSNR_LOG_TOTAL_SIZE_LIMIT	Listener 로그 파일이 저장된 디렉터리의 최대 크기를 설정한다.
instance/\$TB_SID/path	Tibero 서버에서 Spatial과 관련된 함수를 사용하기 위한 라이브러리 파일이 있는 디렉터리이다.	
license	Tibero의 라이선스 파일(<i>license.xml</i>)이 있는 디렉터리이다. XML 형식이므로 일반 텍스트 편집기로도 라이선스의 내용을 확인할 수 있다.	
scripts	Tibero의 데이터베이스를 생성할 때 사용하는 각종 SQL 문장이 있는 디렉터리이다. 또한, Tibero의 현재 상태를 보여주는 각종 뷰의 정의도 이 디렉터리에 있다.	
scripts/pkg	Tibero에서 사용하는 패키지의 생성문이 저장되는 디렉터리이다.	

Chapter 3장 tbSQL

● tbSQL은 Tibero에서 제공하는 SQL 문장을 처리하는 대화형 유ти리티.

● 기능

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

● 문법

```
tbsql [[options] [userpass] [start]]
```

-options

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

-userpass

username[/password[@connect_string]]

옵션	설명
username	사용자명으로, 대소문자를 구분하지 않는다. 단, 큰따옴표("")에 사용자명을 입력하는 경우는 예외이다.
password	패스워드로, 대소문자를 구분하여 입력한다.
connect_string	데이터베이스에 대한 접속 정보를 가진 DSN(Data Source Name)이다.

-start

@filename[.ext]

옵션	설명
filename	파일명이다.
ext	파일의 확장자로, 지정하지 않을 경우 FILEEXT 시스템 변수에 지정된 확장자가 디폴트 값이다.

● 사용자명

- 대소문자 구분하지 않는다. 단 큰따옴표("")에 사용시 예외

● 패스워드

- 대소문자 구분

```
$ tbsql sys/tibero
```

```
tbSQL 6
```

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

```
Connected to Tibero.
```

```
SQL>
```

- **tbSQL 유ти리티가 정상적으로 실행되면 SQL 프롬프트가 출력된다.**

- SQL 프롬프트에서 SQL 문장, tbPSM 프로그램, tbSQL 유ти리티의 명령어를 입력할 수 있다.

- **여러 라인에 걸쳐 입력할 수 있다.**

- SQL 문장과 tbPSM 프로그램은 입력과 실행을 분리할 수 있다. 하지만, tbSQL 유ти리티의 명령어는 입력과 동시에 실행된다.

- **대소문자를 구분하지 않는다.**

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

● 문법

```
SET {시스템 변수} {시스템 변수의 설정 값}
```

● 예제

```
SQL> SET AUTOCOMMIT ON
```

- tbSQL 유ти리티를 종료하려면 SQL 프롬프트 상에서 EXIT 또는 QUIT 명령어를 입력해야 한다.

- 예제

```
SQL> EXIT
```

- tbSQL 유ти리티의 시스템 변수에 설정할 값은 SET 명령어로 설정하고, SHOW 명령어로 출력한다.

시스템 변수	Default Value	설명
AUTOCOMMIT	OFF	자동 커밋 여부를 설정하는 시스템 변수이다.
AUTOTRACE	OFF	수행 중인 질의의 Plan이나 통계 정보를 출력할지를 설정하는 시스템 변수이다.
COMMENT	ON	
BLOCKTERMINATOR	마침표(.)	tbPSM 문장에서 입력의 마지막을 나타내는 문자를 설정하는 시스템 변수이다.
DDLSTATS	OFF	DDL 문장의 Plan이나 통계 정보를 보여줄지를 설정하는 시스템 변수이다.
DEFINE	OFF	& 문자를 치환 변수로 인식할지를 설정하는 시스템 변수이다.
EDITFILE	.tbedit.sql	EDIT 명령어에서 사용하는 파일 이름의 디폴트 값을 설정하는 시스템 변수이다.
ESCAPE	역슬래시(₩)	이스케이프 문자를 설정하는 시스템 변수이다.
FEEDBACK	ON	SQL 문장의 수행 결과를 화면에 출력할지를 설정하는 시스템 변수이다.
FILEEXT	sql	파일 확장자의 디폴트 값을 설정하는 시스템 변수이다.
FILEPATH	현재 디렉토리	파일 경로의 디폴트 값을 설정하는 시스템 변수이다.

시스템 변수	Default Value	설명
HISTORY	50	명령어 히스토리의 크기를 설정하는 시스템 변수이다.
LINESIZE	80	한 라인에 출력할 문자 수를 설정하는 시스템 변수이다.
LONG	80	CLOB 타입의 데이터를 표시하기 위해 사용할 문자 수를 설정하는 시스템 변수이다.
NOESCAPE	OFF	
NUMWIDTH	10	숫자형 데이터의 기본 출력 길이를 설정하는 시스템 변수이다.
OUTPUTSIZE	1024	DBMS_OUTPUT 패키지에서 출력하는 결과의 크기를 설정하는 시스템 변수이다.
PAGESIZE	24	한 화면에 출력할 라인 수를 설정하는 시스템 변수이다.
PRINTSTMT	OFF	현재 수행 중인 SQL 또는 tbPSM 문장의 출력 여부를 설정하는 시스템 변수이다.
PROFILE	OFF	SQL 문장을 수행할 때 걸린 시간과 사용 중인 메모리의 출력 여부를 설정하는 시스템 변수이다.
PROMPT	SQL>	화면상의 프롬프트 문자를 설정하는 시스템 변수이다.
SERVEROUTPUT	OFF	DBMS_OUTPUT 패키지의 결과를 화면에 출력할 것인지를 설정하는 시스템 변수이다.
SQLTERMINATOR	세미 콜론(:)	SQL 문장을 종료하는 문자를 설정하는 시스템 변수이다.

시스템 변수	Default Value	설명
TERMOUT	ON	현재 시간을 화면에 출력할 것인지를 설정하는 시스템 변수이다.
TIME	OFF	현재 시간을 화면에 출력할 것인지를 설정하는 시스템 변수이다.
TIMING	OFF	SQL, tbSQL 문장의 결과를 출력할 때마다 수행 시간을 출력할 것인지를 설정하는 시스템 변수이다.
TRIMOUT	ON	Determines if trailing spaces are trimmed from lines displayed on the screen.
TRIMSPOOL	OFF	Determines if trailing spaces are trimmed from lines spooled to a file.
WRAP	ON	출력할 라인이 긴 경우, 나머지를 다음 라인에 출력할 것인지를 설정하는 시스템 변수이다.

● 명령어의 입력

- SQL 문장의 입력

- 일반적인 입력
- 줄 바꿈
- 주석(comment)의 삽입
- 이전에 저장된 문장을 이용하여 입력

- tbPSM 프로그램의 입력

- 일반적인 입력
- 이전에 저장된 문장을 이용하여 입력
- 주석의 삽입

- tbSQL 유틸리티의 명령어 입력

● 명령어의 실행

- SQL 버퍼에 저장된 SQL 문장이나 tbPSM 프로그램의 실행
 - RUN 또는 / 명령어
- SQL 문장의 실행
 - 전체 문장을 입력하고 세미콜론(;)으로 종료하면 SQL 문장이 바로 실행된다.
- SQL 버퍼에 저장과 동시에 실행

● 기타 기능

- 주석의 삽입

- /* ... */ 를 이용
- 두개의 마이너스 부호(--)를 이용

- 자동 커밋

- SET AUTOCOMMIT 명령어 사용 설정 / SHOW AUTOCOMMIT 명령어 사용 확인

- 운영체제 명령어의 실행

- HOST 명령어 또는 ! 명령어 사용

- 출력 내용의 저장

- SPOOL 명령어를 사용 / 종료는 SPOOL OFF 명령어 사용

●스크립트 기능

-스크립트란 ?

- 한 번의 명령으로 일괄적으로 작업을 실행하기 위한 SQL 문장과 tbPSM 프로그램, tbSQL 유틸리티의 명령어의 모임이다.

고급기능 (2/4)

-스크립트의 생성

```
$ export TB_EDITOR=vi  
$ tbsql tibero/tmax
```

```
SQL> EDIT example
```

```
SET ECHO ON  
  
-- SQL  
CREATE TABLE EMP(  
    ENAME  VARCHAR(10)  
    SALARY NUMBER(10,2)  
    ADDR   VARCHAR(10)  
    DEPTNO NUMBER(2)  
);  
  
INSERT INTO EMP VALUES  
    ('JUNG',100, 'SEOUL', 1);  
  
INSERT INTO EMP VALUES  
    ('HONG',150, 'PARIS', 5);  
  
INSERT INTO EMP VALUES  
    ('PARK',200, 'HONGKONG', 20);
```

```
SELECT * FROM EMP;  
  
UPDATE EMP SET SALARY = SALARY * 1.05  
    WHERE DEPTNO = 5;  
  
SELECT ENAME, SALARY, ADDR FROM EMP  
    WHERE DEPTNO = 5;  
  
-- tbPSM  
DECLARE  
    deptno NUMBER(2);  
BEGIN  
    deptno := 20;  
  
    UPDATE EMP SET SALARY = SALARY * 1.05  
        WHERE DEPTNO = deptno;  
END;  
/  
  
SELECT * FROM EMP;  
  
COMMIT;  
  
DROP TABLE EMP;
```

-스크립트의 실행

```
SQL> START example  
SQL> @example
```

```
$ tbsql tibero/tmax @example
```

● DBA를 위한 기능

- 접속 방법

```
$ tbsql sys/tibero
```

```
SQL> CONNECT sys/tibero
```

● 문법

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

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.
별표(*)	별표(*)로 표시된 내용은 포함되지 않을 수도 있고, 여러 번 포함될 수도 있다. 위의 예에서 arg는 대괄호([]) 바로 뒤에 별표(*)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.

명령어	설명
!	운영체제의 명령어를 실행하는 명령어이다. HOST 명령어와 동일하다.
@	스크립트를 실행하는 명령어이다. START 명령어와 동일하다.
/	현재 SQL 버퍼 내의 SQL 문장 또는 tbPSM 프로그램을 실행하는 명령어이다. RUN 명령어와 동일하다.
ACCEPT	사용자의 입력을 받아 치환 변수의 속성을 설정하는 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.
DEL	SQL 버퍼에 저장된 라인을 지우는 명령어이다.
DESCRIBE	지정된 객체의 컬럼 정보를 출력하는 명령어이다.
DISCONNECT	현재 데이터베이스로부터 접속을 해제하는 명령어이다.
EDIT	특정 파일 또는 SQL 버퍼의 내용을 외부 편집기를 이용하여 편집하는 명령어이다.

명령어	설명
EXECUTE	단일 tbPSM 문장을 수행하는 명령어이다.
EXIT	tbSQL 유ти리티를 종료하는 명령어이다. QUIT 명령어와 동일하다.
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.
LS	현재 사용자가 생성한 데이터베이스 객체를 출력하는 명령어이다.
PAUSE	사용자가 <Enter> 키를 누를 때까지 실행을 멈추는 명령어이다.
PRINT	사용자가 정의한 바인드 변수의 값을 출력하는 명령어이다.
PROMPT	사용자가 정의한 SQL 문장이나 빈 라인을 그대로 화면에 출력하는 명령어이다.

명령어	설명
QUIT	tbSQL 유ти리티를 종료하는 명령어이다. EXIT 명령어와 동일하다.
RUN	현재 SQL 버퍼 내의 SQL 문장이나 tbPSM 프로그램을 실행하는 명령어이다. / 명령어와 동일하다.
SET	tbSQL 유ти리티의 시스템 변수를 설정하는 명령어이다.
SHOW	tbSQL 유ти리티의 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.

● 문자형

- 문법

```
COL[UMN] {name} FORMAT A{n}
```

- 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>
```

● 숫자형

- 문법

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

- 예제

```
SQL> COLUMN x FORMAT 999,999
SQL> SELECT 123456 x FROM DUAL;
X
-----
123,456
1 row selected.
```

Format	예시	설명
,	9,999	주어진 위치에 쉼표 (,)를 출력한다.
.	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	정수부분의 주어진 위치에 그룹 분리자를 출력한다.
L	L9999	주어진 위치에 local currency symbol을 출력한다.
MI	9999MI	음수 뒤에 마이너스 기호를 출력하고, 양수 뒤에 공백을 출력한다.
PR	9999PR	음수인 경우에 <와 >로 감싸서 출력하고, 양수인 경우에 양쪽에 공백을 출력한다.

컬럼 포맷 (4/4)

CHAPTER 3장. tbSQL

Format	예시	설명
RN	RN	대문자로 출력한다.
rn	rn	소문자로 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	10 ⁿ 만큼 곱한 값을 출력한다. 여기서 n은 V뒤에 오는 9의 개수이다.
X	XXXX, xxxx	16진수 형태로 출력한다.

Chapter 4장

tbAdmin

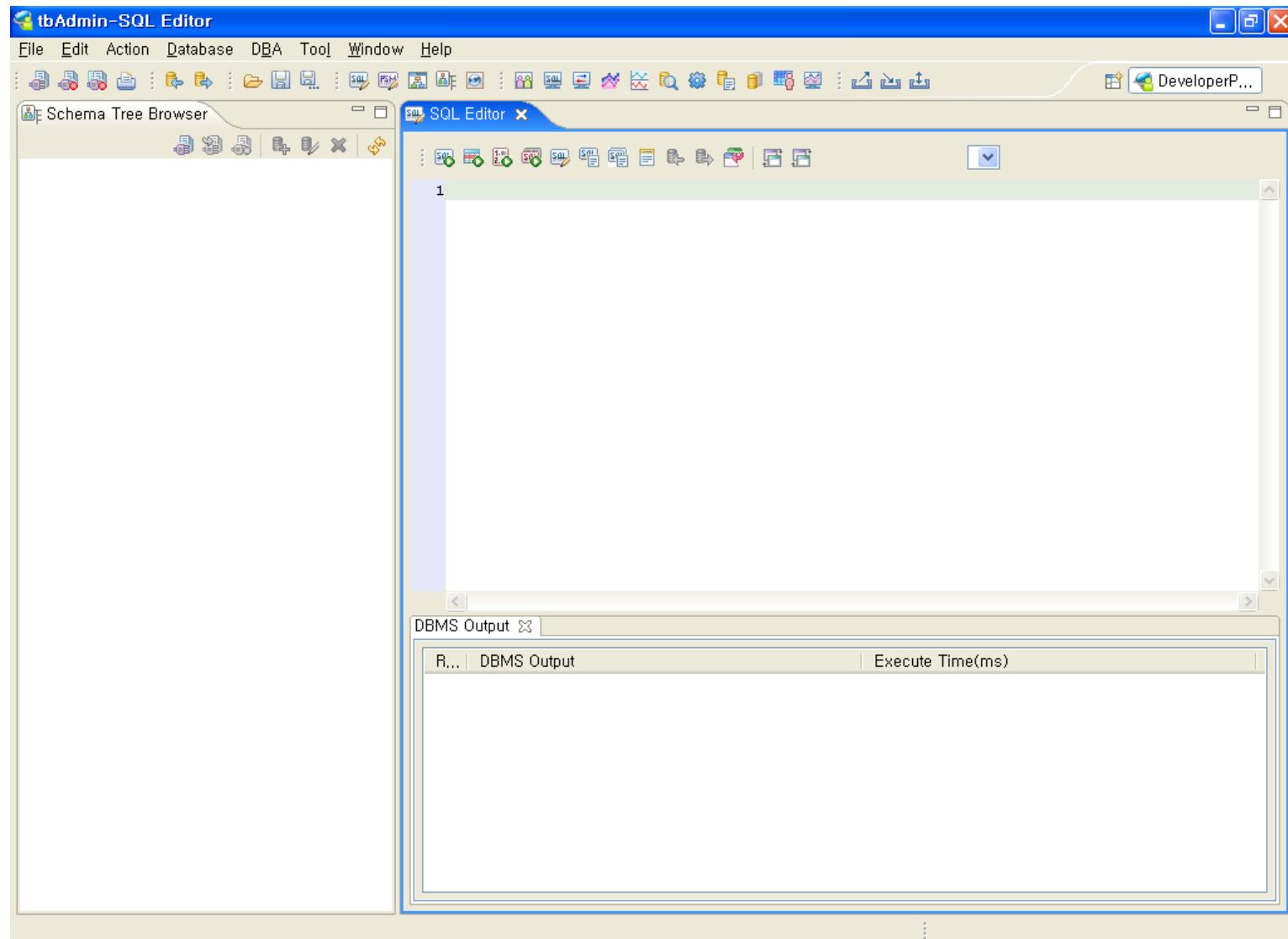
- tbAdmin은 Tibero에서 제공하는 Graphic User Interface(이하 GUI)기반 SQL/PSM 처리 응용프로그램이다.

- 기능

- 일반적인 SQL 문장 및 tbPSM 프로그램을 입력, 편집, 저장, 실행
- 트랜잭션 설정 및 종료
- 다이얼로그를 통한 Schema Object 관리
- DBA에 의한 데이터베이스 관리
- 외부 유ти리티 및 프로그램의 실행(tbExplmp 실행)
- tbAdmin 환경 설정

OverView (2/2)

CHAPTER 4장. tbAdmin



● 설치 및 실행

- JRE 1.5(Java Runtime Environment)가 설치 되어 있는 머신이라면 tbAdmin.zip 파일의 압축을 해제하는 것으로 설치 완료.

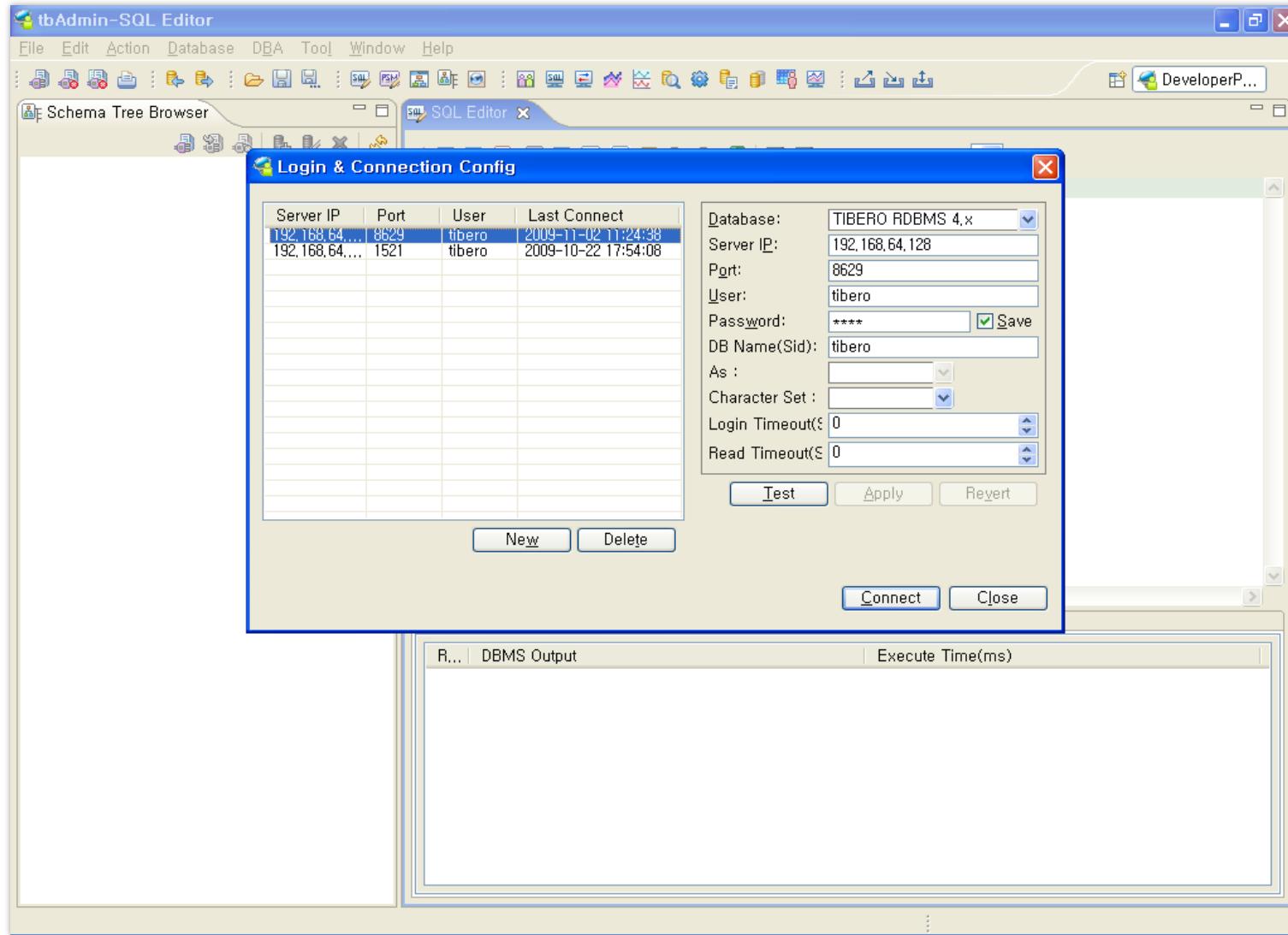
● 설정

- 상위 메뉴중 Window > Preferences 선택

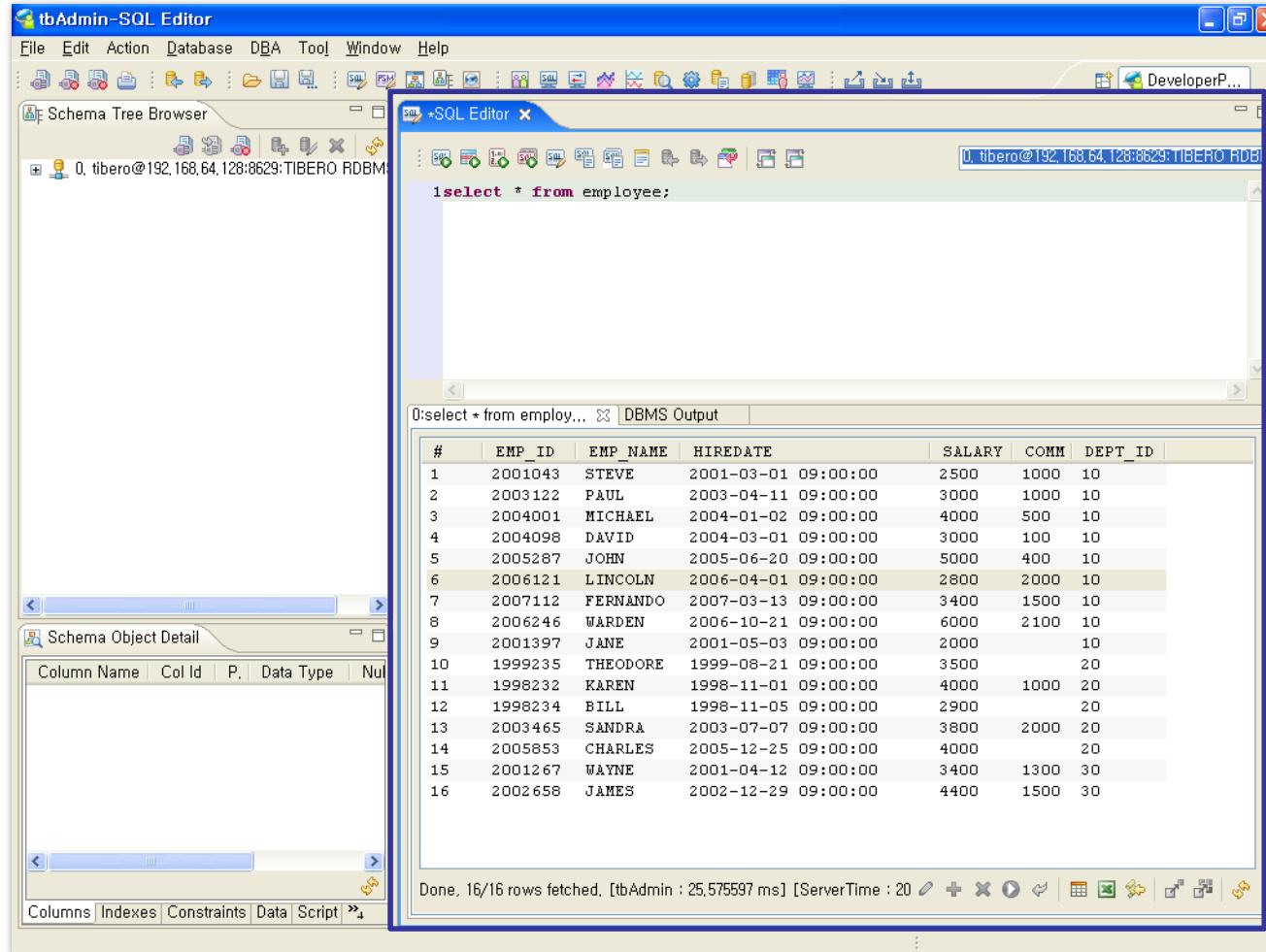
● 실행

- 윈도우 – tbAdmin.exe
- 리눅스 – tbAdmin.sh

● Login & Connection



●SQL Editor



●SQL Editor

The screenshot shows the tbAdmin-PSM Editor interface with the following details:

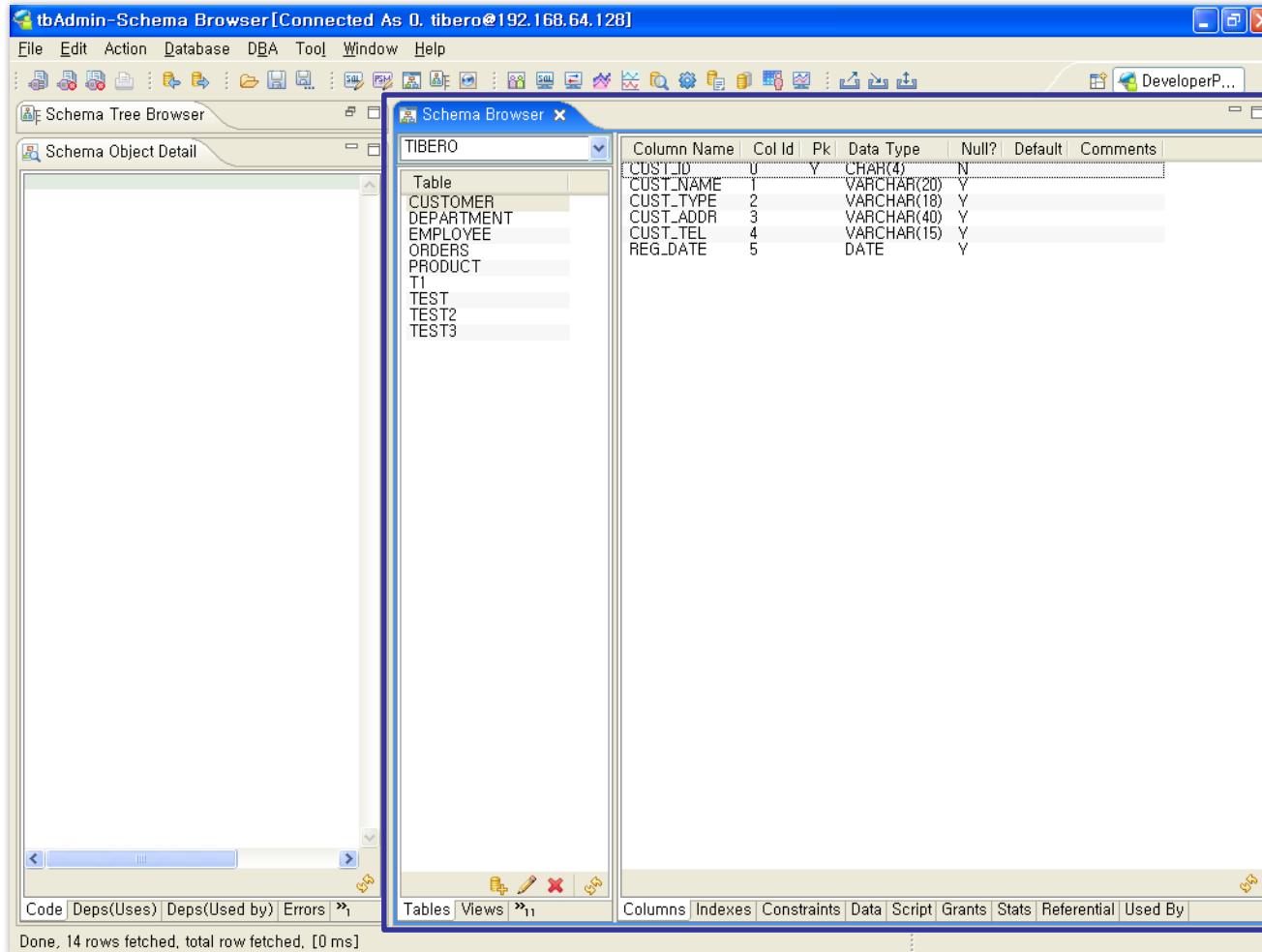
- File Menu:** File, Edit, Action, Database, DBA, Tool, Window, Help.
- Toolbars:** Standard toolbar with icons for Open, Save, Print, etc.
- Schema Tree Browser:** Displays the database schema with PUBLIC, SYS, SYSCAT, and TIBERO nodes. Under TIBERO, there are Table (9), View (0), Synonym (0), and PSM (3) sections. The PSM section contains a Procedure (1) named P_1.
- PSM Editor:** The main editor window displays the following PL/SQL code for Procedure P_1:


```

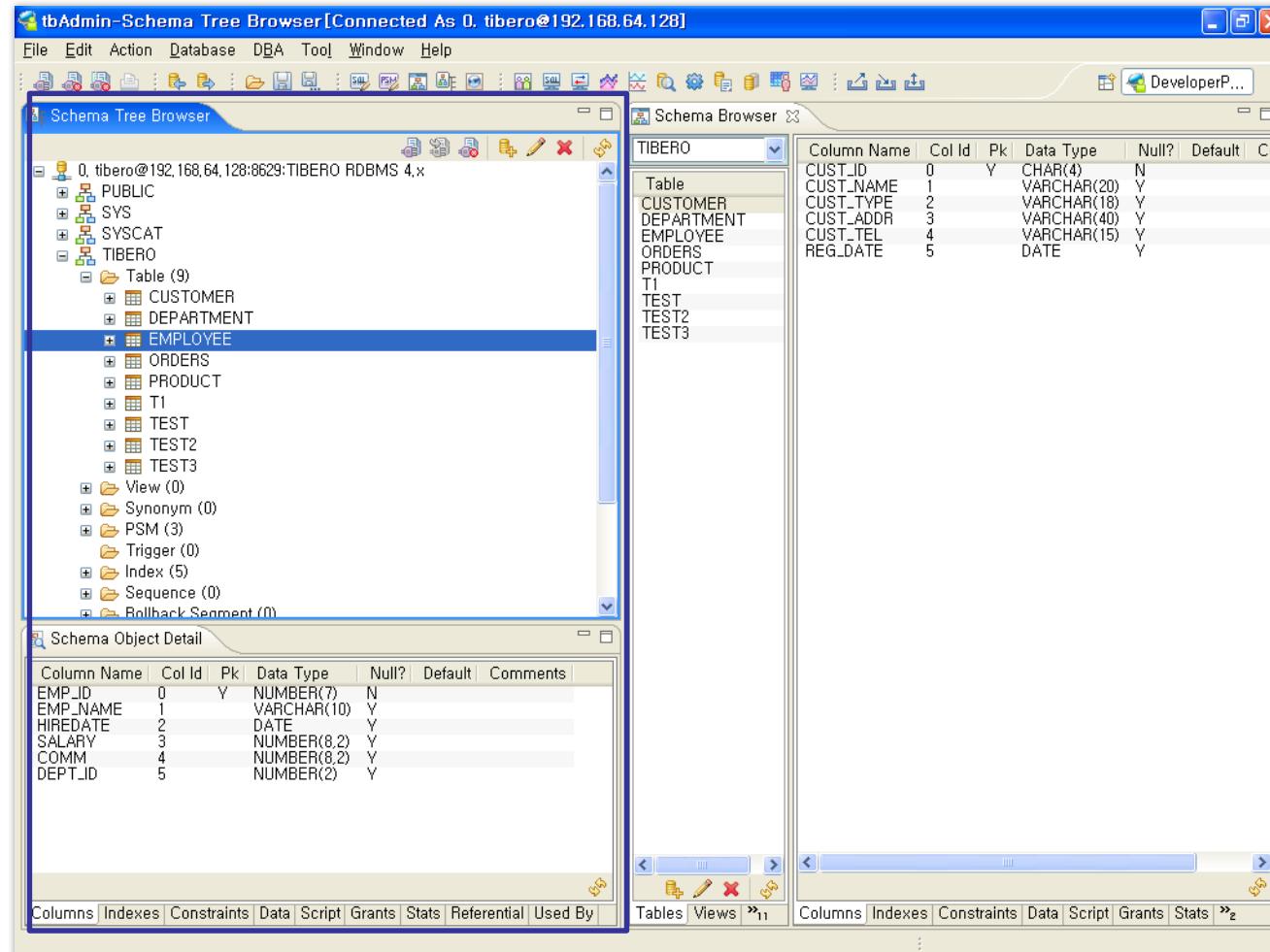
1-- PROCEDURE
2CREATE OR REPLACE PROCEDURE P_1 (V_ENO IN NUMBER)
3IS
4 VENAME VARCHAR(10);
5 VSAL NUMBER(7);
6BEGIN
7   SELECT EMP_NAME, SALARY INTO VENAME, VSAL
8   FROM EMPLOYEE
9   WHERE EMP_ID=V_ENO;
10
11DBMS_OUTPUT.PUT_LINE('-----');
12DBMS_OUTPUT.PUT_LINE('EMPLOYEE      SALARY');
13DBMS_OUTPUT.PUT_LINE('-----');
14DBMS_OUTPUT.PUT_LINE(RPAD(VENAME,8,' ') || ' ' || VSAL);
15DBMS_OUTPUT.PUT_LINE('-----');
16END;|
```
- Schema Object Detail:** A preview window showing the same procedure code.
- PSM Error:** An empty tab.
- DBMS Output:** A table showing the results of the procedure execution:

R..	DBMS Output	Execute Time(ms)
1	-----	5,021029
2	EMPLOYEE SALARY	3,578108
3	-----	2,171784
4	STEVE 2500	2,19134
5	-----	2,167594
- Status Bar:** "successfully compiled".

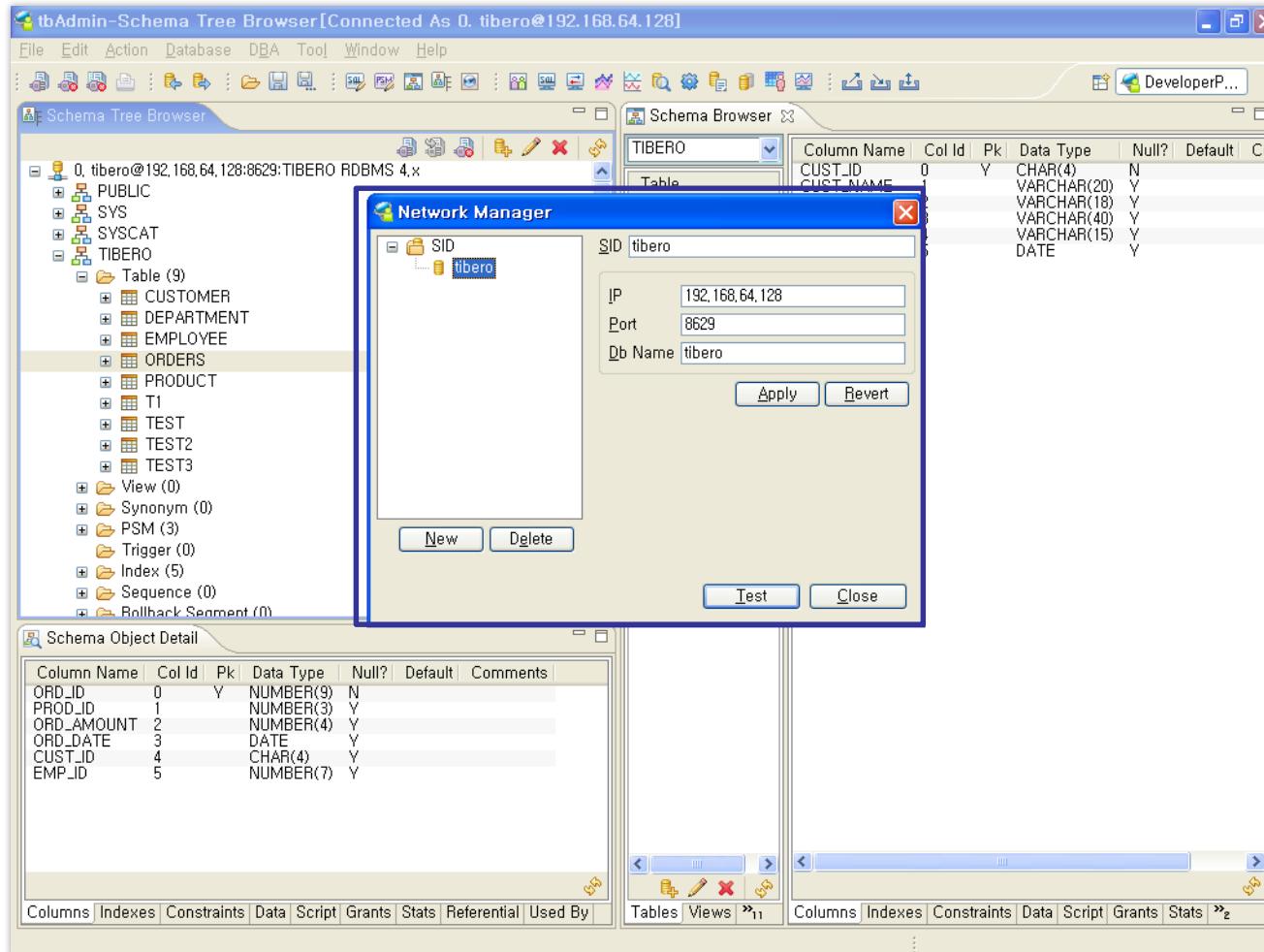
● Schema Browser



● Schema Tree Browser



● Network Manager



●Session Manager

The screenshot shows the tbAdmin Session Manager window. The main pane displays a table of sessions:

Row #	sid	serial no	tibero user	status	type	SQL Trace	log
1	18	5	TIBERO	RUNNING	WTHR	DISABLED	2009
2	19	14	SYS	ACTIVE	WTHR	DISABLED	2009
3	20	15	TIBERO	ACTIVE	WTHR	DISABLED	2009
4	21	16	TIBERO	ACTIVE	WTHR	DISABLED	2009

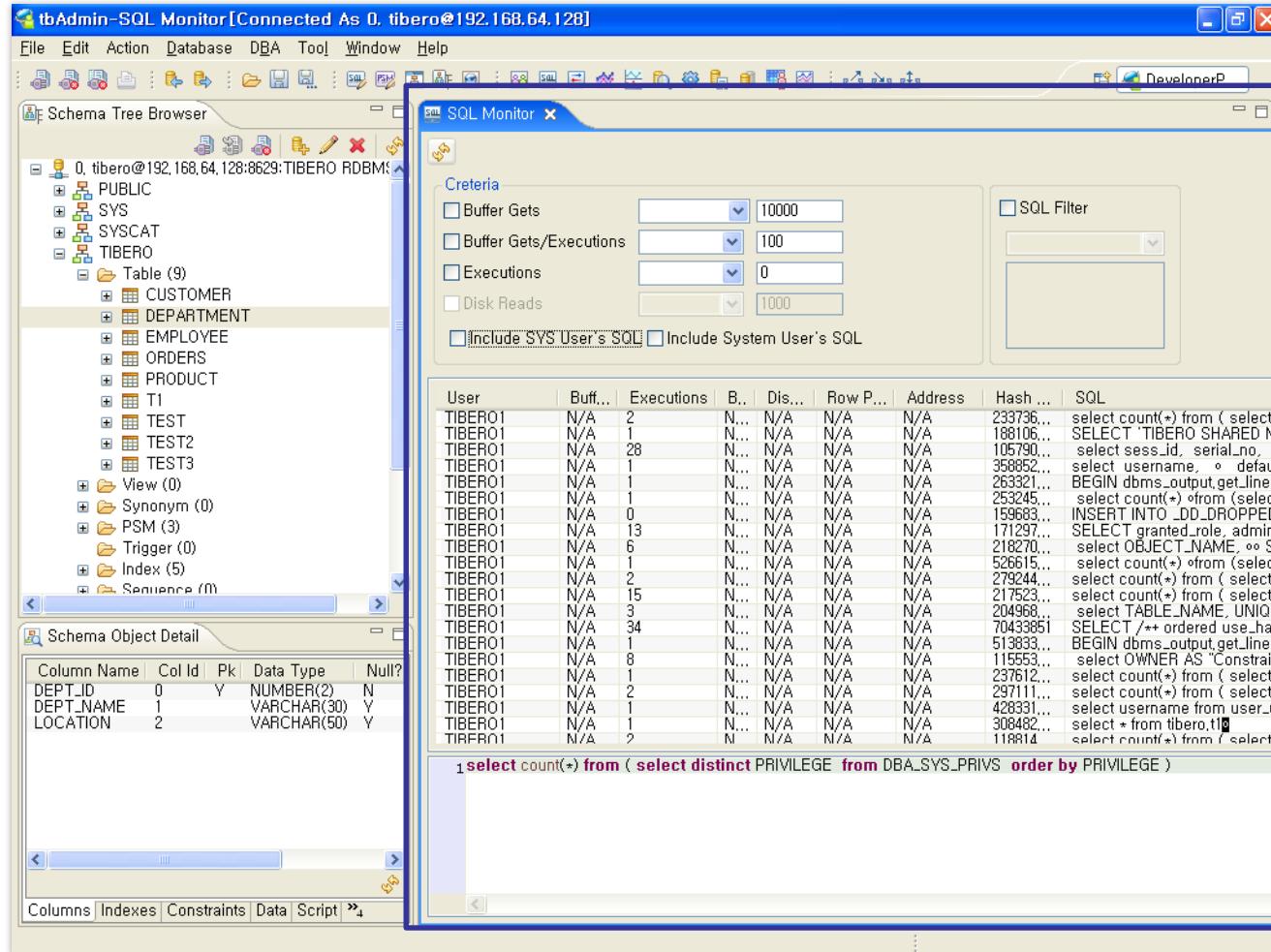
A context menu is open over the fourth row (SID 21), containing options: SQL Trace On/Off, Session Close, and Refresh.

Below the table, there are buttons for Refresh (secs) and Auto Refresh Data?.

The bottom section shows session details for TIBERO SID : 21 and TIBERO User : TIBERO, with tabs for Current Statement and Current Statement Plan.

On the left, the Schema Tree Browser shows the database structure, including the TIBERO schema which contains tables like CUSTOMER, DEPARTMENT, EMPLOYEE, ORDERS, PRODUCT, and TEST.

●SQL Monitor



● Parameter Manager

The screenshot shows the tbAdmin Parameter Manager application window. The title bar reads "tbAdmin - Parameter Manager [Connected As 0. sys@192.168.64.128]". The menu bar includes File, Edit, Action, Database, DBA, Tool, Window, and Help. The toolbar contains various icons for database management tasks.

The left pane is a "Schema Tree Browser" showing the database structure:

- 0. sys@192.168.64.128:8629:TIBERO RDBM
 - PUBLIC
 - SYS
 - SYSCAT
 - TIBERO
 - Table (9)
 - CUSTOMER
 - DEPARTMENT
 - EMPLOYEE
 - ORDERS
 - PRODUCT
 - T1
 - TEST
 - TEST2
 - TEST3
 - View (0)
 - Synonym (0)
 - PSM (3)
 - Trigger (0)
 - Index (5)
 - Sequence (0)

The right pane is the "Parameter Manager" table, listing system parameters with their values and defaults:

Row #	Name	Value	Default
1	NAMED_PIPE_DIR		50606
2	LISTENER_PORT	8629	-1
3	LISTENER_IP	-1	6
4	LOG_LVL_TBLIS	6	6
5	LSNR_LOG_DEST	/home/tibero/tibero4/instance/tibero/log/lsnr/	10485760
6	LSNR_LOG_FILE_SIZE	10485760	429496729
7	LSNR_LOG_TOTAL_SI...	4294967296	
8	LSNR_INVITED_IP		
9	LSNR_DENIED_IP		
10	LOG_LVL_TBCM	2	2
11	TBCM_PORT	8631	8631
12	TBCM_LOG_DEST	/home/tibero/tibero4/instance/tibero/log/tbcm/	10485760
13	TBCM_LOG_FILE_SIZE	10485760	429496729
14	TBCM_LOG_TOTAL_SI...	4294967296	
15	TBCM_TIME_UNIT	10	10
16	TBCM_HEARTBEAT_...	60	60
17	TBCM_NET_EXPIRE_...	5	5
18	TBCM_NODES		
19	TBCM_NAME		
20	TBCM_NODE_ID	0	0
21	TBCM_PROBE_CMD		
22	TBCM_BOOT_CMD		
23	TBCM_DOWN_CMD		
24	TBCM_STANDBY_CMD		
25	TBCM_PROBE_EXPIRE	10	10
26	TBCM_FILE_NAME		
27	TBCM_CLUSTER_MO...		
28	TBCM_USE_WATCHD...	NO	NO
29	TBCM_WATCHDOG_E...	50	50
30	TBCM_COORDINATO...	NO	NO
31	TBCM_COORDINATO...		
32	CLUSTER_DATABASE	NO	NO
33	LOCAL_CLUSTER_AD...		
34	LOCAL_CLUSTER_PO...	0	0

The bottom pane shows "Schema Object Detail" for a selected parameter, which is currently empty.

● Extents Viewer

The screenshot shows the tbAdmin Extents Viewer interface. The main window title is "Extents Viewer [Connected As 0, sys@192.168.64.128]". The left sidebar is a "Schema Tree Browser" showing the database structure under "0, sys@192.168.64.128:8629:TIBERO RDBMS". The "TIBERO" schema is expanded, showing various objects like CUSTOMER, DEPARTMENT, EMPLOYEE, ORDERS, PRODUCT, T1, TEST, TEST2, TEST3, View (0), Synonym (0), PSM (3), Trigger (0), Index (5), and Sequence (0). The central panel is titled "Extents" and displays a table with the following data:

#	OWNER	OBJECT	TYPE	TABLESPACE	TOTAL SIZE	EXTENTS
1	SYS	_DD_VIEW	TABLE	SYSTEM	384KB	3
2	SYS	_DD_COL	TABLE	SYSTEM	384KB	3
3	SYS	_DD_SRC	TABLE	SYSTEM	256KB	2
4	SYS	_DD_PSMIR	TABLE	SYSTEM	256KB	2
5	SYS	_DD_COMMENT	TABLE	SYSTEM	256KB	2
6	TIBERO	T1	TABLE	USR	13312KB	20

At the bottom of the central panel, a message reads "Done, 6/6 rows fetched, [tbAdmin : 0,0 ms]". The bottom navigation bar includes links for Info, Role Grants, and System Privileges.

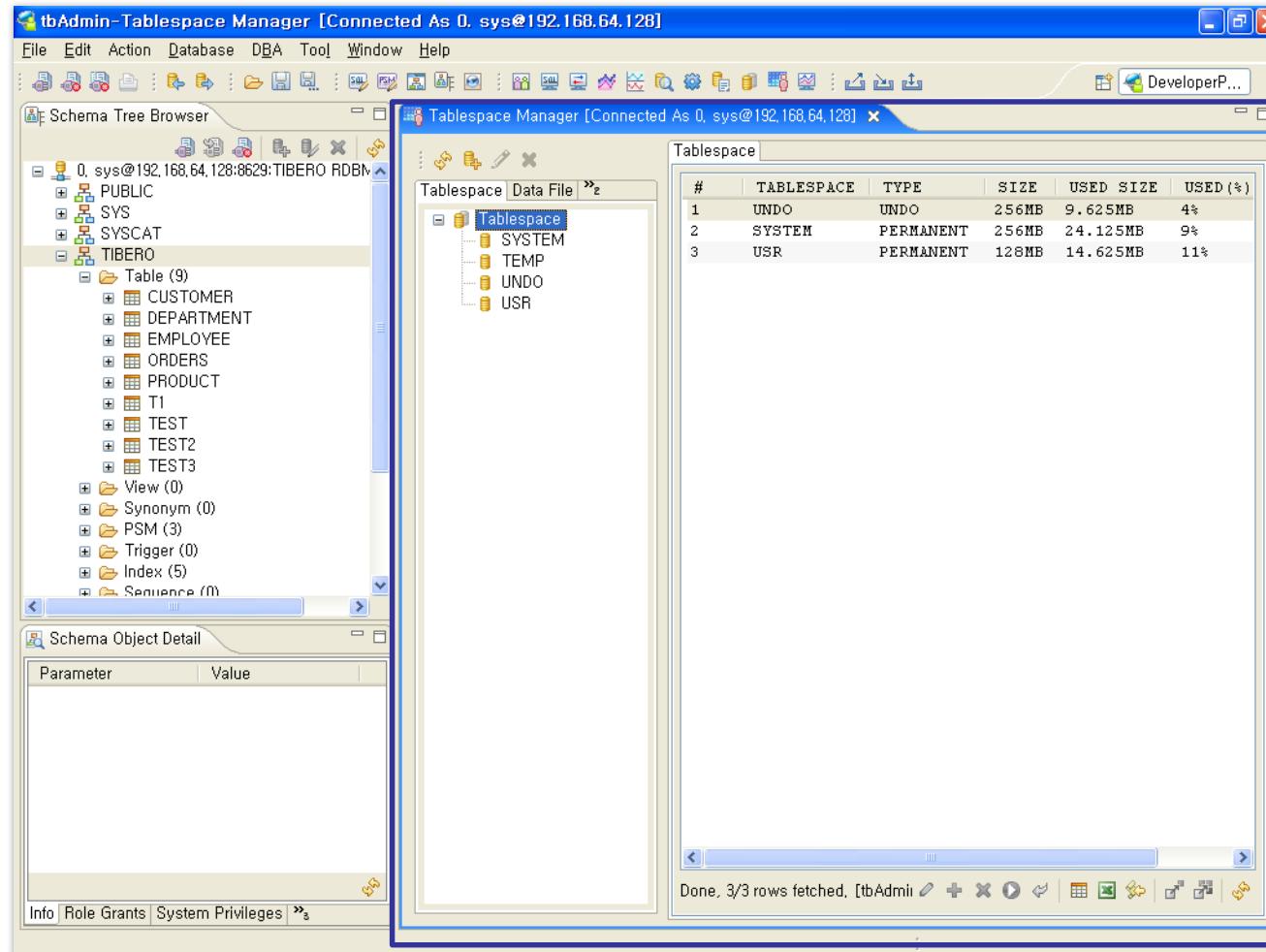
● Database Information

The screenshot shows the tbAdmin interface with the 'Database Information' window open. The window title is 'Database Information [Connected As 0. sys@192.168.64.128]'. The main content area displays a table of database parameters:

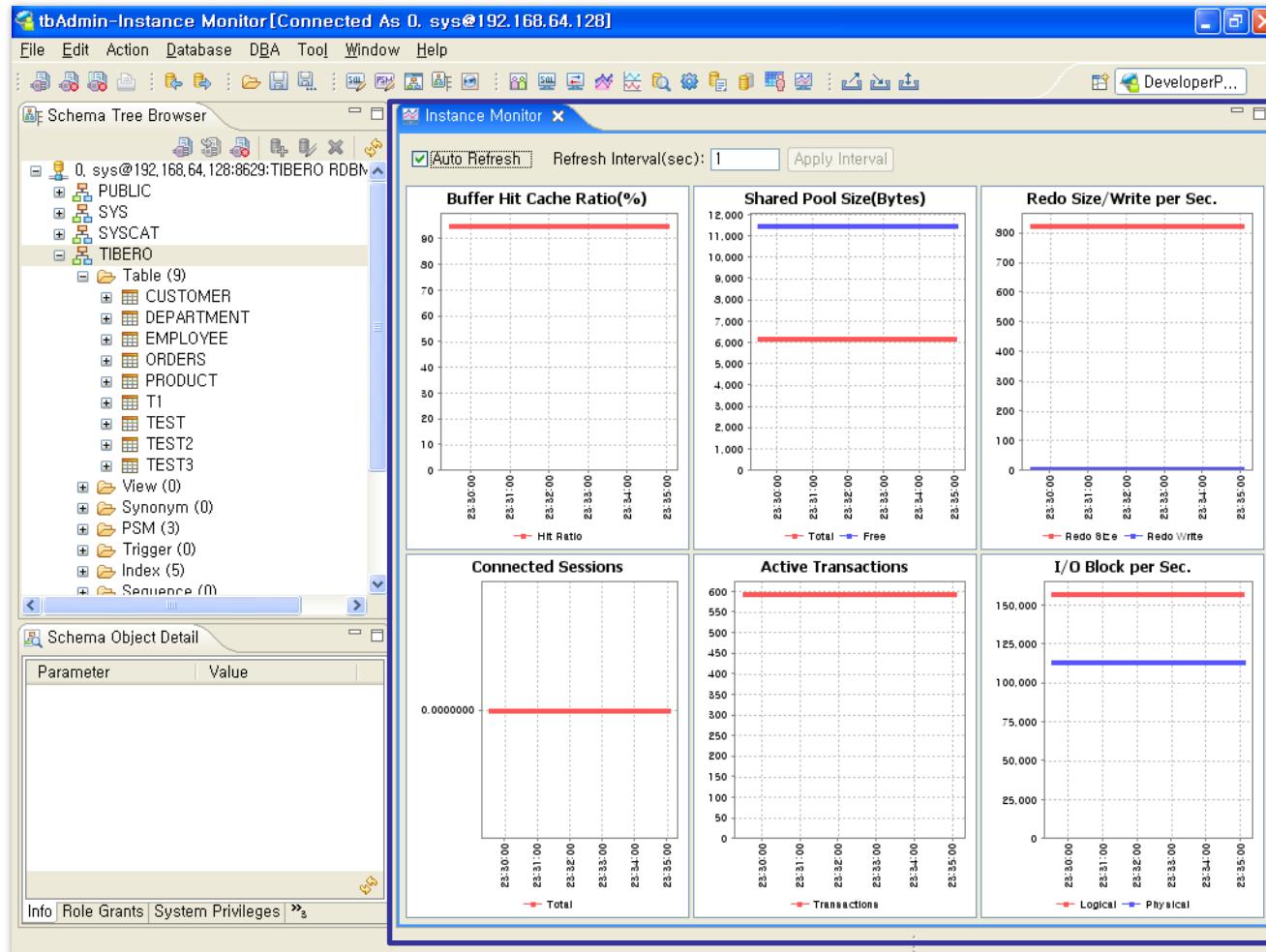
#	NAME	VALUE
1	DBID	-1243938659
2	NAME	tibero
3	CREATE_DATE	2009-10-27 18:54:32
4	OPEN_MODE	READ WRITE
5	LOG_MODE	ARCHIVELOG
6	CURRENT_TSN	38463
7	RESETLOG_TSN	11855
8	PREV_RESETLOG_TSN	0
9	CKPT_TSN	38157
10	CKPT_DATE	2009-11-02 23:04:43

The left sidebar shows the schema tree browser with nodes like PUBLIC, SYS, SYSCAT, and TIBERO, which is expanded to show tables such as CUSTOMER, DEPARTMENT, EMPLOYEE, ORDERS, PRODUCT, TEST, TEST2, and TEST3.

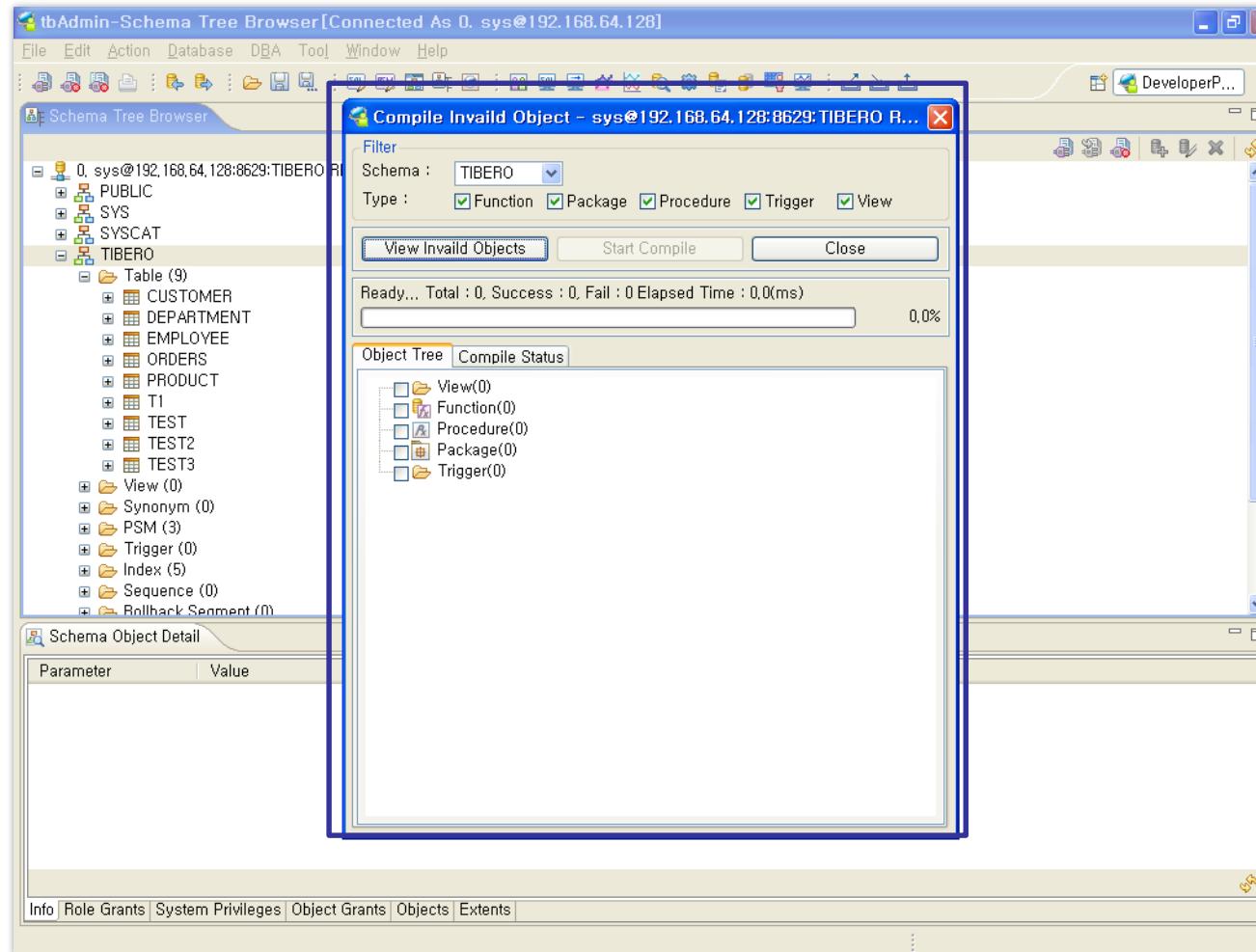
● Tablespace Manager



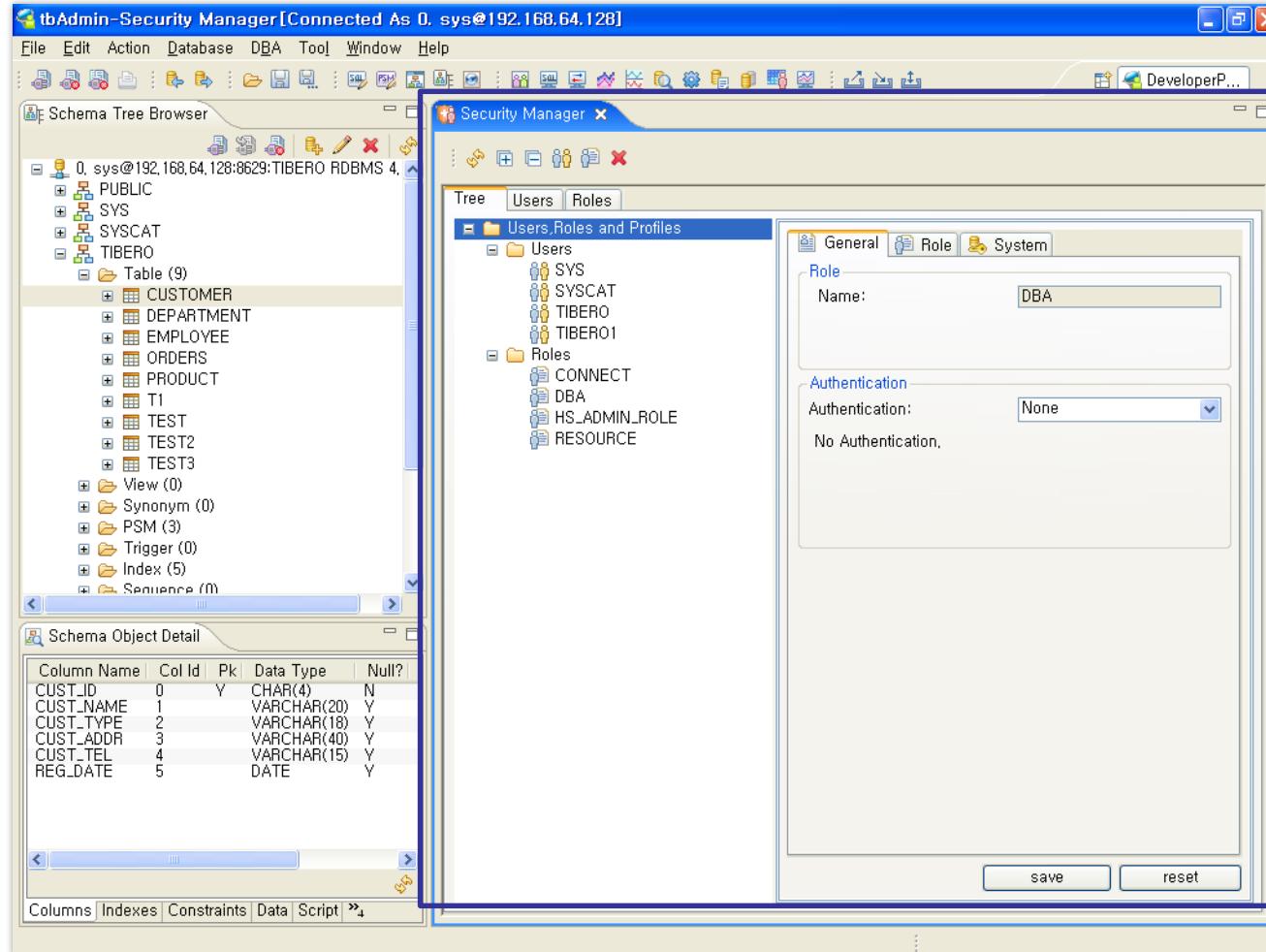
● Instance Monitor



●Compile Invalid Objects



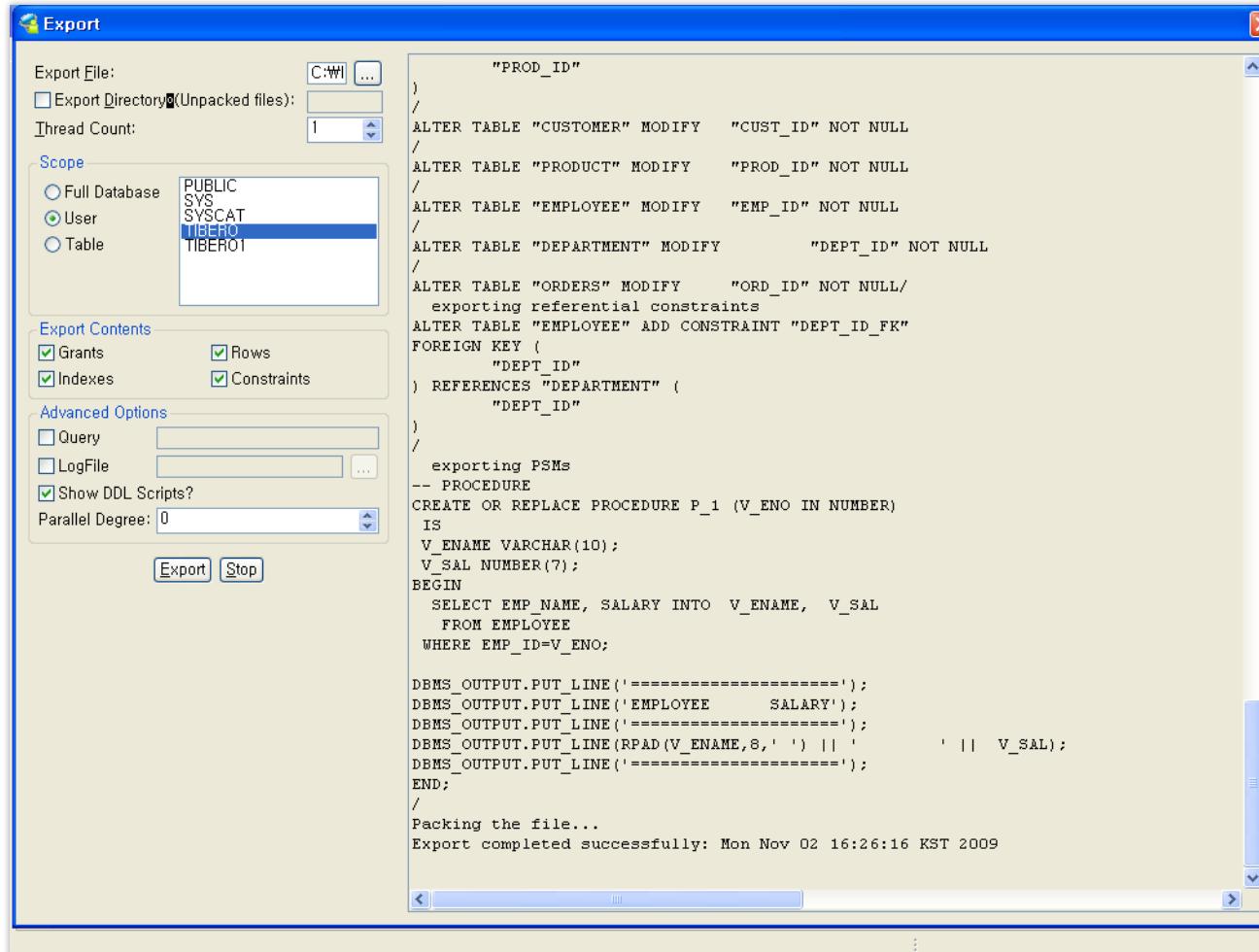
● Security Manager



Export / Import 기능 (1/3)

CHAPTER 4장. tbAdmin

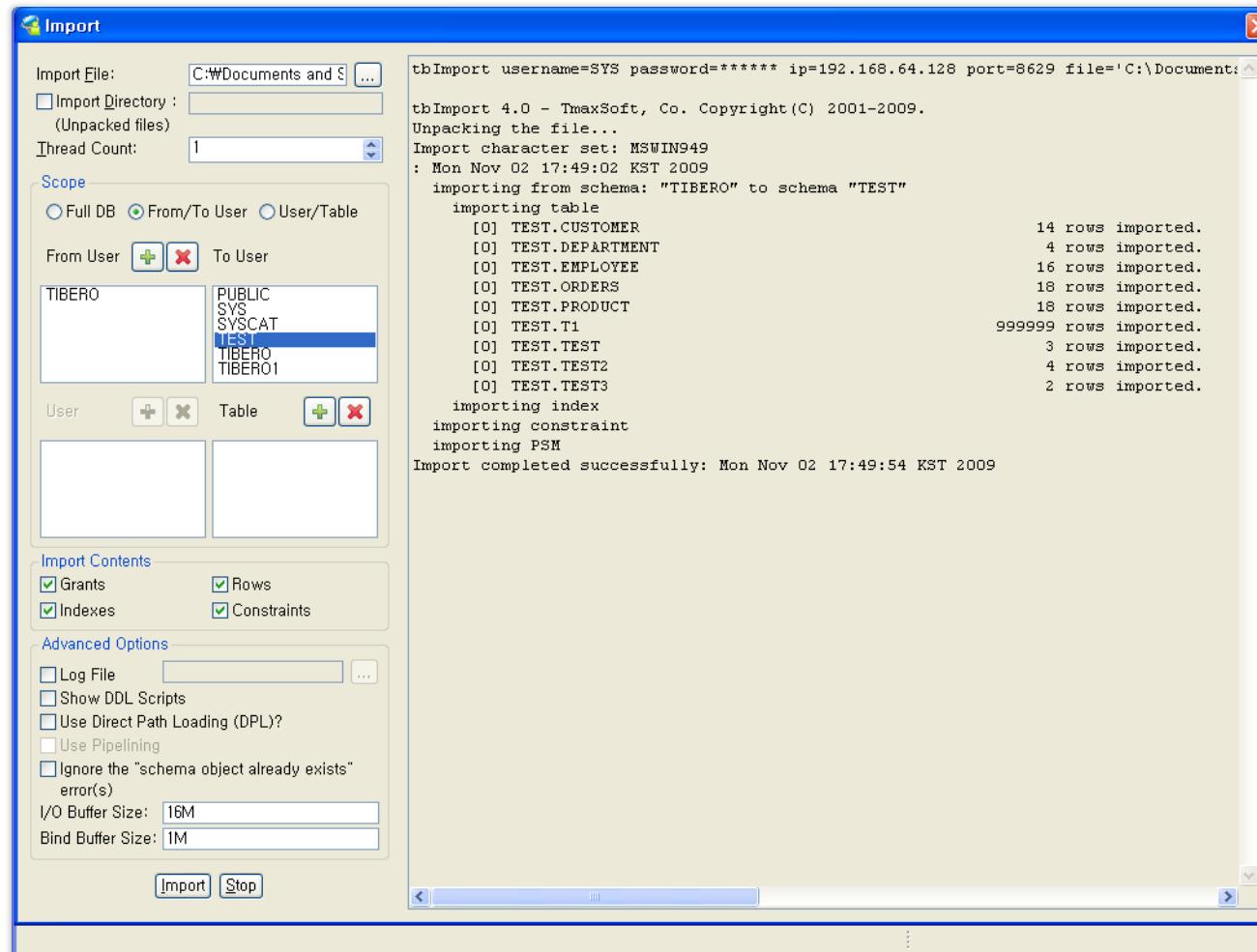
● Export



Export / Import 기능 (2/3)

CHAPTER 4장. tbAdmin

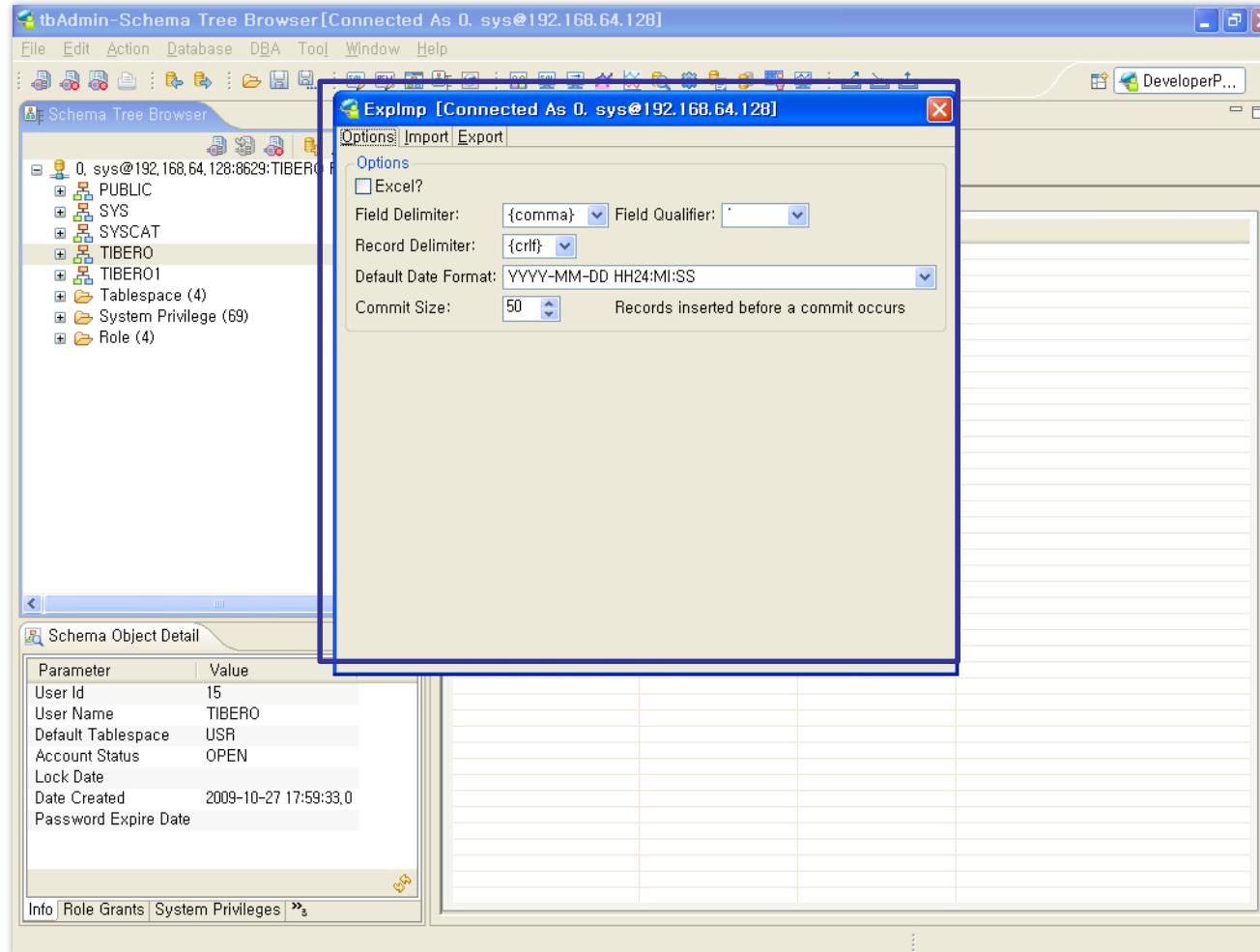
● Import



Export / Import 기능 (3/3)

CHAPTER 4장. tbAdmin

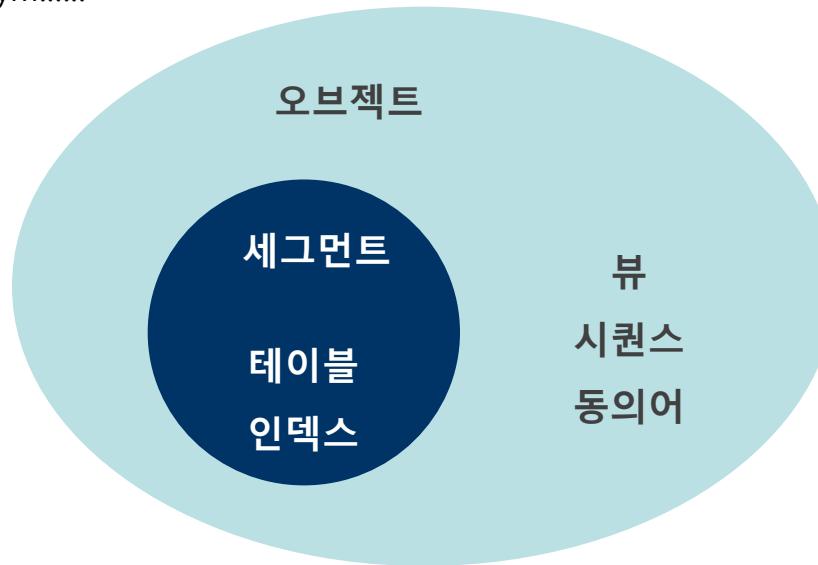
● Explmp



Chapter 5장 Object

● 오브젝트 개요

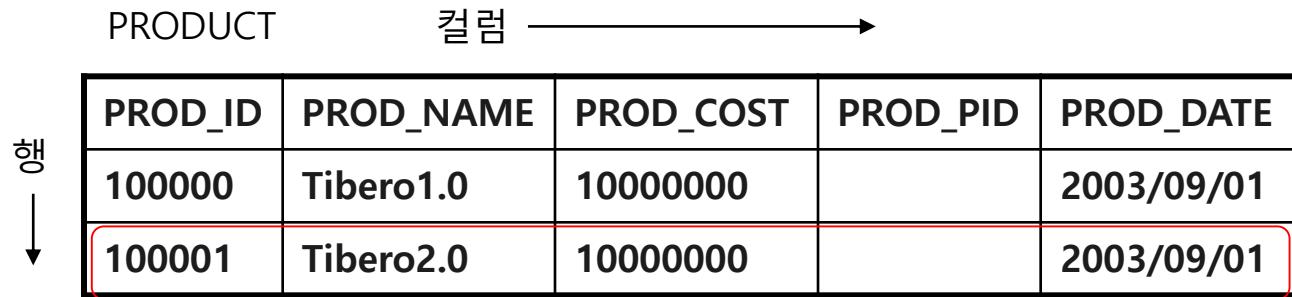
- 티베로의 논리적 구조로 취급할 수 있는 형식의 데이터로 구조상 또는 용법상 관점으로 분류한 것
- 테이블, 인덱스, 뷰, 시퀀스, 동의어, 트리거, 프로시저, 사용자 생성 function, 사용자 생성 패키지
- 모든 오브젝트는 한 스키마에 의해 생성되며, 그 스키마에 속하게 된다.
- 오브젝트 중 실제 물리적 공간을 가지는 오브젝트를 세그먼트라 한다.
 - 세그먼트 : Table, Index, UNDO,
 - 오브젝트 : View, Synonym.....



테이블 관리

● 테이블

- 테이블은 데이터베이스에서 실제 데이터가 저장되는 논리적 구조를 의미
 - 테이블의 구성요소
 - 컬럼(column) : 테이블에 저장될 데이터의 특성을 지정하는 구성요소.
 - 행(row) : 하나의 테이블을 구성하며 다른 유형의 데이터가 저장.
 - 테이블 종류 : 일반 테이블, 파티션 테이블



- 테이블에 사용되는 컬럼 타입

문자형

- CHAR (n) : 고정길이 문자 데이터. 최대 2000byte 까지 선언.
- VARCHAR(n) : 가변길이 문자 데이터. 최대 4000byte 까지 선언.
 - ➔ Tibero에서는 VARCHAR2로 만들어도, 실제로는 VARCHAR 타입으로 생성된다.
 - ➔ Oracle의 VARCHAR2와 호환성을 갖고 있으므로, ORACLE data migration 시 Tibero에서는 VARCHAR 타입을 사용하면 됨.
- NCHAR (n) : 유니코드 문자열을 저장, 고정길이 문자 데이터. 최대 2000byte 까지 선언.
- NVARCHAR (n) : 유니코드 문자열을 저장, 가변길이 문자 데이터. 최대 4000byte 까지 선언.
- RAW : 임의의 바이너리 데이터를 저장하기 위한 타입. 최대 2000byte 까지 선언.
- LONG : VARCHAR와 RAW 타입의 최대 길이를 2G까지 늘린 타입. 일반 문자열 저장
- LONG RAW: VARCHAR와 RAW 타입의 최대 길이를 2G까지 늘린 타입. 바이너리 데이터 저장

숫자형

- NUMBER (p,s) : 가변길이 숫자 데이터. p는 정밀도 자리수, s는 소수점 이하 자리수

날짜형

- DATE : 날짜 및 초 단위까지의 시간을 선언.
- TIME : 초 단위 소수점 9자리까지 시간을 선언.
- TIMESTAMP: 날짜와 초 단위 소수점 9자리까지의 시간을 선언.

대용량 객체형

- CLOB : 최대 4G까지 선언. 일반 문자열 저장
- BLOB : LONG과 LONG RAW 타입을 확장한 데이터 타입이다. 최대 4G까지 선언. 바이너리 저장.
- XMLTYPE : W3C 국제표준 형식으로, XML 데이터를 저장하며 내부적으로 CLOB 형식으로 저장.

- 테이블의 생성

```
SQL> CREATE TABLE PRODUCT (
    PROD_ID      NUMBER(6),
    PROD_NAME    VARCHAR2(50),
    PROD_COST    VARCHAR2(30),
    PROD_PID     NUMBER(6),
    PROD_DATE    DATE
)
TABLESPACE MY_SPACE
PCTFREE 5
INITRANS 3      ;
```

- TABLESPACE : 테이블을 생성하면서 이 테이블의 데이터가 저장될 저장소 지정.
- PCTFREE : 테이블의 각 데이터 블록에 있는 공간 비율로 행을 간신하기 위해 예약해 두는 공간의 비율(%)

```
SQL> ALTER TABLE EMP PCTFREE 10;
```

● 테이블 삭제

- DROP 구문 사용시 테이블의 모든 행이 삭제되고 사용된 공간이 해제된다.

```
SQL>DROP TABLE PRODUCT;
```

- CASCADE 연산자와 함께 사용하면 외래키에 의해 참조되는 기본키를 포함한 테이블일 경우 외래 키 조건도 같이 삭제된다.

```
SQL>DROP TABLE PRODUCT CASCADE CONSTRAINTS ;
```

- 테이블 삭제 구문은 자동 커밋이므로 ROLLBACK이 불가능하다.

● 테이블의 변경

- 사용중인 테이블의 속성을 변경하기 위해서는 ALTER TABLE 문을 이용한다.
- 다른 사용자가 소유한 테이블을 변경하려면 ALTER ANY TABLE 시스템 권한을 갖고 있거나 변경하려는 테이블에 대한 ALTER 오브젝트 권한을 갖고 있어야 한다.
- 컬럼 정의 변경

```
SQL>ALTER TABLE PRODUCT MODIFY (ord_amount default 1 not null);
```

- 컬럼 명 변경

```
SQL>ALTER TABLE PRODUCT RENAME COLUMN ord_amount TO order_amt;
```

- 저장 영역 변경

```
SQL>ALTER TABLE PRODUCT PCTFREE 10;
```

- 테이블의 효율적 관리

- 동시에 액세스될 가능성이 높은 테이블들은 병렬적으로 수행될 가능성을 높이기 위하여 서로 다른 디스크에 저장하는 것이 좋다.
- 각 테이블이 저장될 디스크의 용량을 결정하기 위하여 그 테이블의 최대 크기를 추정하는 것이 중요하다.
- 하나의 테이블을 동시에 액세스하는 트랜잭션의 수에 대해서도 추정해야 한다.
- 테이블에 대한 쓰기 연산이 발생한다면 그에 따라 로그를 저장하기 위한 디스크의 용량도 설정하여야 한다.

● 테이블 정보 조회

뷰	설명
DBA_TABLES USER_TABLES ALL_TABLES	<ul style="list-style-type: none">• 티베로내의 모든 테이블에 대한 정보• 현재 유저에 속한 테이블에 대한 정보• 유저가 접근 가능한 테이블에 대한 정보
DBA_TBL_COLUMNS USER_TBL_COLUMNS ALL_TBL_COLUMNS	<ul style="list-style-type: none">• 티베로내의 모든 테이블, 뷰에 속한 컬럼에 대한 정보• 현재 유저에 속한 테이블, 뷰에 속한 컬럼에 대한 정보• 유저가 접근 가능한 테이블, 뷰에 속한 컬럼에 대한 정보

● 제약조건(Constraints)

- 테이블의 컬럼에 사용자가 원하지 않는 데이터가 입력, 변경, 삭제되는 것을 방지하는 방법
- 제약조건 지정 : Column Level, Table Level

```
CREATE TABLE TEMP_PROD (
    prod_id      number(6)   constraint prod_id_pk primary key ,
    prod_name    varchar2(50) constraint prod_name_nn not null,
    prod_cost    varchar2(30) constraint prod_cost_nn not null,
    prod_pid     number(6) ,
    prod_date    date        constraint prod_date_nn not null
);
```

```
CREATE TABLE TEMP_PROD (
    prod_id number(6),
    prod_name varchar2(50) constraint prod_name_nn not null,
    prod_cost varchar2(30) constraint prod_cost_nn not null,
    prod_pid  NUMBER(6),
    prod_date date       constraint prod_date_nn not null,
    constraint prod_id_pk primary key(prod_id, prod_name )
);
```

- Not Null : 해당 컬럼은 NULL값을 가질 수 없음. 테이블 레벨의 제약 조건 지정은 불가능.
- Unique : 한 테이블 내에서 해당 컬럼은 동일한 값을 가질 수 없음. NULL 값은 여러 행에 입력가능.
- Primary Key : 무결성 제약조건과 고유키 무결성 제약조건을 결합한 개념. NULL 사용 불가
- Foreign Key : 다른 테이블이나 자신 테이블의 Primary key나 Unique key를 참조할 때 사용
- Check : 입력 또는 수정 값이 만족해야 할 조건 지정. 한 컬럼에 대하여 여러 개의 제약조건 지정 가능

- 제약조건의 상태
 - Enable : 제약조건이 활성화 되어 제약 조건을 적용시킨다.
 - Disable : 제약조건이 비활성화 되어 적용되지 않는다.
 - Validate : 기존 저장되어 있는 데이터의 무결성을 보장한다.
 - Novalidate : 기존 저장되어 있는 데이터의 무결성을 보장하지 않는다.
- 제약조건 상태 변경 문

```
SQL> ALTER TABLE PRODUCT MODIFY PRIMARY KEY DISABLE;  
SQL> ALTER TABLE PRODUCT  
      MODIFY CONSTRAINT PROD_UNIQUE ENABLE;  
SQL> ALTER TABLE PRODUCT  
      MODIFY CONSTRAINT PROD_MIN ENABLE NOVALIDATE;
```

- 제약조건 이름 변경

```
SQL> ALTER TABLE PRODUCT RENAME  
      CONSTRAINT PROD_ID_KEY TO PROD_KEY;
```

- 제약조건의 추가

```
SQL> ALTER TABLE PRODUCT  
      ADD CONSTRAINT PROD_DATE_NN CHECK (PROD_DATE IS NOT NULL);
```

- 제약조건의 삭제

```
SQL> ALTER TABLE PRODUCT  
      DROP CONSTRAINT PROD_KEY;
```

- 제약조건에 대한 정보

뷰	설명
DBA_CONSTRAINTS USER_CONSTRAINTS ALL_CONSTRAINTS	<ul style="list-style-type: none">• 티베로내의 모든 제약조건에 대한 정보• 현재 유저에 속한 제약조건에 대한 정보• 유저가 접근 가능한 제약조건에 대한 정보
DBA_CONS_COLUMNS USER_CONS_COLUMNS ALL_CONS_COLUMNS	<ul style="list-style-type: none">• 티베로내의 모든 제약조건에 걸린 컬럼에 대한 정보• 현재 유저에 속한 제약조건에 걸린 컬럼에 대한 정보• 유저가 접근 가능한 제약조건에 걸린 컬럼에 대한 정보

● 인덱스

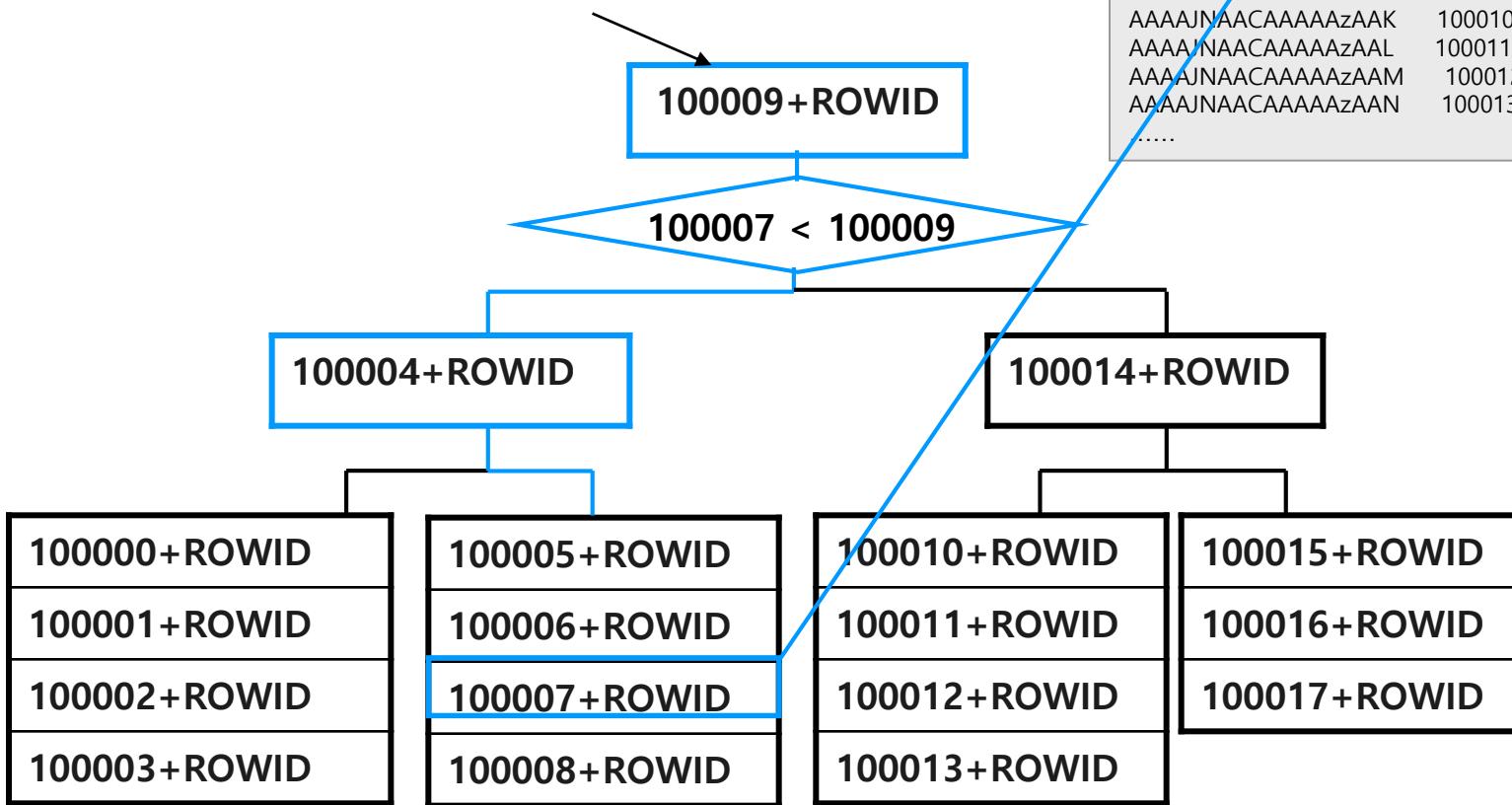
- 테이블에서 원하는 데이터를 빠르게 검색하기 위하여 사용하는 데이터 구조이다.
- 인덱스는 테이블과는 별도의 스키마 오브젝트이므로, 독립적으로 생성, 제거, 변경, 저장할 수 있다
- 인덱스 종류
 - 단일 컬럼 인덱스(Single Index) : 하나의 컬럼으로 만들어진 인덱스.
 - 복합 인덱스(Concatenated Index) : 하나 이상의 컬럼으로 만들어진 인덱스.
 - 유일 인덱스(Unique Index) : 테이블에서 유일한 값을 가진 컬럼으로 만들어진 인덱스.
 - 비유일 인덱스(Non-Unique Index) : 중복되는 값을 인정하는 컬럼으로 만든 인덱스.

인덱스 관리 (계속)

CHAPTER 5장. Object

● 인덱스 기본 구조(Balance-Tree)

```
SQL>SELECT prod_id, prod_name  
      FROM product  
     WHERE prod_id = 100007;
```



● 인덱스의 생성

- 유일 인덱스는 Primary key와 Unique key의 제약조건이 생성될 때 Tibero에서 해당 컬럼에 대하여 인덱스를 자동으로 생성
- 비유일 인덱스 생성 구문

```
CREATE [Unique] INDEX index_name  
ON [schema.]table (column[ASC|DESC][,column])  
[Tablespace tablespace_name]  
[Intrans integer];
```

- Unique : Unique 인덱스 여부
- index_name : 생성할 인덱스의 이름을 설정
- column [ASC | DESC][,column] : 해당 컬럼 정의 및 정렬 방향 지정
- Tablespace tablespace_name : 해당 인덱스의 저장 테이블스페이스 지정
- Intrans integer : 디스크 블록의 파라미터(INITRANS) 값을 설정

● 인덱스 삭제

- 인덱스는 테이블과 같은 독립적인 스키마 오브젝트이므로 인덱스의 삭제는 테이블의 데이터에는 영향을 미치지 않음.
- 인덱스를 삭제하면 해당 컬럼의 데이터를 조회 시 이전과 달리 조회 속도가 느려질 수도 있음.

```
SQL>DROP INDEX prod_id_pk;
```

- 인덱스 생성 지침

- WHERE 조건절에서 자주 사용되는 컬럼을 대상으로 선정.
- Query와 DML 요구 간의 균형.
- Tibero는 기본키, UNIQUE 그리고 외래키 제약조건이 설정된 컬럼들에 대해 자동으로 인덱스 생성.
- 복합키 인덱스 생성 시 컬럼 값의 순서에 유의.

● 인덱스 정보 조회

뷰	설명
DBA_INDEXES USER_INDEXES ALL_INDEXES	<ul style="list-style-type: none">• 티베로내의 모든 인덱스에 대한 정보• 현재 유저에 속한 인덱스에 대한 정보• 유저가 접근 가능한 인덱스에 대한 정보
DBA_IDX_COLUMNS USER_IDX_COLUMNS ALL_IDX_COLUMNS	<ul style="list-style-type: none">• 티베로내의 모든 인덱스에 적용된 컬럼에 대한 정보• 현재 유저에 속한 인덱스에 적용된 컬럼에 대한 정보• 유저가 접근 가능한 인덱스에 적용된 컬럼에 대한 정보

- **파티션(Partition) 테이블**

- 지속적으로 용량이 증가하는 논리적 테이블을 여러 개의 물리적인 공간으로 나누어 성능을 향상시킨 테이블
- 각 파티션 테이블은 별개의 세그먼트에 저장되어 개별적으로 관리가 가능

- **파티션 테이블의 장점**

- 데이터의 엑세스 범위를 줄여 성능 향상
- 물리적 영역을 분할하여 관리되므로 데이터의 손상 가능성 감소
- 각 물리적 영역을 독립적으로 관리하여 대량 데이터에 대한 관리 편의성 제공

- 파티션 테이블 생성

- 파티션을 생성하기 위해서는 CREATE TABLE시 파티션 정보를 서술함으로써 파티션 테이블 생성

```
SQL> CREATE TABLE ORDERED (
    ORD_DATE      VARCHAR(8),
    ORD_ID        CHAR(4),
    PROD_ID       NUMBER(6),
    CUST_ID       CHAR(5),
    EMP_ID        CHAR(4),
    ORD_AMOUNT    NUMBER(4)
)
PARTITION BY RANGE (ORD_DATE)
(
    PARTITION PART1 VALUES LESS THAN ('20090501') TABLESPACE MY_SPACE1,
    PARTITION PART2 VALUES LESS THAN ('20090701') TABLESPACE MY_SPACE2,
    PARTITION PART3 VALUES LESS THAN ('20091001') TABLESPACE MY_SPACE3,
    PARTITION PART4 VALUES LESS THAN ('20100101') TABLESPACE MY_SPACE4
);
```

- 파티션 추가 삭제

```
SQL> ALTER TABLE ORDERED
    ADD PARTITION PART5 VALUES LESS THAN ('20100401');
SQL> ALTER TABLE ORDERED
    DROP PARTITION PART1;
```

- 파티션 테이블 정보 조회

뷰	설명
DBA_PART_TABLES	• 티베로내의 모든 파티션 테이블에 대한 정보
USER_PART_TABLES	• 현재 유저에 속한 파티션 테이블에 대한 정보
ALL_PART_TABLES	• 유저가 접근 가능한 파티션 테이블에 대한 정보

- **파티션(Partition) 인덱스**

- 테이블 뿐 아니라 인덱스 또한 파티션 지정 가능
- 로컬 파티션 인덱스는 테이블의 한 파티션과 1:1로 대응

```
SQL> CREATE INDEX ORDERED_INDEX1
      ON ORDERED (ORD_DATE)
      LOCAL
      (
          PARTITION PART1,
          PARTITION PART2,
          PARTITION PART3,
          PARTITION PART4
      );
```

- 글로벌 파티션 인덱스는 테이블의 파티션과 무관. 어느 파티션에 있는 행도 가르킬 수 있음.

```
SQL> CREATE INDEX ORDERED_INDEX2
      ON ORDERED (ORD_DATE);
```

```
SQL> CREATE INDEX ORDERED_INDEX2
      ON ORDERED (ORD_DATE)
      GLOBAL PARTITION BY RANGE (ORD_DATE)
      (
          PARTITION PART1 VALUES LESS THAN ('20090701') TABLESPACE MY_SPACE5,
          PARTITION PART2 VALUES LESS THAN ('20100101') TABLESPACE MY_SPACE6
      );
```

● 파티션 인덱스 정보 조회

뷰	설명
DBA_PART_INDEXES	• 티베로내의 모든 파티션 인덱스에 대한 정보
USER_PART_INDEXES	• 현재 유저에 속한 파티션 인덱스에 대한 정보
ALL_PART_INDEXES	• 유저가 접근 가능한 파티션 인덱스에 대한 정보

● 뷰(View)

- SELECT 문장으로 표현되는 질의에 이름을 부여하여 정의한 가상의 테이블
- SQL 문장 내에서 테이블과 동일하게 사용
- 뷰를 통한 데이터 변경 가능
 - 모든 뷰에 대한 질의 연산은 가능하지만, 삽입, 갱신, 삭제 연산이 불가능한 뷰가 존재
 - 삽입, 갱신, 삭제 연산을 수행할 수 있는 뷰를 갱신 가능한 뷰(updatable view)라고 함.

● 뷰(View) 생성

```
SQL> CREATE VIEW V_PROD AS
      SELECT prod_id, prod_name, prod_cost
        FROM product
       WHERE prod_id= 100001;
```

● 뷰(View) 삭제

```
SQL> DROP VIEW V_PROD;
```

● 뷰(View) 정의 변경

- 뷰의 정의 변경은 저장된 SQL구문을 새로 생성하는 것으로 삭제 후 새로 생성

```
SQL> CREATE OR REPLACE VIEW V_PROD AS
      SELECT prod_id, prod_name, prod_cost
        FROM product;
```

● 뷰(View) 정보 조회

뷰	설명
DBA_VIEWS USER_VIEWS ALL_VIEWS	<ul style="list-style-type: none">• 티베로내의 모든 뷰에 대한 정보• 현재 유저에 속한 뷰에 대한 정보• 유저가 접근 가능한 뷰에 대한 정보
DBA_UPDATABLE_COLUMNS USER_UPDATABLE_COLUMNS ALL_UPDATABLE_COLUMNS	<ul style="list-style-type: none">• 티베로내의 모든 뷰에 속한 컬럼에 대한 갱신 가능성 정보• 현재 유저에 속한 뷰에 속한 컬럼에 대한 갱신 가능성 정보• 유저가 접근 가능한 뷰에 속한 컬럼에 대한 갱신 가능성 정보

● 시퀀스(Sequence)

- Tibero에서 순차적으로 자동 부여한 고유번호
- 시퀀스 생성 (CREATE SEQUENCE 권한이 있어야 생성 가능)

```
CREATE SEQUENCE [schema.]sequence_name  
INCREMENT BY n  
MINVALUE n  
MAXVALUE n  
CYCLE|NOCYCLE  
CACHE n|NOCACHE;
```

- INCREMENT BY : 시퀀스 번호의 증가치를 설정
- MINVALUE / MAXVALUE : 시퀀스의 최소값 / 최대값 설정
- NOCYCLE : 최고값까지 증가가 완료되면 에러를 발생.
- CACHE : 시퀀스의 값을 메모리에서 관리하는 방법으로 기본적으로 10

```
SQL> INSERT INTO test VALUES (new_id.NEXTVAL);
```

```
SQL> ALTER SEQUENCE new_id  
INCREMENT BY 5  
MAXVALUE 999999  
NOCYCLE  
CACHE 10;
```

```
SQL> DROP SEQUENCE new_id;
```

● 시퀀스(Sequence) 정보 조회

뷰	설명
DBA_SEQUENCES	• 티베로내의 모든 시퀀스에 대한 정보
USER_SEQUENCES	• 현재 유저에 속한 시퀀스에 대한 정보
ALL_SEQUENCES	• 유저가 접근 가능한 시퀀스에 대한 정보

- 시노님(Synonym) 관리

- 시노님(Synonym)은 오브젝트에 대한 별칭(ALIAS)
- 동의어 생성

```
SQL> CREATE SYNONYM T1 FOR SM.PRODUCT;
```

```
SQL> CREATE PUBLIC SYNONYM PUB_T1 FOR SM.PRODUCT;
```

- 동의어 정의를 변경하기 위해서는 동의어를 제거하고 다시 생성
- 동의어를 제거하기 위해서는 DROP SYNONYM 을 사용하며, 공유 동의어를 제거하려면 DROP PUBLIC SYNONYM 명령 이용

```
SQL> DROP SYNONYM T1;
```

```
SQL> DROP PUBLIC SYNONYM T1;
```

● 시노님(Synonym) 정보 조회

뷰	설명
DBA_SYNONYMS	• 티베로내의 모든 동의어에 대한 정보
USER_SYNONYMS	• 현재 유저에 속한 동의어에 대한 정보
ALL_SYNONYMS	• 유저가 접근 가능한 동의어에 대한 정보
PUBLICSYN	• 모든 공유 동의어에 대한 정보

Chapter 6장 SQL 문장의 구성요소

SQL 문장의 구성요소

CHAPTER 6장. SQL 문장의 구성요소

구분	설명
데이터 타입	Tibero에서는 SQL 표준에 기반한 여러 가지 데이터 타입을 제공한다
리터럴	상수 값을 의미한다.
형식 문자열	NUMBER 타입과 날짜형 타입의 값을 문자열로 변환하기 위한 형식을 정의한 것이다.
의사 컬럼	시간 간격을 표현하는 데이터 타입이다.
NULL	한 로우에서 어떤 컬럼에 값이 없을 때 그 컬럼을 NULL이라고 한다.
주석	책이나 문서에서 주석이 낱말이나 문장의 뜻을 쉽게 풀이하는 역할을 하듯 SQL 문장에도 주석을 활용하여 해당 문장의 부연 설명을 삽입할 수 있다.
힌트	SQL 문장에 힌트를 추가하여 Tibero의 질의 최적화기(Optimizer)에 특정 행동을 지시하거나 질의 최적화기의 실행 계획을 변경한다.
스키마 객체	한 사용자가 하나의 스키마만을 정의할 수 있고, 스키마의 이름은 항상 사용자의 이름과 동일하다. 이러한 스키마에 포함된 객체를 스키마 객체라 한다.

- Tibero에서는 SQL 표준에 기반한 여러 가지 데이터 타입을 제공한다.

구분	Data Type	설명
문자형	CHAR, VARCHAR, NCHAR, NVARCHAR, RAW, LONG, LONG RAW	문자열을 저장하는 데이터 타입이다.
숫자형	NUMBER	정수나 실수의 숫자를 저장하는 데이터 타입이다.
날짜형	DATE, TIME, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE	시간이나 날짜를 저장하는 데이터 타입이다.
간격형	INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND	시간 간격을 표현하는 데이터 타입이다.
대용량 객체형	CLOB, BLOB, XML TYPE	대용량 객체를 저장하기 위해 제공하는 가장 큰 데이터 타입이다.
내재형	ROWID	사용자가 명시적으로 선언하지 않아도 Tibero RDBMS가 자동으로 삽입되는 로우마다 포함하는 컬럼의 타입이다.

● 문자형

- 문자열을 표현하는 데이터 타입이다. 문자형에는 CHAR 타입, VARCHAR 타입, NCHAR 타입, NVARCHAR 타입, RAW 타입, LONG 타입, LONG RAW 타입이 있다.

구분	Data Type	설명
문자형	CHAR, VARCHAR, NCHAR, NVARCHAR, RAW, LONG, LONG RAW	문자열을 저장하는 데이터 타입이다.

●CHAR

- 문자열을 저장하는 데이터 타입이다. 항상 고정된 문자열 길이를 갖는다.

CHAR(size[BYTE|CHAR])

옵션	설명
size	<ul style="list-style-type: none">✓ 최댓값은 2,000byte이다.✓ 문자열의 길이가 2,000 byte를 넘으면 에러가 발생한다.<ul style="list-style-type: none">• size를 입력한 경우: size만큼의 고정 길이를 갖는다.• size를 입력하지 않은 경우: 디폴트 값인 1byte의 고정 길이를 갖는다.• 문자열의 길이는 byte 기준과 문자 기준이 있다. 예) CHAR (10, BYTE), CHAR (10, CHAR)• 문자기준 일 때는 한 문자가 몇 byte로 표현되는 문자 집합인지에 따라 그 길이가 달라진다• SQL 문장에서 CHAR 타입의 값을 표현할 때에는 작은따옴표(' ')를 사용한다.• 문자열의 길이가 0인 값은 NULL로 인식된다.

- CHAR 타입을 설명하는 예이다.

PRODUCT_NAME CHAR(10)

← 'Tibero' 문자열 입력

- 위와 같이 정의된 PRODUCT_NAME 컬럼에 'Tibero' 문자열이 입력되었다면, 네 개의 공백 문자(Space)가 채워져서 'Tibero____' 문자열이 저장된다.

● VARCHAR

- 문자열을 저장하는 데이터 타입으로, 가변 길이를 갖는다.

VARCHAR(size[BYTE|CHAR])

옵션	설명
size	<ul style="list-style-type: none">✓ 최댓값은 65,532 byte이다.✓ 문자열의 길이가 65,532 byte를 넘으면 에러가 발생한다.<ul style="list-style-type: none">• size를 입력한 경우: size만큼의 고정 길이를 갖는다.• size를 입력하지 않은 경우: 디폴트 값인 1byte의 고정 길이를 갖는다.• 문자열의 길이는 byte 기준과 문자 기준이 있다. 예) VARCHAR (10, BYTE), VARCHAR (10, CHAR)• 문자기준 일 때는 한 문자가 몇 byte로 표현되는 문자 집합인지에 따라 그 길이가 달라진다• SQL 문장에서 VARCHAR 타입의 값을 표현할 때에는 작은따옴표(' ')를 사용한다.• 문자열의 길이가 0인 값은 NULL로 인식된다.

- CHAR 타입을 설명하는 예이다.

EMP_NAME VARCHAR(10)

← 'Peter' 문자열 입력

- 위와 같이 정의된 EMP_NAME 컬럼에 'Peter' 문자열이 입력되었다면, 10byte로 선언되었지만 실제로 저장된 문자열 길이는 5byte 가 된다.

●NCHAR

-유니코드 문자열을 저장하기 위한 타입으로, 고정된 문자열 길이를 갖는다.

NCHAR(size)	
옵션	설명
size	<ul style="list-style-type: none">✓ 최댓값은 2,000 자, 길이는 문자 기준이다.(2,000 byte를 초과할 수 없음)✓ 타입의 길이는 데이터베이스의 다국어 문자 집합에 따라 정해진다. 예) UTF8인 경우엔 size의 최대 3배, UTF16인 경우엔 size의 최대 2배가 된다.✓ SQL 문장에서 CHAR 타입의 값을 표현할 때에는 작은따옴표(' ')를 사용한다.✓ 문자열의 길이가 0인 값은 NULL로 인식된다.

●NVARCHAR

-유니코드 문자열을 저장하기 위한 타입으로, 가변 길이를 갖는다.

NCHAR(size)	
옵션	설명
size	<ul style="list-style-type: none">✓ 최댓값은 65,532 자 (65,532byte를 초과할 수 없음)✓ 길이는 문자 기준이다.✓ 타입의 길이는 데이터베이스의 다국어 문자 집합에 따라 정해진다. 예) UTF8인 경우엔 size의 최대 3배, UTF16인 경우엔 size의 최대 2배가 된다.✓ SQL 문장에서 CHAR 타입의 값을 표현할 때에는 작은따옴표(' ')를 사용한다.✓ 문자열의 길이가 0인 값은 NULL로 인식된다.

●RAW

- 바이너리 데이터를 저장하는 데이터 타입으로, 가변 길이를 갖는다.
- CHAR, VARCHAR 와 차이점
 - RAW 타입은 데이터 중간에 NULL 문자('W0')가 올 수 있지만 CHAR, VARCHAR 타입은 그렇지 않다. 따라서 RAW 타입은 NULL 문자로 데이터의 끝을 나타낼 수 없으므로 항상 길이 정보를 같이 저장한다.

RAW(size)

옵션	설명
size	<ul style="list-style-type: none">✓ 최댓값은 2,000byte이다.✓ 데이터 중간에 NULL 문자('W0')가 올 수 있다.

●LONG RAW

- RAW 타입을 확장한 타입으로 바이너리 데이터가 저장된다.

LONG RAW(size)

옵션	설명
size	<ul style="list-style-type: none">✓ 최댓값은 2GB이다.✓ 테이블 내의 한 컬럼에만 선언 할 수 있다.✓ LONG RAW 타입의 컬럼에 대해 인덱스를 생성 할 수 없다.

● LONG

-VARCHAR 타입을 확장한 데이터 타입이다. 가변 길이 문자열이 저장된다.

LONG

설명

- ✓ 최댓값은 2GB이다.
- ✓ LONG 컬럼이 있는 테이블을 생성하지 마십시오. 대신에 LOB 열 (CLOB , BLOB)를 사용하십시오. LONG 컬럼은 Oracle 과의 호환성을 위해 지원하고 있습니다.
- ✓ LONG 타입의 컬럼을 포함한 로우(Row)가 디스크에 저장될 때에는 다른 컬럼의 값과 함께 동일한 디스크 블록에 저장되며, 길이에 따라 여러 디스크 블록에 걸쳐 저장될 수 있다.
- ✓ 테이블 내의 한 컬럼에만 선언 할 수 있다.
- ✓ LONG 타입의 컬럼에 대해 인덱스를 생성 할 수 없다.
- ✓ LONG 열은 WHERE 절이나 무결성 제약 조건은 지정할 수 없습니다 (NULL 및 NOT NULL 제약 조건은 제외).
- ✓ 내장 함수는 LONG 값을 변경할 수 없습니다.
- ✓ LONG 데이터는 정규 표현식에서 사용할 수 없습니다.
- ✓ LONG 데이터 형식을 사용하여 PL / SQL 프로그램 단위의 변수 또는 인수를 선언할 수 있습니다. 그러나 이 프로그램은 SQL에서 호출할 수 없습니다.

● 숫자형

- 정수나 실수의 숫자를 저장하는 데이터 타입이다. 숫자형에는 NUMBER 타입, INTEGER 타입, FLOAT 타입이 있다.
- Tibero에서는 ANSI에서 제정한 SQL 표준의 숫자 타입의 선언을 지원한다. 즉, INTEGER 타입 또는 FLOAT 타입으로 컬럼을 선언하더라도 내부적으로 적절한 정밀도와 스케일을 설정하여 NUMBER 타입으로 변환해 준다.

구분	Data Type	설명
숫자형	NUMBER, INTEGER , FLOART	정수나 실수의 숫자를 저장하는 데이터 타입이다.

●NUMBER(1/2)

- 정수 또는 실수를 저장하는 데이터 타입이다.
- 실제로 데이터베이스에 저장될 때는 숫자의 크기에 따라 가변 길이로 저장된다.

NUMBER[(precision | precision , scale)]

옵션	설명
precision	<p>정밀도는 데이터의 전체 자릿수이다.</p> <ul style="list-style-type: none">- 정밀도를 초과하는 자릿수의 데이터는 저장할 수 없다.- 정밀도는 1 ~ 38까지 정의할 수 있다.- 정밀도는 별표(*)로도 선언할 수 있다. <p>최대 38자리 한도 내에서 임의의 자릿수를 갖는 모든 데이터 값을 받아들이겠다는 의미이며, 대체로 스케일 값을 함께 선언한다.</p>
scale	<p>스케일이며, 소수점의 자릿수를 설정한다.</p> <ul style="list-style-type: none">- 마이너스의 스케일은 소수점 위의 자릿수이다.- 스케일을 초과하는 데이터는 반올림을 수행한다.- 스케일은 -125 ~ 130까지 정의할 수 있다.

● NUMBER(2/2)

- 다음은 입력된 데이터가 NUMBER 타입의 정밀도와 스케일에 정의된 값에 따라 실제 데이터베이스에 어떤 형태로 저장되는지를 보여준다.

입력된 데이터	NUMBER 타입 선언	실제저장된 데이터
12,345.678	NUMBER	12,345.68
12,345.678	NUMBER(*,3)	12,345.68
12,345.678	NUMBER(8,3)	12,345.68
12,345.678	NUMBER(8,2)	12,345.68
12,345.678	NUMBER(8)	12,346
12,345.678	NUMBER(8,-2)	12,300
12,345.678	NUMBER(3)	(ERROR) 입력된 데이터의 자릿수가 정밀도를 초과했으므로 저장할 수 없다.

● 날짜형

- 시간이나 날짜를 저장하는 데이터 타입이다. 날짜형에는 DATE 타입, TIME 타입, TIMESTAMP 타입이 있다.

구분	Data Type	설명
날짜형	DATE, TIME, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE	시간이나 날짜를 저장하는 데이터 타입이다.

● DATE

- 특정 날짜와 초 단위 까지의 시간을 표현하는 데이터 타입이다.

DATE

설명

- ✓ 연도, 월, 시, 분, 초를 표현할 수 있다.
- ✓ 연도는 BC 9,999 ~ AD 9,999까지 표현할 수 있다.
- ✓ 시간은 24시간 단위로 표현된다.
- ✓ 초 이하 단위(밀리 초 등)과, Timezone과 관련된 정보를 저장하지 않음.

●TIMESTAMP

- 날짜와 초 단위 소수점 9자리까지의 시간을 모두 표현하는 데이터 타입이다.
 - 연도, 월, 시, 분, 초, 10^{-9} 초를 표현할 수 있다.
 - 연도는 BC 9,999 ~ AD 9,999까지 표현 가능하다.
 - 시간은 24시간 단위로 표현된다.

TIMESTAMP[(frac_sec_prec)]

옵션	설명
frac_sec_prec	<ul style="list-style-type: none">✓ 시간 값의 소수 초 정밀도(fractional second precision)를 설정 한다.<ul style="list-style-type: none">- 0부터 9까지 사용할 수 있다.- 디폴트 값은 6이다.

●TIMESTAMP WITH TIME ZONE

- TIMESTAMP 타입을 확장하여 시간대까지 표현하는 데이터 타입이다.
 - 연도, 월, 일, 시, 분, 초, 소수점 초 등은 TIMESTAMP 타입과 동일한 특징을 가진다.
 - 각 시간 요소들을 UTC(Coordinated Universal Time) 시간으로 정규화해서 저장한다.
 - 지역 이름이나 오프셋으로 표현된 시간대를 포함하여 저장한다. 여기서 오프셋은 현재 지역의 시간과 UTC 시간과의 차이를 말한다.

TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE

옵션	설명
fractional_seconds_precision	<ul style="list-style-type: none">✓ 시간 값의 소수 초 정밀도(fractional second precision)를 설정 한다.<ul style="list-style-type: none">- 0부터 9까지 사용할 수 있다.- 디폴트 값은 6이다.

● INTERVAL YEAR TO MONTH

- 년과 월의 차이를 저장하는 데이터 타입이다
- 실제로 데이터베이스에 저장될 때는 고정 길이로 저장된다.

INTERVAL YEAR [(year_prec)] TO MONTH

옵션	설명
year_prec	year precision를 설정한다. - 0부터 9까지 사용할 수 있다. - 디폴트 값은 2이다. - 년의 자릿수를 제한할 수 있다.

● TIME

- 초 단위 소수점 9자리까지의 특정 시간을 표현하는 데이터 타입이다.

● INTERVAL DAY TO SECOND

- 날짜와 시간의 차이를 저장하는 데이터 타입이다.
- 실제로 데이터베이스에 저장될 때는 고정 길이로 저장된다.

INTERVAL DAY [(day_prec)] TO SECOND [(frac_sec_prec)]

옵션	설명
day_prec	day precision를 설정한다. - 0부터 9까지 사용할 수 있다. - 디폴트 값은 2이다.
frac_sec_prec	TIMESTMAP 타입에 정의된 옵션과 같다.

● 대용량 객체형

- 대용량의 객체를 저장하기 위해 Tibero에서 제공하는 가장 큰 데이터 타입이며, CLOB 타입과 BLOB 타입, XMLTYPE 타입이 있다.

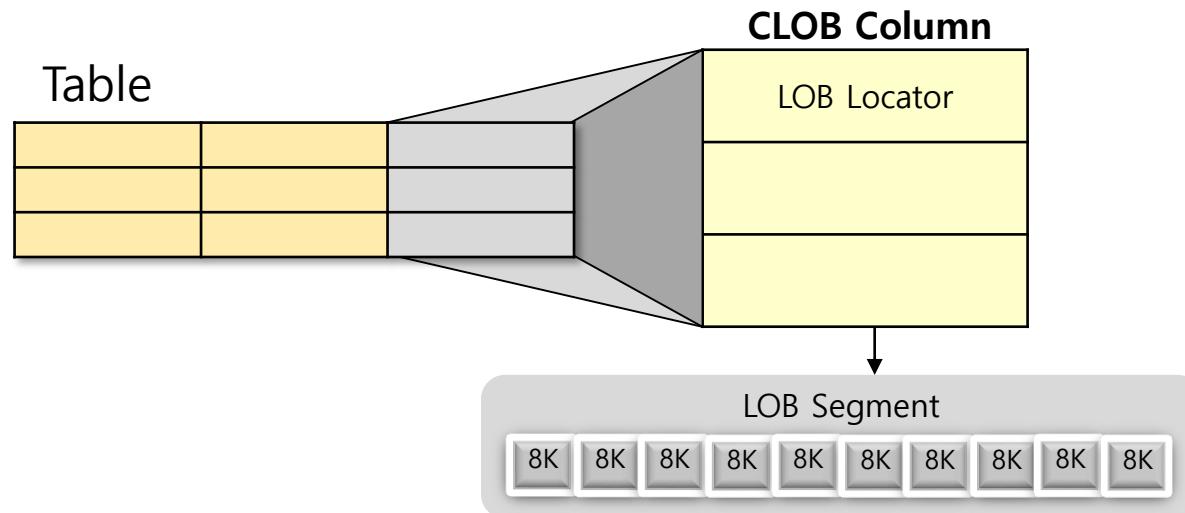
구분	Data Type	설명
대용량 객체형	CLOB, BLOB, XML TYPE	대용량 객체를 저장하기 위해 제공하는 가장 큰 데이터 타입이다.

● CLOB

- LONG 타입을 확장한 데이터 타입이다.

설명

- ✓ 최댓값은 4GB이다.
- ✓ 테이블 내에서 하나 이상의 컬럼에 선언할 수 있다.
- ✓ 데이터에 접근할 때, LONG 타입과 달리 임의의 위치에서 접근할 수 있다.
 - CLOB 타입의 컬럼 값은 같은 테이블의 다른 타입으로 선언된 컬럼 값과 동일한 디스크 블록에 저장되지 않는다.
 - 디스크 블록 내의 로우는 별도의 디스크 블록에 저장된 CLOB 타입의 포인터만 저장하고 있다.

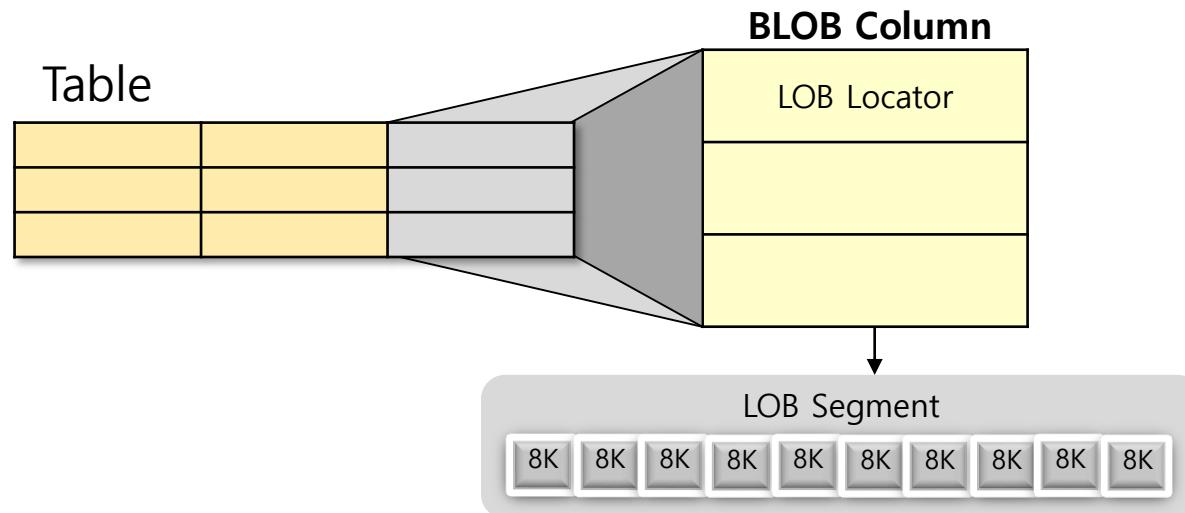


● BLOB

- LONG RAW를 확장한 데이터 타입으로, CLOB과 유사한 특징을 갖고 있다.

설명

- ✓ 최댓값은 4GB이다.
- ✓ 테이블 내에서 하나 이상의 컬럼에 선언할 수 있다.
- ✓ 데이터에 접근할 때, LONG RAW 타입과 달리 임의의 위치에서 접근할 수 있다.
 - BLOB 타입의 컬럼 값은 같은 테이블의 다른 타입으로 선언된 컬럼 값과 동일한 디스크 블록에 저장되지 않는다.
 - 디스크 블록 내의 로우는 별도의 디스크 블록에 저장된 BLOB 타입의 포인터만 저장하고 있다.



● XML TYPE

- XML (Extensible Markup Language)은 구조화 되거나 그렇지 않은 모든 데이터를 표현하기 위해 W3C (World Wide Web Consortium)에 의해 표준으로 제정된 형식이다.
- Tibero에서는 이 XML 데이터를 저장하기 위해 XMLTYPE 타입을 제공하며, 내부적으로 CLOB 형식으로 저장된다.

설명

- ✓ 데이터를 CLOB의 최대 크기까지 저장할 수 있다.
- ✓ 테이블 내에서 하나 이상의 컬럼에 선언할 수 있다.
- ✓ XML 데이터에 대한 접근, 추출, 질의를 수행할 때 사용한다.

●ROWID

- 데이터베이스 내의 각 로우를 식별하기 위해, Tibero가 각 로우마다 자동으로 부여하는 데이터 타입이다.
- 각 로우가 저장되어 있는 물리적인 위치를 포함하고 있다.

- 상수 값을 나타내는 단어이다.
- 상수란 변수에 대응되는 개념으로 말 그대로 변하지 않는 값을 의미한다.
- 문자열 리터럴은 작은따옴표를 사용하여 다른 스키마 객체와 구분한다.
- 리터럴은 SQL 문장에서 연산식이나 조건식의 일부로 사용할 수 있다.

구분	설명
문자열 리터럴	문자열을 표현할 때 사용하는 리터럴이다
숫자 리터럴	정수 또는 실수를 표현할 때 사용하는 리터럴이다.
간격 리터럴	특정 시간과 시간 사이의 간격을 표현하는 리터럴이다.
날짜형 리터럴	날짜와 시간 정보를 표현하는 리터럴이다.

● 문자열 리터럴

- 문자열을 표현할 때 사용하는 리터럴.
- 실제로 데이터베이스에 저장될 때는 고정 길이로 저장된다.

설명

- ✓ 최대 4,000byte까지 선언할 수 있다.
- ✓ 연산식이나 조건식에 문자열 리터럴이 사용되면, 문자열 리터럴은 CHAR 타입으로 취급된다.
- ✓ CHAR 타입의 데이터와 문자형 리터럴을 비교하는 경우
더 짧은 길이를 가진 데이터에 공백 문자를 삽입하여 비교한다.
- ✓ VARCHAR 타입의 데이터와 문자형 리터럴을 비교하는 경우
공백 문자를 삽입하지 않고 비교한다

- 문자열 리터럴 사용 예)

```
'Tibero'  
'Database'  
'2009/11/11'
```

● 숫자 리터럴

- 정수 또는 실수를 표현할 때 사용하는 리터럴이다.

설명

- ✓ 정수 리터럴과 실수 리터럴이 있다.
- ✓ 과학적 기수법(Scientific Notation)으로 표기할 수 있다.
- ✓ 예를 들어 8.33e-4는 0.000833을 8.33e+4는 83,300을 의미한다. e 대신 E를 사용해도 e를 사용할 때와 의미는 동일하다. e나 E 다음에 나오는 숫자는 지수를 나타낸다. 이때 지수는 -130 ~ 125 사이의 값이어야 한다.
- ✓ NUMBER 타입이 표현할 수 있는 최대 38자리의 정밀도를 초과하는 경우 숫자 리터럴은 NUMBER 타입의 최대 정밀도에 맞춘 후 맨 마지막 수를 버린다. 또한, NUMBER 타입이 표현할 수 있는 범위를 넘어서는 숫자 리터럴이 입력되면 에러가 발생한다.

- 숫자 리터럴 사용 예)

```
123  
+1.23  
0.123  
123e-123  
-123
```

● 간격 리터럴 (Interval literal)

- 특정 시간과 시간 사이의 간격을 표현하는 리터럴이다. 이러한 간격은 '연과 월'로 구성된 단위나 '날짜, 시간, 분, 초'로 구성된 단위 중 하나로 표현될 수 있다.

설명

- ✓ 두 가지 타입으로 지원한다.
 - YEAR TO MONTH
간격을 연과 월로 표현.
 - DAY TO SECOND
간격을 연과 월로 표현.

- YEAR TO MONTH 사용 예)

```
INTERVAL '12-3' YEAR TO MONTH
INTERVAL '123' YEAR(3)
INTERVAL '123' MONTH
INTERVAL '1' YEAR
INTERVAL '1234' MONTH(3)
```

- DAY TO SECOND 사용 예)

```
INTERVAL '1 2:3:4.567' DAY TO SECOND(3)
INTERVAL '1 2:3' DAY TO MINUTE
INTERVAL '123 4' DAY(3) TO HOUR
INTERVAL '123' DAY(3)
INTERVAL '12:34:56.1234567' HOUR TO SECOND(7)
INTERVAL '12:34' HOUR TO MINUTE
INTERVAL '12' HOUR
INTERVAL '12:34' MINUTE TO SECOND
INTERVAL '12' MINUTE
INTERVAL '12.345678' SECOND(2,6)
```

● 날짜형 리터럴

- 날짜형 리터럴은 날짜와 시간 정보를 표현하는 리터럴이다.

구분	설명
✓ DATE 리터럴	날짜와 시간 정보를 표현하는 날짜형 리터럴이다.
✓ TIME 리터럴	시간 정보를 표현하는 날짜형 리터럴이다.
✓ TIMESTAMP 리터럴	DATE 리터럴을 확장한 날짜형 리터럴이다.

● DATE 리터럴

- 날짜와 시간 정보를 표현하는 날짜형 리터럴이다.

설명

- ✓ 세기, 년, 월, 일, 시, 분, 초의 속성이 있다.
- ✓ 리터럴 변환

```
TO_DATE('2005/01/01 12:38:20', 'YY/MM/DD HH24:MI:SS')
```

- ✓ 한쪽에만 시간 정보가 있고 다른 쪽에는 시간 정보가 없을 경우 두 날짜가 같다고 비교하기 위해서는 시간 정보를 제거하고 비교해야 하는데 이때 TRUNC 함수를 사용하면 된다.

```
TO_DATE('2005/01/01', 'YY/MM/DD') =  
TRUNC(TO_DATE('2005/01/01 12:38:20', 'YY/MM/DD HH24:MI:SS'))
```

- ✓ ANSI 표현

```
DATE '2005-01-01'
```

- 시간 정보가 없다.
- 기본 형식은 'YYYY-MM-DD'이다.
- 구분자는 하이픈(-) 외에도 여러 가지가 있다.
- 슬래시(/), 별표(*), 점(.) 등이 있다.

● TIME 리터럴

- 시간 정보를 표현하는 날짜형 리터럴이다.

설명

- ✓ 시, 분, 초, 소수점 아래의 초의 속성이 있다.
- ✓ TO_TIME 함수를 사용하여 시간 값을 직접 지정하거나 문자 리터럴이나 숫자 리터럴로 표현된 시간 값을 TIME 리터럴로 변환할 수 있다.

```
TO_TIME('12:38:20.123456789', HH24:MI:SSXFF')
```

- ✓ 기본 시간 형식은 초기화 파라미터 파일에 NLS_TIME_FORMAT 파라미터로 정의되어 있다.
- ✓ ANSI 표현

```
TIME '10:23:10.123456789'  
TIME '10:23:10'  
TIME '10:23'  
TIME '10'
```

- 기본 형식은 'HH24:MI:SS.FF9'이다.
- 분 이하는 생략할 수 있다.

●TIMESTAMP 리터럴

-DATE 리터럴을 확장한 날짜형 리터럴이다.

설명

- ✓ '년, 월, 일'의 날짜와 '시, 분, 초', '소수점 아래의 초'의 속성이 있다.
- ✓ TIMESTAMP 리터럴로의 변환

```
TO_TIMESTAMP('09-Aug-01 12:07:15.50', 'DD-Mon-RR HH24:MI:SS.FF')
```

- ✓ 기본 시간 형식은 초기화 파라미터 파일에 NLS_TIMESTAMP_FORMAT 파라미터로 정의되어 있다.
- ✓ ANSI 표현

```
TIMESTAMP '2005/01/31 08:13:50.112'  
TIMESTAMP '2005/01/31 08:13:50'  
TIMESTAMP '2005/01/31 08:13'  
TIMESTAMP '2005/01/31 08'  
TIMESTAMP '2005/01/31'
```

- 기본 형식은 'YYYY/MM/DD HH24:MI:SSxFF' 이다.
- 날짜 부분('YYYY/MM/DD') 이외에는 생략할 수 있다.
- 소수점 아래의 초('FF') 부분은 0~9자리까지 표현할 수 있다.

● 형식 문자열

- 형식 문자열이란 NUMBER 타입과 날짜형 타입의 값을 문자열로 변환하기 위한 형식을 정의한 것이다.
- 디폴트 시간 형식으로 되어 있지 않은 문자열이나 숫자 이외의 문자를 포함하는 문자열은 각각 날짜형 또는 NUMBER 타입의 값으로 변환할 수 없다. 이러한 경우 반드시 TO_DATE, TO_NUMBER 등의 변환 함수를 사용해야 한다.
- 형식 문자열은 TO_CHAR, TO_DATE, TO_NUMBER 함수의 파라미터로 사용된다. 만약 함수 파라미터로 형식 문자열이 주어지지 않으면, 디폴트 형식을 사용하여 변환한다.
- 형식 문자열은 종류
 - NUMBER 타입의 형식 문자열
 - 날짜형 타입의 형식 문자열

● NUMBER 타입의 형식 문자열

함수	설명
TO_CHAR	NUMBER 타입의 값을 문자열로 변환한다.
TO_NUMBER	문자열을 NUMBER 타입의 값으로 변환한다.

-NUMBER 타입의 값을 문자열로 변환하기 위한 형식

NUMBER 타입의 값	형식 문자열	출력 결과
0	99.99	' .00'
0.1	99.99	' .10'
-0.1	99.99	' -.10'
0	90.99	' 0.00'
0.1	90.99	' 0.10'
-0.1	90.99	' -0.10'
0	9999	' 0'
1	9999	' 1'
0.1	9999	' 0'
-0.1	9999	' -0'

● 날짜형 타입의 형식 문자열

함수	설명
TO_CHAR	날짜형 타입의 값을 문자열로 변환한다.
TO_DATE	문자열을 날짜형 타입의 값으로 변환한다.

- 날짜형 타입의 값을 문자열로 변환하기 위한 형식

NUMBER 타입의 값	형식 문자열	출력 결과
Q	아니오	분기를 출력한다. (1-4)
RM	예	달을 로마 숫자로 출력한다. (I-XII)
RR	예	두 자릿수의 연도의 입력 값에 따라 몇 세기인지 자동으로 조절한다.
RRRR	예	반올림한 연도. 4자리 혹은 2자리를 입력 받는다. 2자리로 입력했을 경우는 RR과 똑같이 동작한다.
SS	예	초를 출력한다. (0-59)
SSSSS	예	자정을 기준으로 현재 몇 초인지 출력한다. (0-86399)
WW	아니오	1년 중 몇 번째 주인지 출력한다. (1-53). 첫 번째 주는 1월 1일에 시작하고 1월 7일에 끝난다.
W	아니오	1개월 중 몇 번째 주인지 출력한다. (1-5). 첫 번째 주는 그 달 1일에 시작하고 그 달 7일에 끝난다.
X	예	점(.)을 출력한다.

● 날짜형 타입의 형식 문자열

함수	설명
TO_CHAR	날짜형 타입의 값을 문자열로 변환한다.
TO_DATE	문자열을 날짜형 타입의 값으로 변환한다.

- 날짜형 타입의 값을 문자열로 변환하기 위한 형식

NUMBER 타입의 값	형식 문자열	출력 결과
Q	아니오	분기를 출력한다. (1-4)
RM	예	달을 로마 숫자로 출력한다. (I-XII)
RR	예	두 자릿수의 연도의 입력 값에 따라 몇 세기인지 자동으로 조절한다.
RRRR	예	반올림한 연도. 4자리 혹은 2자리를 입력 받는다. 2자리로 입력했을 경우는 RR과 똑같이 동작한다.
SS	예	초를 출력한다. (0-59)
SSSSS	예	자정을 기준으로 현재 몇 초인지 출력한다. (0-86399)
WW	아니오	1년 중 몇 번째 주인지 출력한다. (1-53). 첫 번째 주는 1월 1일에 시작하고 1월 7일에 끝난다.
W	아니오	1개월 중 몇 번째 주인지 출력한다. (1-5). 첫 번째 주는 그 달 1일에 시작하고 그 달 7일에 끝난다.
X	예	점(.)을 출력한다.

● 의사 컬럼 이란?

- 사용자가 명시적으로 선언하지 않아도 시스템이 자동으로 모든 테이블에 포함시키는 컬럼을 말한다.
- 종류
 - CONNECT_BY_IS_LEAF
 - ROWID
 - ROWNUM
 - LEVEL

```
SELECT rowid, rownum, emp_id
FROM employee;
```

Grid Result

	ROWID	ROWNUM	EMP_ID
1	AAAAUXAACAAAAABPAAK	1	1998232
2	AAAAUXAACAAAAABPAAL	2	1998234
3	AAAAUXAACAAAAABPAAJ	3	1999235
4	AAAAUXAACAAAAABPAAA	4	2001043
5	AAAAUXAACAAAAABPAAO	5	2001267
6	AAAAUXAACAAAAABPAAI	6	2001397

Grid Result Text Output Explain Plan

Ln 6, Col 18 OVF 0.03 sec, 16 rows

AutoCommit is ON CAP NUM OVR

●CONNECT_BY_IS_LEAF 의사 컬럼

- 현재 로우가 CONNECT BY 조건에 의해 정의된 트리(Tree)의 리프(Leaf)이면 1을 반환하고 그렇지 않을 경우에는 0을 반환한다. 이 정보는 해당 로우가 계층 구조(Hierarchy)를 보여주기 위해 확장될 수 있는지 없는지를 나타낸다.
- 다음은 CONNECT_BY_IS_LEAF 의사 컬럼을 사용한 예이다.

```
SQL> SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL, SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
  FROM EMP2
    START WITH ENAME = 'Clark'
    CONNECT BY PRIOR EMPNO = MGRNO
    ORDER BY ENAME;
```

ENAME	CONNECT_BY_ISLEAF	LEVEL	PATH
Alicia	1	3	-Clark-Martin-Alicia
Allen	1	3	-Clark-Ramesh-Allen
Clark	0	1	-Clark
James	1	3	-Clark-Martin-James
John	0	3	-Clark-Ramesh-John
Martin	0	2	-Clark-Martin
Ramesh	0	2	-Clark-Ramesh
Ward	1	4	-Clark-Ramesh-John-Ward

●ROWID 의사 컬럼

- 전체 데이터베이스 내의 하나의 로우를 유일하게 참조하는 식별자이다. ROWID는 그 로우의 디스크의 물리적인 위치를 가리키고 있으며, 그 로우가 삭제될 때까지 변화되지 않는다.

Segment#	Data File#	Data Block#	Row#
4 byte	2 byte	4 byte	2 byte

- ROWID 값을 표현하기 위한 포맷으로는 BASE64 인코딩을 이용한다. BASE64 인코딩은 6bit에 포함된 숫자를 8bit 문자로 나타내는 방식으로, 0 ~ 63까지의 숫자를 A ~ Z, a ~ z, 0 ~ 9, +, /로 대치한다.
- ROWID를 BASE64 인코딩으로 변환하면 세그먼트#, 데이터 파일#, 데이터 블록#, 로우#가 각각 6, 3, 6, 3byte로 되고, 'SSSSSSFFFFBBBBBBRRR'의 형태를 갖는다. 예를 들어, 세그먼트# = 100, 데이터 파일# = 20, 데이터 블록# = 250, 로우# = 0인 ROWID는 'AAAABkAAUAAAAD6AAA'로 나타낸다.

● ROWNUM 의사 컬럼

- SELECT 문장의 실행 결과로 나타나는 로우에 대하여 순서대로 번호를 부여한다.
- ROWNUM이 할당되는 순서
 - ① 질의를 수행한다.
 - ② 질의 결과로 로우가 생성된다.
 - ③ 로우를 반환하기 직전에 그 로우에 ROWNUM이 할당된다. Tiberio는 내부적으로 ROWNUM 카운터를 가지고 있으며, 카운터 값을 질의 결과의 로우에 할당한다.
 - ④ ROWNUM을 할당 받은 로우에 ROWNUM에 대한 조건식을 적용한다.
 - ⑤ 조건식을 만족하면 할당된 ROWNUM이 확정되고, 내부의 ROWNUM 카운터의 값이 1로 증가한다.
 - ⑥ 조건식을 만족하지 않으면 그 로우는 버려지고, 내부의 ROWNUM 카운터의 값은 증가하지 않는다.
- 10개의 로우만을 반환하는 예

```
SELECT * FROM EMP WHERE ROWNUM <= 10;
```

● ROWNUM 의사 컬럼

-주의 사항

- WHERE 절을 포함하는 모든 부질의를 처리한 다음에 ORDER BY 절을 처리한다.
따라서 ORDER BY 절을 이용해서 항상 같은 결과를 얻을 수는 없다.
- 예를 들어, 다음의 질의는 실행할 때마다 다른 결과를 얻는다.

```
SELECT * FROM EMP WHERE ROWNUM <= 10 ORDER BY EMPNO;
```

- 다음과 같이 변환하면 ORDER BY 절을 먼저 처리하게 되므로 항상 같은 결과를 얻을 수 있다.

```
SELECT * FROM (SELECT * FROM EMP ORDER BY EMPNO)  
WHERE ROWNUM <= 10;
```

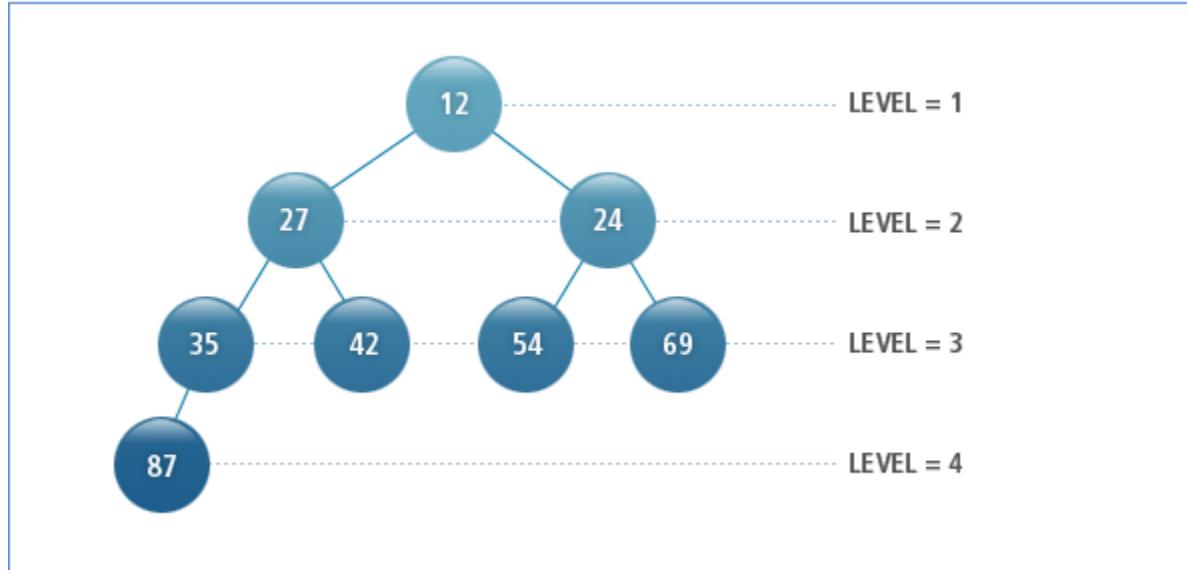
- 다음과 같은 SELECT 문장은 하나의 로우도 반환하지 않는다.

```
SELECT * FROM EMP WHERE ROWNUM > 1;
```

- ROWNUM 값이 확정되기 전에 ROWNUM에 대한 조건식이 수행되기 때문에 첫 번째
로우가 ROWNUM = 1 으로 조건식을 만족하지 않고, 두 번째 결과 로우도 ROWNUM
= 1이므로 반환되지 않는다.

● LEVEL 의사 컬럼

-계층 질의(Hierarchical Query) 를 실행한 결과에 각 로우의 트리 내 계층을 출력하기 위한 컬럼 타입이다. 최상위 로우의 LEVEL 값은 1이며, 하위 로우로 갈수록 1씩 증가한다.



●NULL

- 어떤 컬럼에 값이 없을 때 그 컬럼을 NULL이라고 하거나 NULL 값을 가진다고 한다.
- NULL은 NOT NULL 제약과 PRIMARY KEY 제약이 걸리지 않은 모든 데이터 타입의 칼럼에 포함될 수 있다.
- 실제 값을 모르거나 아무런 의미 없는 값이 필요할 때 사용할 수 있다. NULL과 0은 다르기 때문에 NULL을 0으로 나타내면 안 된다. 다만 문자 타입의 컬럼에 빈 문자열("")이 들어가면 NULL로 처리된다.
- 다음 연산의 결과는 NULL이다.

NULL + 1 = NULL

DATA = {1000, 500, NULL, NULL, 1500}
AVG(DATA) = (1000 + 500 + 1500) / 3 = 1000

● NULL에 대한 비교조건

- NULL을 검사할 수 있는 비교조건은 IS NULL과 IS NOT NULL만 가능하다. NULL은 데이터가 없다는 것을 의미한다. 때문에 NULL과 NULL, NULL과 NULL이 아닌 다른 값을 서로 비교할 수 없다.
- 다만 DECODE 함수에서는 두 개의 NULL을 비교할 수 있다.

```
SQL> SELECT DECODE(NULL, NULL, 1) FROM DUAL;
```

```
DECODE(NULL,NULL,1)
```

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

- 만일 NULL에 다른 비교조건을 사용하면, 결과는 UNKNOWN으로 나타난다. UNKNOWN으로 판별되는 조건은 거의 대부분 FALSE처럼 처리된다.
- 그 예로 SELECT 문에서 WHERE 절에 UNKNOWN으로 판별되는 조건이 있을 경우 반환되는 로우가 없다. 하지만 UNKNOWN이 FALSE와 다른 점은 UNKNOWN 조건에 또 다른 연산자가 더해져도 결과는 UNKNOWN이라는 점이다.

● 주석

- SQL문장과 데이터베이스 오브젝트에 대한 주석을 작성할 수 있다.
- HINT를 제외한 주석은 SQL 문장의 실행에 영향을 주지 않는다.
- 주석시작 /*, 주석 끝 */ 으로 표기하고, 따로 공백이나 줄 바꿈으로 내용과 구분할 필요는 없다.
- '--'로 주석의 시작을 나타내고 바로 뒤에 주석의 내용을 적는다.

```
SELECT emp_id, emp_name,          /* 부서가 총무과인 직원의 명단을 출력한다. */  
       e.dept_id  
FROM emp e, dept d            -- 테이블  
WHERE e.dept_id = d.dept_id  
AND  d.dept_name = '총무과'  
AND  e.status != 1;           /* 퇴사한 사람 제외 */
```

- 스키마 객체에도 주석을 삽입할 수 있다. 즉 COMMENT 명령을 사용하여 스키마 객체인 테이블, 뷰, 컬럼에 주석을 삽입할 수 있다. 스키마 객체에 삽입된 주석은 데이터 사전에 저장된다.

● 힌트(HINT)

- SQL문에 주석을 추가하여 Optimizer의 특정 행동을 지시 할 수 있다.
- 힌트는 반드시 SELECT, INSERT, UPDATE, DELETE 키워드 뒤에만 사용할 수 있다.
- 문법에 맞지 않는 힌트는 주석으로 취급한다.

```
(DELETE|INSERT|SELECT|UPDATE) /*+ hint [hint] ... */
```

```
(DELETE|INSERT|SELECT|UPDATE) --+ hint [hint] ...
```

- 힌트를 사용할 때 주의할 점

- 힌트는 반드시 DELETE, INSERT, SELECT, UPDATE 절 뒤에만 올 수 있다.
- '+' 기호는 반드시 주석 구분자('/*' 또는 '--') 바로 뒤에 공백 없이 붙여 써야 한다.
- 힌트와 '+'기호 사이에 공백은 있어도 되고, 없어도 된다.
- 문법에 맞지 않는 힌트는 주석으로 취급되며, 에러는 발생하지 않는다.

● 힌트(HINT)의 종류

구분	힌트
질의 변형	NO_MERGE, UNNEST, NO_UNNEST,
최적화 방법	ALL_ROWS, FIRST_ROWS
접근 방법	FULL, INDEX, NO_INDEX, INDEX_ASC, INDEX_DESC, INDEX_FFS, NO_INDEX_FFS
조인 순서	LEADING, ORDERED,
조인 방법	USE_NL, NO_USE_NL, USE_NL_WITH_INDEX, USE_MERGE, NO_USE_MERGE, USE_HASH, NO_USE_HASH
병렬 처리	PARALLEL, NO_PARALLEL, PQ_DISTRIBUTE
실체화 뷰	REWRITE, NO_REWRITE
기타	APPEND, NOAPPEND

● 힌트(HINT) - 질의 변형

구분	힌트
NO_MERGE	<ul style="list-style-type: none"> ✓ 특정 뷰에 대해 뷰 병합을 하지 않도록 지시하는 힌트이다. Tibero에서는 뷰 병합이 디폴트로 수행되며, 뷰가 병합이 가능할 경우 상위의 질의 블록과 결합해 하나의 질의 블록을 형성한다. NO_MERGE 힌트를 사용하면 이렇게 디폴트로 수행되는 뷰의 병합을 막을 수 있다. <pre>SELECT * FROM T1, (SELECT /*+ NO_MERGE */ * FROM T2, T3 WHERE T2.A = T3.B) V WHERE T1.C = V.D</pre> <ul style="list-style-type: none"> ✓ 힌트가 없었다면 뷰가 병합되어 질의 최적화기에서 테이블 T1, T2, T3에 대한 조인 순서와 조인 방법을 고려하게 되지만, 위와 같이 힌트가 있을 경우는 뷰가 병합되지 못하기 때문에 T2와 T3가 먼저 조인되고, 그 이후에 T1이 조인된다.
UNNEST	Subquery 를 언네스팅하도록 지시하는 힌트이다. 디폴트로 수행하지만, 특정 쿼리만 언네스팅을 하려면 초기화 파라미터에서 언네스팅을 해제하면 된다. 그러면 UNNEST 힌트를 이용할 수 있다. UNNEST 힌트는 부질의 블록에 명시한다.
NO_UNNEST	Subquery 언네스팅을 수행하지 않도록 지시하는 힌트이다. 언네스팅을 디폴트로 수행하며 언네스팅이 가능한 경우 부질의를 조인으로 변환한다. 이때 NO_UNNEST 힌트를 사용해서 언네스팅을 막을 수 있다. NO_UNNEST 힌트는 부질의 블록에 명시한다.

● 힌트(HINT) - 최적화 방법

- 처리 과정과 결과 표시를 최적화할 수 있다. 만약 최적화 방법이 적용된 힌트가 사용된 질의가 있다면 해당 질의에 대해서는 통계 정보와 초기화 파라미터의 최적화 방법(OPTIMIZER MODE)의 값이 없는 것처럼 처리된다.

구분	힌트
ALL_ROWS	최소한의 리소스를 사용하여 전체 결과에 대한 처리량이 가장 많도록 처리과정의 최적화 방법을 선택하는 힌트이다.
FIRST_ROWS	첫 로우부터 파라미터로 입력된 번호의 로우까지 가장 빠르게 보여줄 수 있도록 결과 표시의 최적화 방법을 선택하는 힌트이다.

● 힌트(HINT) - 접근 방법

- 접근 방법이 적용된 힌트는 질의 최적화기가 특정 접근 방법의 사용이 가능한 경우, 그 방법을 사용하도록 명시한다. 만일 힌트에서 명시한 방법을 사용할 수 없는 경우에는 질의 최적화기는 그 힌트를 무시한다.
- 힌트에 명시하는 테이블명은 SQL 문에서 사용하는 이름과 동일해야 한다. 즉, 테이블 이름에 대한 별칭을 사용하였다면, 테이블 이름 대신에 별칭을 사용하여야 한다.
- SQL 문에서 테이블 이름에 스키마 이름을 포함하여 명시하였더라도 힌트에서는 테이블 이름만을 명시하여야 한다.

● 힌트(HINT) - 접근 방법 종류

구분	힌트
FULL	전체 테이블을 스캔하도록 지시하는 힌트이다. WHERE 절에 명시된 조건식에 맞는 인덱스가 있더라도 전체 테이블 스캔을 사용한다.
INDEX	명시한 인덱스를 사용하여 인덱스 스캔을 하도록 지시하는 힌트이다.
NO_INDEX	명시한 인덱스를 사용하는 인덱스 스캔을 하지 않도록 지시하는 힌트이다. 만일 NO_INDEX 힌트와 INDEX 또는 INDEX_ASC, INDEX_DESC 힌트가 동일한 인덱스를 명시한다면 질의 최적화기는 이 두 힌트를 모두 무시한다.
INDEX_ASC	명시한 인덱스를 사용하여 인덱스 스캔을 하도록 지시하는 힌트이다. 만일 인덱스 범위 스캔을 사용하는 경우에는 인덱스를 오름차순으로 스캔하도록 한다. 현재 Tibero의 인덱스 스캔의 기본 동작이 오름차순이기 때문에 INDEX_ASC는 INDEX와 동일한 작업을 수행한다. 분할된 인덱스의 경우 분할된 각 영역 내에서 오름차순으로 스캔한다.
INDEX_DESC	명시한 인덱스를 사용하여 인덱스 스캔을 하도록 지시하는 힌트이다. 만일 인덱스 범위 스캔을 사용하는 경우에는 인덱스를 내림차순으로 스캔하도록 한다. 분할된 인덱스의 경우 분할된 각 영역 내에서 내림차순으로 스캔한다.
INDEX_FFS	명시한 인덱스를 사용하는 빠른 전체 인덱스 스캔을 사용하지 않도록 지시하는 힌트이다.

● 힌트(HINT) - 조인 순서

- LEADING, ORDERED는 조인 순서를 결정하는 힌트이다.
- LEADING 힌트가 ORDERED보다 질의 최적화기를 선택할 수 있는 폭이 넓어서 LEADING을 사용하는 것이 좋다.

구분	힌트
LEADING	<ul style="list-style-type: none">✓ LEADING은 조인에서 먼저 조인되어야 할 테이블의 집합을 명시하는 힌트이다.✓ LEADING 힌트가 먼저 조인될 수 없는 테이블을 포함하는 경우 무시된다.✓ LEADING 힌트끼리 충돌하는 경우에는 LEADING, ORDERED 힌트가 모두 무시된다. 만일 ORDERED 힌트가 사용되는 경우에는 LEADING 힌트는 모두 무시된다.
ORDERED	<ul style="list-style-type: none">✓ ORDERED는 테이블을 FROM 절에 명시된 순서대로 조인하도록 지시하는 힌트이다.✓ 질의 최적화기는 조인의 결과 집합의 크기에 대한 정보를 추가로 알고 있다.✓ 사용자가 그 정보를 통해 질의 최적화기의 조인 순서를 명확히 알고 있을 경우에만 ORDERED 힌트를 사용하는 것이 좋다.

● 힌트(HINT) – 병렬 처리

구분	힌트
PARALLEL	<ul style="list-style-type: none">✓ 지정한 개수의 스레드를 사용해 질의의 수행을 병렬로 진행하도록 지시하는 힌트이다.
NO_PARALLEL	<ul style="list-style-type: none">✓ 질의의 수행을 병렬로 진행하지 않도록 지시하는 힌트이다.
PQ_DISTRIBUTE	<ul style="list-style-type: none">✓ 조인을 포함한 질의의 병렬 처리에서 조인될 로우의 분산 방법을 지시하는 힌트이다.✓ 분산 방법으로는 HASH-HASH, BROADCAST-NONE, NONE-BROADCAST, NONE-NONE이 있으며 특정한 분산 방법을 선택함으로써 병렬 처리에서 조인의 성능을 향상시킬 수 있다.

● 힌트(HINT) – 실체화 뷰

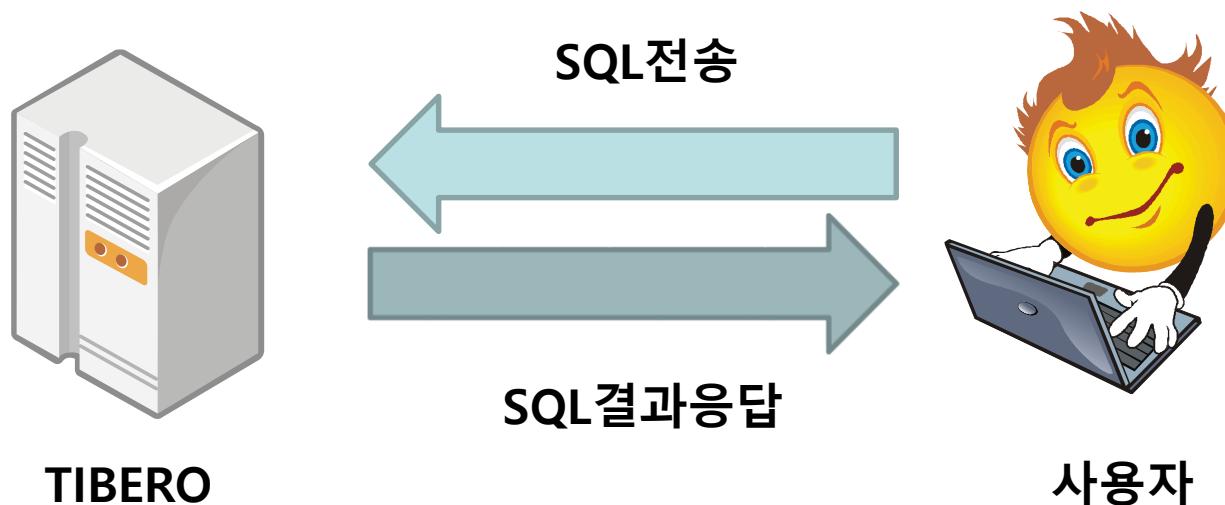
구분	힌트
REWRITE	<ul style="list-style-type: none">✓ REWRITE는 해당 질의 블록에서 비용의 비교 없이 실체화 뷰를 사용하여 질의의 다시 쓰기를 하도록 지시하는 힌트이다.✓ 따라서 최종으로는 REWRITE 힌트가 사용된 질의 블록만 다시 쓰기를 한 결과와 모든 블록에서 다시 쓰기를 한 결과의 비용을 비교해서 더 좋은 쪽을 질의 최적화기가 선택하게 된다.✓ 그리고 실체화 뷰의 목록이 명시된 경우에는 목록에 있는 실체화 뷰만 사용하여 질의의 다시 쓰기를 시도한다.
NO_REWRITE	<ul style="list-style-type: none">✓ NO_REWRITE는 해당 질의 블록에서는 질의의 다시 쓰기를 하지 않도록 지시하는 힌트이다.

● 힌트(HINT) – 기타

구분	힌트
APPEND	<ul style="list-style-type: none">✓ APPEND는 DML 문장에서 직접 데이터 파일에 추가하는 삽입 방법 즉 Direct-Path 방식을 수행하도록 지시하는 힌트이다.✓ Direct-Path 방식은 일반적인 삽입 방법과 달리 항상 새로운 데이터 블록을 할당받아서 데이터 삽입을 수행하며, 버퍼 캐시를 이용하지 않고 직접 데이터 파일을 추가하기 때문에 성능 향상에 많은 이점이 있다.
NOAPPEND	<ul style="list-style-type: none">✓ NOAPPEND는 DML 문장에서 Direct-Path 방식을 수행하지 않도록 지시하는 힌트이다.

Chapter 7장 데이터 조회.

- SQL (Structure Query Language) : 구조 질의 언어
- 데이터베이스와 소통을 위해 필요한 언어



DML

- 데이터베이스 객체 내의 데이터 조회 및 삽입, 삭제, 변경을 위한 문장.
- SELECT, INSERT, UPDATE, DELETE

DDL

- 데이터베이스 객체를 생성, 변경, 제거하기 위한 문장.
- CREATE, ALTER, DROP, RENAME, TRUNCATE

DCL

- 데이터 및 객체들에 대한 작업등에 대한 권한을 부여하고 취소하기 위한 문장
- GRANT, REVOKE

Transaction Control

- Transaction 종료 및 취소하기 위한 문장
- COMMIT, ROLLBACK, SAVEPOINT

●SQL : 문법

```
SELECT [DISTINCT] 컬럼명1, 컬럼명2,...  
FROM 테이블명;
```

●Query 및 결과 예제

```
SELECT *  
FROM employee;
```

Grid Result						
	EMP_ID	EMP_NAME	HIREDATE	SALARY	COMM	DEPT_ID
1	2001043	STEVE	2001/03/01 09:00:00	1800	1000	10
2	2003122	PAUL	2003/04/11 09:00:00	3000	1000	10
3	2004001	MICHAEL	2004/01/02 09:00:00	4000	500	10
4	2004098	DAVID	2004/03/01 09:00:00	3000	100	10
5	2005287	JOHN	2005/06/20 09:00:00	5000	400	10
6	2006121	LINCOLN	2006/04/01 09:00:00	2800	2000	10
7	2007112	FERNANDO	2007/03/13 09:00:00	3400	1500	10
8	2006246	WARDEN	2006/10/21 09:00:00	6000	2100	10
9	2001397	JANE	2001/05/03 09:00:00	2000		10
10	1999235	THEODORE	1999/08/21 09:00:00	3500		20
11	1998232	KAREN	1998/11/01 09:00:00	4000	1000	20

●SQL : 문법

WHERE 조건(Condition)

●Query 및 결과 예제

```
SELECT emp_id, emp_name, hiredate  
FROM employee  
WHERE emp_name = 'PAUL';
```

Grid Result			
	EMP_ID	EMP_NAME	HIREDATE
1	2003122	PAUL	2003/04/11 09:00:00

Grid Result | Text Output | Explain Plan

Ln 3, Col 27 | OVF 0,01 sec. | 1 rows

AutoCommit is ON | CAP | NUM | OVR

●SQL : 문법

WHERE 조건(Condition) AND 조건(Condition)

●Query 및 결과 예제

```
SELECT emp_id, emp_name, salary  
FROM employee  
WHERE salary >= 3000 AND salary <= 4000;
```

Grid Result			
	EMP_ID	EMP_NAME	SALARY
1	2001267	WAYNE	3400
2	2005853	CHARLES	4000
3	2003465	SANDRA	3800
4	1998232	KAREN	4000
5	1999235	THEODORE	3500
6	2007112	FERNANDO	3400

Grid Result Text Output Explain Plan

Ln 6, Col 23 OVR 0,03 sec. 10 rows

AutoCommit is ON CAP NUM OVR

●SQL : 문법

WHERE 조건(Condition) OR 조건(Condition)

●Query 및 결과 예제

```
SELECT *
FROM employee
WHERE dept_id = 10 OR dept_id = 30;
```

Grid Result

	EMP_ID	EMP_NAME	HIREDATE	SALARY	COMM	DEPT_ID	
7	2007112	FERNANDO	2007/03/13 09:00:00	3400	1500	10	
8	2006246	WARDEN	2006/10/21 09:00:00	6000	2100	10	
9	2001397	JANE	2001/05/03 09:00:00	2000		10	
10	2001267	WAYNE	2001/04/12 09:00:00	3400	1300	30	
11	2002658	JAMES	2002/12/29 09:00:00	4400	1500	30	
12	2009001	test	2009/09/30 08:30:47			10	

Grid Result Text Output Explain Plan

Ln 3, Col 37 OVR 0,09 sec. 13 rows

AutoCommit is ON CAP NUM OVR

●SQL : 문법

WHERE 컬럼명 BETWEEN value1 AND value2

●Query 및 결과 예제

```
SELECT *
FROM employee
WHERE salary BETWEEN 3000 AND 4000;
```

Grid Result						
	EMP_ID	EMP_NAME	HIREDATE	SALARY	COMM	DEPT_ID
1	2001267	WAYNE	2001/04/12 09:00:00	3400	1300	30
2	2005853	CHARLES	2005/12/25 09:00:00	4000		20
3	2003465	SANDRA	2003/07/07 09:00:00	3800	2000	20
4	1998232	KAREN	1998/11/01 09:00:00	4000	1000	20
5	1999235	THEODORE	1999/08/21 09:00:00	3500		20
6	2007112	FERNANDO	2007/03/13 09:00:00	3400	1500	10

Grid Result Text Output Explain Plan

Ln 7, Col 27 OVF 0,06 sec. 10 rows

AutoCommit is ON CAP NUM OVR

●SQL : 문법

WHERE 컬럼명 IN (value1, value2,...)

●Query 및 결과 예제

```
SELECT *
FROM employee
WHERE emp_name IN ('STEVE','PAUL');
```

Grid Result

	EMP_ID	EMP_NAME	HIREDATE	SALARY	COMM	DEPT_ID
1	2001043	STEVE	2001/03/01 09:00:00	1800	1000	10
2	2003122	PAUL	2003/04/11 09:00:00	3000	1000	10

Grid Result | Text Output | Explain Plan

Ln 4, Col 1 OVF 0,05 sec. 2 rows

AutoCommit is ON CAP NUM OVR

●SQL : 문법

WHERE 컬럼명 IS [NOT] NULL

●Query 및 결과 예제

```
SELECT *
FROM employee
WHERE comm IS NULL;
```

Grid Result

	EMP_ID	EMP_NAME	HIREDATE	SALARY	COMM	DEPT_ID
1	2001397	JANE	2001/05/03 09:00:00	2000		10
2	1999235	THEODORE	1999/08/21 09:00:00	3500		20
3	1998234	BILL	1998/11/05 09:00:00	2900		20
4	2005853	CHARLES	2005/12/25 09:00:00	4000		20

Grid Result Text Output Explain Plan

Ln 3, Col 1 OVF 0,05 sec. 4 rows

AutoCommit is ON CAP NUM OVR

●SQL : 문법

WHERE 컬럼명 LIKE [%]value1[%]

●Query 및 결과 예제

```
SELECT *
FROM employee
WHERE emp_name LIKE 'J%';
```

Grid Result

	EMP_ID	EMP_NAME	HIREDATE	SALARY	COMM	DEPT_ID
1	2002658	JAMES	2002/12/29 09:00:00	4400	1500	30
2	2001397	JANE	2001/05/03 09:00:00	2000		10
3	2005287	JOHN	2005/06/20 09:00:00	5000	400	10

Grid Result Text Output Explain Plan

Ln 4, Col 1 OVF 0,03 sec. 3 rows

AutoCommit is ON CAP NUM OVR

=

- 동일함 (같음)을 테스트 한다.

!=, <>, ^=

- 동일하지 않음(같지 않음)을 테스트 한다.

>=, >, <=, <

- 큰 경우 혹은 크거나 같은 경우의 비교 테스트를 한다.
- 이와 반대의 경우도 적용된다.

AND

- 'A AND B'의 경우 두 조건 모두를 만족해야 한다.

OR

- 'A OR B'의 경우 두 조건 중 하나를 만족해야 한다.

BETWEEN AND

- 주어진 범위의 값에 해당하면 그 값을 반환한다.

IN

- 하나 이상의 동일 조건을 만족해야 한다.

LIKE

- 조건의 일부와 일치하는 경우에 적용된다.

IS NULL
(IS NOT NULL)

- 할당되지 않은 값을 만족하는 경우이다.
- 할당된 값의 경우 NOT NULL을 사용한다.

●SQL : 문법

WHERE 컬럼명 ORDER BY [ASC|DESC]

●Query 및 결과 예제

```
SELECT *
FROM employee
WHERE salary >= 5000
ORDER BY salary;
```

Grid Result

	EMP_ID	EMP_NAME	HIREDATE	SALARY	COMM	DEPT_ID
1	2005287	JOHN	2005/06/20 09:00:00	5000	400	10
2	2006246	WARDEN	2006/10/21 09:00:00	6000	2100	10

Grid Result Text Output Explain Plan

Ln 5, Col 1 OVF 0.03 sec. 2 rows

AutoCommit is ON CAP NUM OVR

- GROUP BY 절과 집합 함수를 이용하여 다양한 데이터 조회 가능
- GROUP BY 절 이후의 컬럼 리스트 별로 GROUP을 묶어 조회한다.
- Query 및 결과 예제

```
SELECT dept_id, count(*)
FROM employee
GROUP BY dept_id;
```

Grid Result

DEPT_ID	COUNT(*)
1	10
2	20
3	30

Grid Result | Text Output | Explain Plan

Ln 4, Col 1 OVF 0,05 sec. 3 rows

AutoCommit is ON CAP NUM OVR

- GROUP BY 절과 결과로 부터 특정 조건을 만족하는 값을 얻고자 할 경우 HAVING절을 사용

- Query 및 결과 예제

```
SELECT dept_id, count(*)  
FROM employee  
GROUP BY dept_id  
HAVING count(*) >= 5;
```

Grid Result

DEPT_ID	COUNT(*)
1	10
2	20

Ln 4, Col 1 OVF 0,03 sec. 2 rows

AutoCommit is ON CAP NUM OVR

Grid Result Text Output Explain Plan

●데이터 타입

타입종류	데이터 타입	길이	설명
문자타입	CHAR	최대 2000 Byte	고정 길이 문자 데이터 타입
	VARCHAR	최대 4000 Byte	가변 길이 문자 데이터 타입
	RAW	최대 2000 Byte	임의의 바이너리 데이터를 저장하기 위한 데이터 타입
	LONG	최대 2GB	가변 길이 문자 데이터 타입
	LONG RAW	최대 2GB	바이너리 데이터 저장 타입
숫자타입	NUMBER	1~38자리, -37~37	정수 또는 실수를 저장하기 위한 데이터 타입
	INTEGER,FLOAT		NUMBER 타입과 동일
날짜타입	DATE	날짜 + 시간(초)	고정 길이 날짜 데이터 타입
	TIME	시간(소수점9자리)	
	TIMESTAMP	날짜+시간(소수점9자리)	고정 길이 시간 데이터 타입
대용량객체 타입	BLOB	최대 4GB	가변 길이 원시 이진 데이터
	CLOB	최대 4GB	가변 길이 문자 데이터

●데이터 형식 (NUMBER)

형식요소	예제	결과	설명
,(쉼표) or G	9,999 9G999	1,234	해당 위치에 쉼표를 표기한다.
.(마침표) or D	9,999.99	1,234.00	해당 위치에 소수점을 표기하고 지정된 자릿수에 데이터가 없으면 0을 표기한다.
\$	\$9,999	\$1,234	숫자의 시작에 달러 문자를 표기한다.
0	00,999	01,234	해당되는 위치에 값이 없을 경우 0을 표기한다.
MI	9999MI	-1234 1234-	음수 부호를 표기하는 형식 요소로 문자를 숫자로 변환 시엔 음수기호를 앞에 표기해 주고 숫자를 문자로 변환 시엔 음수 기호를 뒤에 표기한다.
PR	9999PR	<1234>	음수에 대해서 숫자를 문자로 변환할 때만 음수 표시를 <>로 표기한다.
S	S9999 9999S	+1234 1234+	양수/음수 부호를 해당 위치에 표기한다.

●데이터 형식 (DATE)

형식요소	예제	결과	설명
'..;/" text'	YYYY-MM-DD	2009-10-07	결과 값의 해당하는 위치에 그대로 출력된다.
D	D	4	일주일 중 몇 번째 날(1~7)
DAY	DAY	수요일	요일 이름을 표기한다.
DD	DD	07	일자 표기(1~31)
DDD	DDD	280	일년 중 몇 번째 날(1~366)
DY	DY	수	축약한 요일 이름 표기
MM	MM	10	달을 표기
MON	MON	10월	축약한 달 이름 표기
MONTH	MONTH	10월	달 이름 표기
Q	Q	4	일년 중 몇 번째 분기(1~4)
YYYY	YYYY	2009	연도 4자리수 표기
YEAR	YEAR	TWO THOUSAND NINE	연도를 말로 풀어서 표기

●데이터 형식 (TIME)

형식요소	예제	결과	설명
FF[1..9]	'HH:MI:SS,FF3'	11:38:43.123	소수점 이하 자리의 초를 표시. FF 뒤에 명시한 숫자의 개수 만큼 소수점 이하 자릿수가 출력된다.
FM	FM YYYY /MM/DD	2009/10/7	앞뒤 공백을 제거하고 출력하도록 하는 조절자
HH HH12	HH:MI:SS HH12:MI:SS	11:40:22	시간 표기(1~12)
HH24	HH24:MI:SS	23:40:38	시간표기(0~23)
MI	HH:MI:SS	11:40:22	시간 중 분을 표기(0~59)
SS	HH:MI:SS	11:40:22	시간 중 초를 표기(0~59)
SSSSS	SSSSS	85192	자정부터 현재 몇 초(0~86399)

SQL문은 대소문자를 구별하지는 않지만 프로젝트의 규율에 맞추도록 합니다.

SQL문은 하나 이상의 줄에 입력할 수 있지만, 키워드는 여러 줄에 나누어 입력할 수 없습니다.

SQL 구문은 가로 최대 길이가 80 컬럼 이하가 되도록 작성하여 SQL 튜닝시 또는 유지 보수를 할 때 쉽게 이해할 수 있도록 한다.

절은 일반적으로 읽기 쉽고 편집하기 쉽도록 다른 줄에 씁니다.

들여쓰기를 사용하면 좀더 읽기 쉬운 SQL문을 작성할 수 있습니다.

일반적으로 키워드는 대문자로 입력하고 테이블 이름, 열 등 다른 단어는 모두 소문자로 입력합니다.

TAB과 들여쓰기를 사용하여 좀 더 읽기 쉬운 SQL로 작성하도록 합니다.

Chapter 8장 함수 사용.

- 주어진 인수를 처리하여 결과값을 반환한다.
- SELECT문을 간결하게 만들어 Data 조작을 쉽고 간결하게 만든다.
- 종류

함수 (function)

단일행 함수
(Single-Row Functions)

집합 함수
(Aggregate Functions)

분석 함수
(Analytical Functions)

- 숫자 데이터 타입의 값을 조작하여 변환된 숫자 값을 반환한다.

SQL 예시	결과
<code>SELECT MOD(10,3) "Mod" FROM dual;</code>	1
<code>SELECT ROUND(18.172,2) "Round" FROM dual;</code>	18.17
<code>SELECT ABS(-10) "Abs" FROM dual;</code>	10
<code>SELECT CEIL(10.1) "Ceil" FROM dual;</code>	11
<code>SELECT TRUNC(50.627,2) "Trunc" FROM dual;</code>	50.62

- 문자 데이터 타입의 값을 조작하여 변환된 문자 값을 반환한다.

SQL 예시	결과
SELECT CONCAT('Tmax','Soft') "Concat" FROM dual;	TmaxSoft
SELECT SUBSTR('TmaxSoft',2,5) "Substr" FROM dual;	maxSo
SELECT LENGTH('TmaxSoft') "Length" FROM dual;	8
SELECT INSTR('TmaxSoft','a') "Instr" FROM dual;	3
SELECT LOWER('TmaxSoft') "Lower" FROM dual;	tmaxsoft
SELECT UPPER('TmaxSoft') "Upper" FROM dual;	TMAXSOFT
SELECT LPAD('TmaxSoft',10,'*') "Lpad" FROM dual;	**TmaxSoft
SELECT RPAD('TmaxSoft',10,'*') "Rpad" FROM dual;	TmaxSoft**

- DATE로 지정된 데이터 타입의 값에 적용한다.

SQL 예시

```
SELECT  
ADD_MONTHS(TO_DATE('2009/10/07','YYYY/MM/DD'),1)  
"Add_months" FROM dual;
```

```
SELECT  
ROUND(TO_DATE('2009/10/07','YYYY/MM/DD'),'YEAR')  
"Round" FROM dual;
```

```
SELECT TRUNC(TO_DATE('2009/10/07','YYYY/MM/DD'),'YEAR')  
"Round" FROM dual;
```

```
SELECT SYSDATE FROM dual;
```

```
SELECT LAST_DAY(SYSDATE) FROM dual;
```

```
SELECT MONTHS_BETWEEN(LAST_DAY(SYSDATE),SYSDATE)  
FROM dual;
```

결과

2009/11/07

2010/01/01

2009/01/01

현재시스템날짜

해당월의
마지막 일

.77419355485

- 데이터 타입을 변환시켜 표현한다.

SQL 예시

결과

`SELECT TO_CHAR(sysdate) FROM dual;` 2009/10/07

`SELECT TO_DATE('07/10/2009','DD/MM/YYYY') FROM dual;` 2009/10/07

`SELECT TO_NUMBER('10000') FROM dual;` 10000

DECODE 함수

●SQL : 문법

DECODE(expr, search, result, default)

●Query 및 결과 예제

```
SELECT DISTINCT DECODE(PROD_PID,101,'AnyLink'  
                      ,201,'ProFrame'  
                      ,301,'JEUS'  
                      ,401,'WebtoB'  
                      ,501,'TMAX'  
                      ,601,'TIBERO'  
                      ,'기타') "제품구분"  
  
FROM PRODUCT;
```

Grid Result	
1	ProFrame
2	JEUS
Grid Result	Text Output

NVL 함수

●SQL : 문법

NVL(expr1, expr2)

●Query 및 결과 예제

```
SELECT prod_pid, prod_name, NVL(prod_pid,999) "nvl result"  
FROM product;
```

	PROD_PID	PROD_NAME	nvl result
1		AnyLink	999
2	101	AnyLink2,0	101
3	101	AnyLink3,0	101
4		ProFrame	999
5	201	ProFrame3,0	201
6	201	ProFrame4,0	201

NVL2 함수

●SQL : 문법

NVL2(expr1, expr2, expr3)

●Query 및 결과 예제

```
SELECT prod_pid, prod_name, NVL2(prod_pid,prod_id,999) "nvl2 result"  
FROM product;
```

Grid Result			
	PROD_PID	PROD_NAME	nvl2 result
1		AnyLink	999
2	101	AnyLink2,0	102
3	101	AnyLink3,0	103
4		ProFrame	999
5	201	ProFrame3,0	202
6	201	ProFrame4,0	203

Grid Result Text Output Explain Plan

Ln 30, Col 27 OVR 0,01 sec. 18 rows
AutoCommit is ON CAP NUM OVR

COALESCE 함수

●SQL : 문법

COALESCE(expr1, expr2, expr3,...)

●Query 및 결과 예제

```
SELECT comm, COALESCE(comm,1) "COALESCE" FROM employee;
```

	COMM	COALESCE
1	1000	1000
2	1000	1000
3	500	500
4	100	100
5	400	400
6	2000	2000

Grid Result Text Output Explain Plan

Ln 42, Col 19 OVF 0,03 sec, 16 rows

AutoCommit is ON CAP NUM OVR

AVG 함수

●SQL : 문법

AVG(expr)

●Query 및 결과 예제

```
SELECT AVG(salary) "Avg"  
FROM employee;
```

Grid Result	
	Avg
1	3562,5

Grid Result	Text Output	Explain Plan
Ln 34, Col 17 OVF 0,01 sec. 1 rows AutoCommit is ON CAP NUM OVR		

RANK 함수

●SQL : 문법

RANK(expr) WITHIN GROUP(ORDER BY expr)

●Query 및 결과 예제

```
SELECT -- 급여가 3000 인 사람의 상위 급여 순위를 나타낸다.  
       RANK(3000) WITHIN GROUP (ORDER BY salary DESC) "rank"  
FROM   employee;
```

Grid Result	
	rank
1	5

Grid Result Text Output Explain Plan

Ln 46, Col 1 OVF: 0,05 sec., 1 rows

AutoCommit is ON CAP NUM OVR

RANK 함수 (계속)

●SQL : 문법

RANK(expr) OVER(query_partition_clause order_by_clause)

●Query 및 결과 예제

```
SELECT emp_id, salary, emp_name, RANK() OVER(ORDER BY salary) "rank"  
FROM employee;
```

Grid Result

	EMP_ID	SALARY	EMP_NAME	rank
4	1998234	2900	BILL	4
5	2003122	3000	PAUL	5
6	2004098	3000	DAVID	5
7	2001267	3400	WAYNE	7
8	2007112	3400	FERNANDO	7
9	1999235	3500	THEDDORF	9

Grid Result Text Output Explain Plan

Ln 54, Col 1 OVF 0,03 sec. 16 rows

AutoCommit is ON CAP NUM OVR

SUM, MIN, MAX, COUNT, STDDEV 함수

●SQL : 문법

SUM(expr), MIN(expr), MAX(expr), COUNT(expr), STDDEV(expr)

●Query 및 결과 예제

```
SELECT -- 연도별 입사자의 수와 최대, 최소 급여를 조회한다.  
       TO_CHAR(hiredate,'YYYY'), COUNT(emp_id), MAX(salary), MIN(salary)  
FROM   employee  
GROUP BY TO_CHAR(hiredate,'YYYY');
```

Grid Result				
	TO_CHAR(HIREDATE, 'YYYY')	COUNT(EMP_ID)	MAX(SALARY)	MIN(SALARY)
1	2002	1	4400	4400
2	2005	2	5000	4000
3	2003	2	3800	3000
4	2004	2	4000	3000
5	1999	1	3500	3500
6	2001	3	3400	1800

Grid Result Text Output Explain Plan

Ln 63, Col 13 OVF 0,03 sec. 9 rows

AutoCommit is ON CAP NUM OVR

FIRST_VALUE 함수

●SQL : 문법

FIRST_VALUE(expr) OVER (analytic_clause)

●Query 및 결과 예제

```
SELECT -- 전체 사원의 급여와 함께 각 부서의 최고 급여를 조회한다.  
      emp_id, salary  
      , FIRST_VALUE(salary) OVER (PARTITION BY dept_id ORDER BY salary DESC)  
      "FirstValue"  
  FROM employee;
```

Grid Result

	EMP_ID	SALARY	FirstValue
1	2006246	6000	6000
2	2005287	5000	6000
3	2004001	4000	6000
4	2007112	3400	6000
5	2003122	3000	6000
6	2004098	3000	6000

Grid Result Text Output Explain Plan

Ln 4, Col 19 OVF 0.03 sec. 16 rows

AutoCommit is ON CAP NUM OVR

COUNT 함수

●SQL : 문법

COUNT(expr) OVER (analytic_clause)

●Query 및 결과 예제

```
SELECT /* 부서번호가 10인 부서 직원에 대해 각 직원의 급여보다 같거나 적게 받는
         사람에 대한 누적 합을 조회한다. */
        emp_id, salary
       ,COUNT(*) OVER (ORDER BY salary) "count"
  FROM employee
 WHERE dept_id = 10;
```

Grid Result

	EMP_ID	SALARY	count
1	2001043	1800	1
2	2001397	2000	2
3	2006121	2800	3
4	2004098	3000	5
5	2003122	3000	5
6	2007112	3400	6

Grid Result Text Output Explain Plan

Ln 18, Col 22 OVF 0,03 sec. 9 rows

AutoCommit is ON CAP NUM OVR

SUM 함수

●SQL : 문법

SUM(expr) OVER (analytic_clause)

●Query 및 결과 예제

```
SELECT -- 전직원에 대한 누적급여 합계를 조회한다.  
      emp_id, emp_name, salary  
     ,SUM(salary) OVER (ORDER BY emp_id) "sum"  
FROM   employee;
```

	EMP_ID	EMP_NAME	SALARY	sum
1	1998232	KAREN	4000	4000
2	1998234	BILL	2900	6900
3	1999235	THEODORE	3500	10400
4	2001043	STEVE	1800	12200
5	2001267	WAYNE	3400	15600
6	2001397	JANIE	2000	17600

Ln 18, Col 2 OVF 0,05 sec. 16 rows
AutoCommit is ON CAP NUM OVR

ROW_NUMBER 함수

●SQL : 문법

ROW_NUMBER() OVER ([query_partition_clause] order_by_clause)

●Query 및 결과 예제

```
SELECT -- 부서별 급여 순위 계산하여 조회한다.  
       e.emp_id, e.emp_name, d.dept_name, e.salary  
     ,ROW_NUMBER() OVER (PARTITION BY d.dept_id ORDER BY salary) "row_num"  
FROM   employee e, department d  
WHERE  e.dept_id = d.dept_id;
```

Grid Result

	EMP_ID	EMP_NAME	DEPT_NAME	SALARY	row_num
1	2001043	STEVE	engineer	1800	1
2	2001397	JANE	engineer	2000	2
3	2006121	LINCOLN	engineer	2800	3
4	2003122	PAUL	engineer	3000	4
5	2004098	DAVID	engineer	3000	5
6	2007112	FERNANDO	engineer	3400	6

Grid Result | Text Output | Explain Plan

Ln 59, Col 30 OVF 0,11 sec. 16 rows

AutoCommit is ON CAP NUM OVR

RANK 함수

●SQL : 문법

RANK() OVER (order_by_clause)

●Query 및 결과 예제

```
SELECT -- 부서별 급여 순위 계산하여 조회한다.  
       e.emp_id, e.emp_name, d.dept_name, e.salary  
     ,RANK() OVER (PARTITION BY d.dept_id ORDER BY salary) "rank"  
FROM   employee e, department d  
WHERE  e.dept_id = d.dept_id;
```

Grid Result

	EMP_ID	EMP_NAME	DEPT_NAME	SALARY	rank
2	2001397	JANE	engineer	2000	2
3	2006121	LINCOLN	engineer	2800	3
4	2003122	PAUL	engineer	3000	4
5	2004098	DAVID	engineer	3000	4
6	2007112	FERNANDO	engineer	3400	6
7	2004001	MICHAEL	engineer	4000	7

Grid Result Text Output Explain Plan

Ln 41, Col 31 OVF 0,36 sec. 16 rows

AutoCommit is ON CAP NUM OVR

DENSE_RANK 함수

●SQL : 문법

DENSE_RANK() OVER ([query_partition_clause] order_by_clause)

●Query 및 결과 예제

```
SELECT -- 부서별 급여 순위 계산하여 조회한다.  
       e.emp_id, e.emp_name, d.dept_name, e.salary  
     ,DENSE_RANK() OVER (PARTITION BY d.dept_id ORDER BY salary) "rank"  
FROM   employee e, department d  
WHERE  e.dept_id = d.dept_id;
```

Grid Result

	EMP_ID	EMP_NAME	DEPT_NAME	SALARY	rank
2	2001397	JANE	engineer	2000	2
3	2006121	LINCOLN	engineer	2800	3
4	2003122	PAUL	engineer	3000	4
5	2004098	DAVID	engineer	3000	4
6	2007112	FERNANDO	engineer	3400	5
7	2004001	MICHAEL	engineer	4000	6

Grid Result Text Output Explain Plan

Ln 58, Col 33 OVF 0,03 sec. 16 rows

AutoCommit is ON CAP NUM OVR

Chapter 9장 데이터 조회 고급 활용

조인(JOIN)이란?

- 둘 이상의 테이블을 연결하여 데이터를 검색하는 방법
- 둘 이상의 행들의 공통된 값을 사용하여 조인한다.

조인유형 (ANSI)

- American National Standard Institute에서 정의한 JOIN 표준안

- CROSS JOIN
- NATURAL JOIN
- JOIN ~ USING
- JOIN ~ ON
- OUTER JOIN

SELECT FROM을 이용한 조인

CHAPTER 9장. 데이터 조회 고급 활용

●SQL : 문법

```
SELECT 컬럼명1, 컬럼명2  
FROM   테이블명1, 테이블명2  
WHERE  테이블명1.컬럼이름 = 테이블명2.컬럼이름
```

●Query 및 결과 예제

```
SELECT e.emp_id, e.emp_name, d.dept_name  
FROM   employee e, department d  
WHERE  e.dept_id = d.dept_id;
```

Grid Result			
	EMP_ID	EMP_NAME	DEPT_NAME
7	2007112	FERNANDO	engineer
8	2006246	WARDEN	engineer
9	2001397	JANE	engineer
10	1999235	THEODORE	sales
11	1998232	KAREN	sales
12	1998234	RIKI	sales

Grid Result Text Output Explain Plan

Ln 81, Col 1 OVF 0,08 sec. 16 rows

AutoCommit is ON CAP NUM OVR

●SQL : 문법

```
SELECT 컬럼명1, 컬럼명2  
FROM 테이블명1 JOIN 테이블명2  
ON 테이블명1.컬럼이름 = 테이블명2.컬럼이름
```

●Query 및 결과 예제

```
SELECT e.emp_id, e.emp_name, d.dept_name  
FROM employee e JOIN department d  
ON e.dept_id = d.dept_id;
```

Grid Result			
	EMP_ID	EMP_NAME	DEPT_NAME
7	2007112	FERNANDO	engineer
8	2006246	WARDEN	engineer
9	2001397	JANE	engineer
10	1999235	THEODORE	sales
11	1998232	KAREN	sales
12	1998234	RIKI	sales

Grid Result Text Output Explain Plan

Ln 81, Col 1 OVF 0,08 sec. 16 rows

AutoCommit is ON CAP NUM OVR

●SQL : 문법

```
SELECT 컬럼명1, 컬럼명2, ...
FROM   테이블명1 별칭1, 테이블명2 별칭2, 테이블명3 별칭3
WHERE  별칭1.컬럼명1 = 별칭2.컬럼명1
AND    별칭1.컬럼명2 = 별칭3.컬럼명2
```

●Query 및 결과 예제

```
SELECT e.emp_name, c.cust_name, o.ord_amount
FROM   employee e, customer c, orders o
WHERE  e.emp_id = o.emp_id
AND    o.cust_id = c.cust_id;
```

Grid Result			
	EMP_NAME	CUST_NAME	ORD_AMOUNT
1	PAUL	chance	100
2	STEVE	shbank	100
3	WARDEN	kmbank	200
4	LINCOLN	teach	250

Grid Result | Text Output | Explain Plan

Ln 104, Col 20 | OVR 0,11 sec. | 18 rows

AutoCommit is ON | CAP | NUM | OVR

CROSS JOIN

●Query 및 결과 예제

```
SELECT emp_name FROM employee CROSS JOIN department;
```

Grid Result	
	EMP_NAME
1	STEVE
2	STEVE
3	STEVE
4	STEVE
5	PAUL
6	PAUL
7	PAUL
8	PAUL
9	MICHAEL
10	MICHAEL
11	MICHAEL
12	MICHAEL
13	DAVID

Grid Result Text Output Explain Plan

Ln 72, Col 1 OVF 0,06 sec, 64 rows

AutoCommit is ON CAP NUM OVR

NATURAL JOIN

●Query 및 결과 예제

```
SELECT emp_id, emp_name, dept_name  
FROM employee NATURAL JOIN department;
```

Grid Result

	EMP_ID	EMP_NAME	DEPT_NAME
1	2001043	STEVE	engineer
2	2003122	PAUL	engineer
3	2004001	MICHAEL	engineer
4	2004098	DAVID	engineer
5	2005287	JOHN	engineer
6	2006121	LINCOLN	engineer
7	2007112	FERNANDO	engineer
8	2006246	WARDEN	engineer
9	2001397	JANE	engineer
10	1999235	THEODORE	sales
11	1998232	KAREN	sales
12	1998234	BILL	sales
13	2003465	SANDRA	sales

Grid Result Text Output Explain Plan

Ln 112, Col 17 OVF 0.08 sec. 16 rows

AutoCommit is ON CAP NUM OVR

JOIN ~ USING

●Query 및 결과 예제

```
SELECT e.emp_id, e.emp_name, d.dept_name  
FROM employee e JOIN department d USING(dept_id);
```

Grid Result

	EMP_ID	EMP_NAME	DEPT_NAME
1	2001043	STEVE	engineer
2	2003122	PAUL	engineer
3	2004001	MICHAEL	engineer
4	2004098	DAVID	engineer
5	2005287	JOHN	engineer
6	2006121	LINCOLN	engineer
7	2007112	FERNANDO	engineer
8	2006246	WARDEN	engineer
9	2001397	JANE	engineer
10	1999235	THEODORE	sales
11	1998232	KAREN	sales
12	1998234	BILL	sales
13	2003465	SANDRA	sales

Grid Result Text Output Explain Plan

Ln 112, Col 17 OVF 0.08 sec. 16 rows

AutoCommit is ON CAP NUM OVR

JOIN ~ ON

●Query 및 결과 예제

```
SELECT e.emp_id, e.emp_name, d.dept_name  
FROM employee e JOIN department d ON e.dept_id=d.dept_id
```

Grid Result

	EMP_ID	EMP_NAME	DEPT_NAME
1	2001043	STEVE	engineer
2	2003122	PAUL	engineer
3	2004001	MICHAEL	engineer
4	2004098	DAVID	engineer
5	2005287	JOHN	engineer
6	2006121	LINCOLN	engineer
7	2007112	FERNANDO	engineer
8	2006246	WARDEN	engineer
9	2001397	JANE	engineer
10	1999235	THEODORE	sales
11	1998232	KAREN	sales
12	1998234	BILL	sales
13	2003465	SANDRA	sales

Grid Result Text Output Explain Plan

Ln 112, Col 17 OVF 0.08 sec. 16 rows

AutoCommit is ON CAP NUM OVR

OUTER JOIN

●Query 및 결과 예제

NO	NAME
10	AAA
20	BBB

T1 테이블

NO	NAME
10	ATT
30	CCC

T2 테이블

The screenshot shows a 'Grid Result' window from a database interface. The results are as follows:

	NO	NAME	NO	NAME
1	10	AAA	10	ATT

Below the grid, there are tabs for 'Grid Result', 'Text Output', and 'Explain Plan'. Status information at the bottom includes: Ln 138, Col 7 | OVF 0,01 sec., AutoCommit is ON | CAP NUM OVR.

```
SELECT T1.NO, T1.NAME, T2.NO, T2.NAME  
FROM T1, T2  
WHERE T1.NO = T2.NO;
```

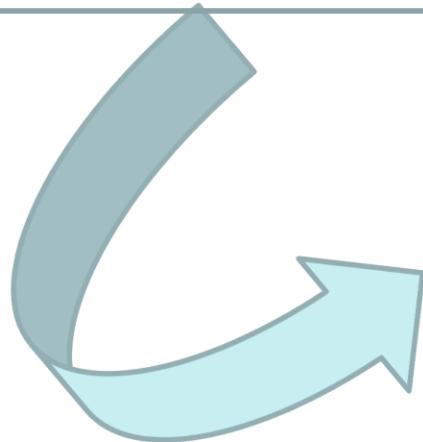


OUTER JOIN (계속)

●Query 및 결과 예제

```
SELECT T1.NO, T1.NAME, T2.NO, T2.NAME  
FROM T1, T2  
WHERE T1.NO(+) = T2.NO;
```

```
SELECT T1.NO, T1.NAME, T2.NO, T2.NAME  
FROM T1 RIGHT OUTER JOIN T2  
ON T1.NO = T2.NO;
```



Grid Result

	NO	NAME	NO	NAME
1	10	AAA	10	ATT
2			30	CCC

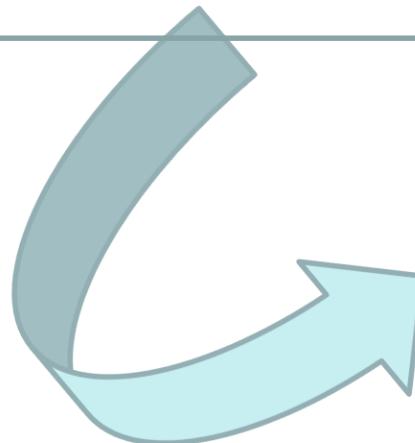
Grid Result Text Output Explain Plan
Ln 139, Col 1 OVF 0,05 sec.
AutoCommit is ON CAP NUM OVR

OUTER JOIN (계속)

●Query 및 결과 예제

```
SELECT T1.NO, T1.NAME, T2.NO, T2.NAME  
FROM T1, T2  
WHERE T1.NO = T2.NO(+);
```

```
SELECT T1.NO, T1.NAME, T2.NO, T2.NAME  
FROM T1 LEFT OUTER JOIN T2  
ON T1.NO = T2.NO;
```



Grid Result

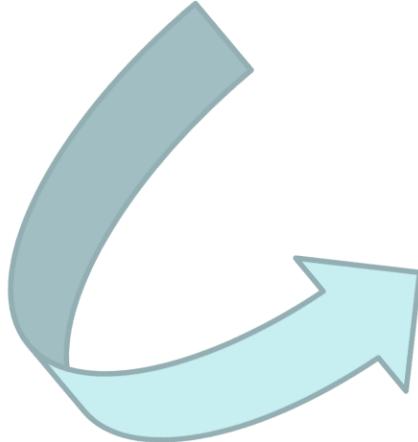
	NO	NAME	NO	NAME
1	10	AAA	10	ATT
2	20	BBB		

Grid Result | Text Output | Explain Plan
Ln 147, Col 1 OVF 0,06 sec.
AutoCommit is ON CAP NUM OVR

OUTER JOIN (계속)

●Query 및 결과 예제

```
SELECT T1.NO, T1.NAME, T2.NO, T2.NAME  
FROM   T1 FULL OUTER JOIN T2  
ON    T1.NO = T2.NO;
```



Grid Result

	NO	NAME	NO	NAME
1	10	AAA	10	ATT
2	20	BBB		
3			30	CCC

Grid R... Text O... Explai...
Ln 147, Col 20 OVF 0.05 sec.
AutoCommit is ON CAP NUM OVR

●Query 및 결과 예제

```
SELECT emp_id, emp_name, dept_id, salary
FROM employee
WHERE salary IN (
    SELECT MAX(salary)
    FROM employee
    GROUP BY dept_id
)
ORDER BY dept_id;
```

Grid Result

	EMP_ID	EMP_NAME	DEPT_ID	SALARY
1	2004001	MICHAEL	10	4000
2	2006246	WARDEN	10	6000
3	1998232	KAREN	20	4000
4	2005853	CHARLES	20	4000
5	2002658	JAMES	30	4400

Grid Result Text Output Explain Plan

Ln 150, Col 1 OVF 0,06 sec. 5 rows

AutoCommit is ON CAP NUM OVR

●Query 및 결과 예제

```
SELECT emp_name, salary
    ,CASE WHEN dept_id =
        (SELECT dept_id FROM employee WHERE emp_id = '2004098')
        THEN 'PART1'
        ELSE 'OTHER'
    END AS "location"
FROM employee;
```

Grid Result			
	EMP_NAME	SALARY	location
6	LINCOLN	2800	PART1
7	FERNANDO	3400	PART1
8	WARDEN	6000	PART1
9	JANE	2000	PART1
10	THEODORE	3500	OTHER
11	KAREN	4000	OTHER
12	BILL	2900	OTHER

Grid Result Text Output Explain Plan

Ln 300, Col 23 OVF 0,03 sec. 16 rows

AutoCommit is ON CAP NUM OVR

●Query 및 결과 예제

```
SELECT b.emp_id, b.emp_name, b.salary, b.dept_id
FROM  (
    SELECT emp_id
    FROM employee
    WHERE salary > (SELECT AVG(salary) FROM employee WHERE dept_id = 10)
)a
, employee b
WHERE a.emp_id = b.emp_id
AND b.dept_id != 20;
```

Grid Result

	EMP_ID	EMP_NAME	SALARY	DEPT_ID
1	2004001	MICHAEL	4000	10
2	2005287	JOHN	5000	10
3	2006246	WARDEN	6000	10
4	2002658	JAMES	4400	30

Grid Result | Text Output | Explain Plan

Ln 261, Col 9 OVF 0.03 sec. 4 rows

AutoCommit is ON CAP NUM OVR

●Query 및 결과 예제

```
SELECT a.emp_id, a.cnt
FROM  (
    SELECT emp_id, count(*) cnt
    FROM   ORDERS
    GROUP BY emp_id
    ORDER BY cnt DESC
) a
WHERE  rownum<=3;
```

	ROWNUM	EMP_ID	CNT
1	1	1999235	4
2	2	2004098	3
3	3	2006246	3

Grid Result | Text Output | Explain Plan
Ln 278, Col 11 OVF 0.03 sec. 3 rows
AutoCommit is ON CAP NUM OVR

●SQL : 문법

```
SELECT 컬럼명, 계산식, 함수 ...
FROM   테이블명
[WHERE 조건]
[GROUP BY ROLLUP 그룹화할 조건]
```

●Query 및 결과 예제

```
SELECT e.emp_name
      , e.dept_id
      , ROUND(AVG(e.salary),2) AS
        "Avg_salary"
  FROM employee e, department d
 WHERE e.dept_id = d.dept_id
 GROUP BY ROLLUP (e.dept_id,
 e.emp_name);
```

	EMP_NAME	DEPT_ID	Avg_salary
1	DAVID	10	3000
2	FERNANDO	10	3400
3	JANE	10	2000
4	JOHN	10	5000
5	LINCOLN	10	2800
6	MICHAEL	10	4000
7	PAUL	10	3000
8	STEVE	10	1800
9	WARDEN	10	6000
10		10	3444.44
11	BILL	20	2900
12	CHARLES	20	4000
13	KAREN	20	4000
14	CANDRA	20	3800

●SQL : 문법

```
SELECT 컬럼명, 계산식, 함수 ...
FROM   테이블명
[WHERE 조건]
[GROUP BY CUBE 그룹화할 조건]
```

●Query 및 결과 예제

```
SELECT cust_id, emp_id,
       SUM(ord_amount)
  FROM orders
 GROUP BY CUBE (cust_id, emp_id);
```

Grid Result			
	CUST_ID	EMP_ID	SUM(ORD_AMOUNT)
27	s004	2004001	100
28	s004		100
29		1999235	500
30		2001043	100
31		2001267	100
32		2002658	130
33		2003122	300
34		2003465	130
35		2004001	100
36		2004098	320
37		2006121	250
38		2006246	420
39			2350

●비교정리

ROLLUP(a,b,c)	GROUPING SETS((a,b,c), (a,b), (a), ())
CUBE(a,b,c)	GROUPING SETS((a,b,c), (a,b), (a,c), (b,c), (a), (b), (c), ())

●Query 및 결과 예제

```
SELECT cust_id, emp_id,  
SUM(ord_amount)  
FROM orders  
GROUP BY GROUPING SETS ((cust_id,  
emp_id),(cust_id),(emp_id),());
```

Grid Result

	CUST_ID	EMP_ID	SUM(ORD_AMOUNT)
27	s004	2004001	100
28	s004		100
29		1999235	500
30		2001043	100
31		2001267	100
32		2002658	130
33		2003122	300
34		2003465	130
35		2004001	100
36		2004098	320
37		2006121	250
38		2006246	420
39			2350

Grid Result Text Output Explain Plan

Ln 195, Col 1 OVF 0,06 sec. 39 rows

AutoCommit is ON CAP NUM OVR

● 비교정리

```
SELECT [LEVEL], 컬럼명  
FROM 테이블명  
[WHERE 조건]
```

[**START WITH** 계층의 시작점이 될 행을 구별하는 논리식 표현]
[**CONNECT BY** 계층을 구성할 때 사용될 논리식 표현]

●Query 및 결과 예제

```
SELECT LPAD(prod_id,level*3,'.') prod_id, prod_name  
FROM product  
START WITH PROD_ID = 999  
CONNECT BY PRIOR PROD_ID = PROD_PID;
```

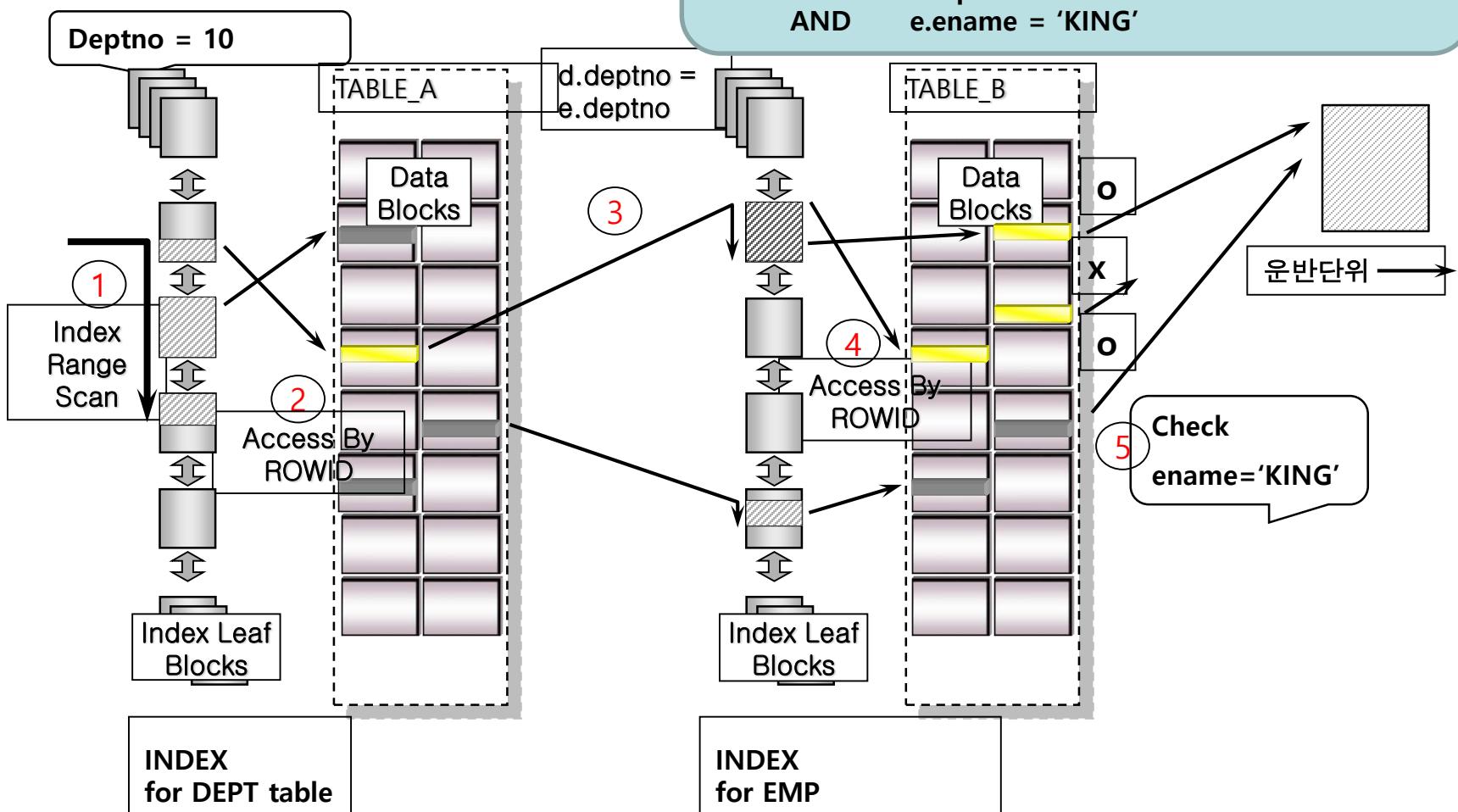
Grid Result		
	PROD_ID	PROD_NAME
1	999	TMAX PRODUCTS
2	...901	TMAX WINDOW
3902	TMAX WINDOW 11
4	...701	PRO IFRS
5	...801	PRO ERP

Grid Result Text Output Explain Plan

Ln 254, Col 8 OVF 0.03 sec. 5 rows

AutoCommit is ON CAP NUM OVR

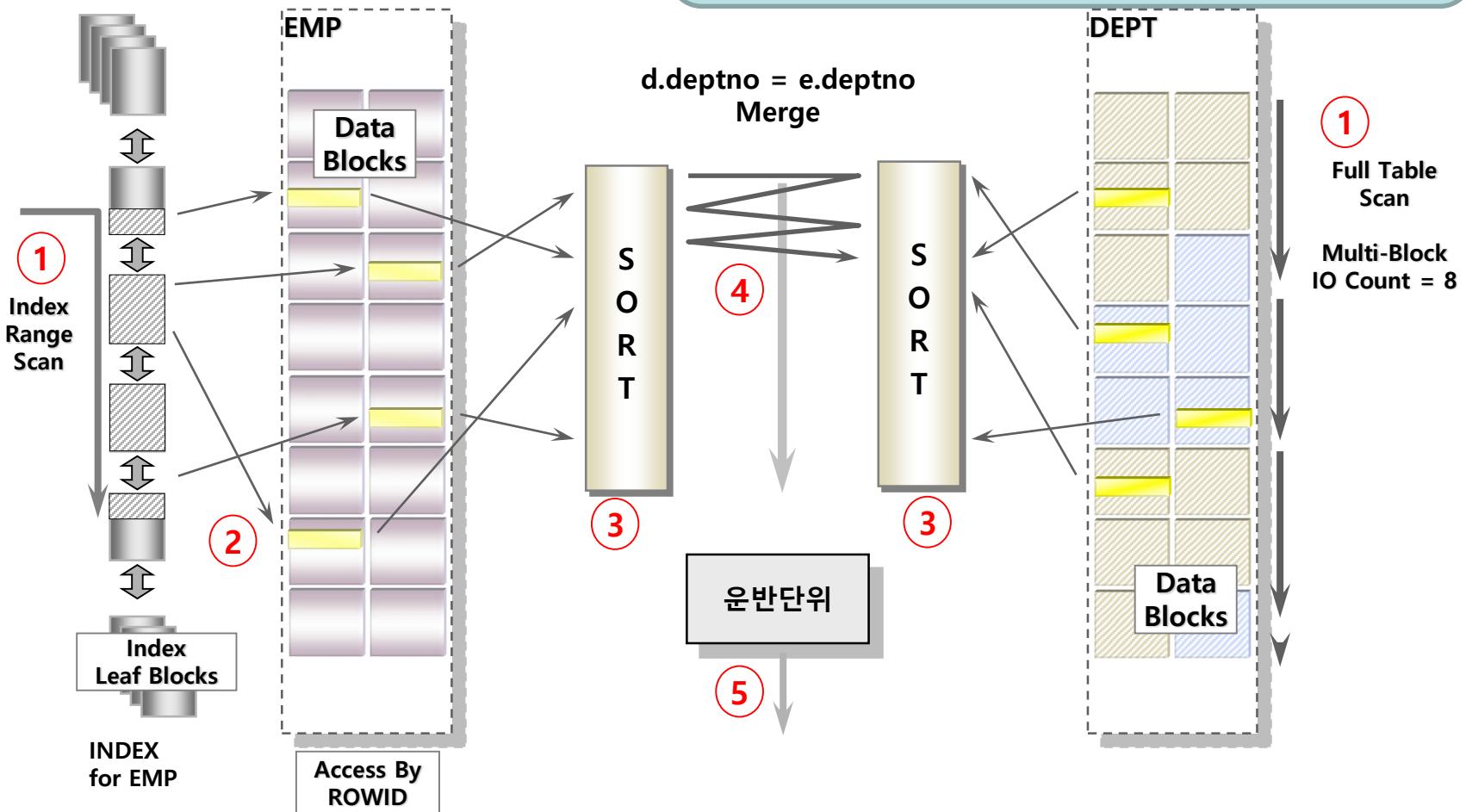
● NESTED LOOP JOIN(NLJ)



조인방식 (2/3)

● MERGE JOIN

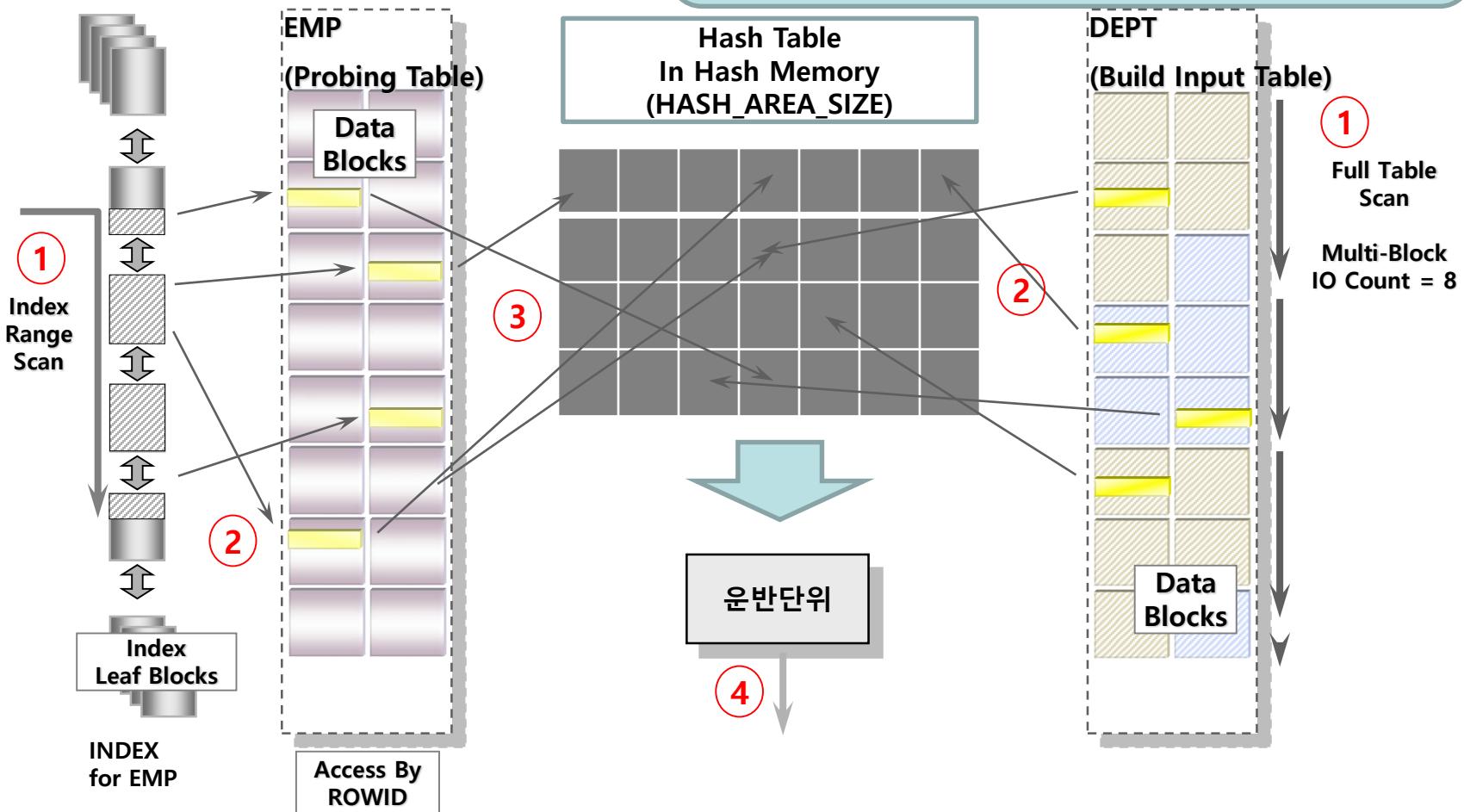
ename = 'KING'



조인방식 (3/3)

● HASH JOIN

ename = 'KING'



AUTOTRACE

●SQL : 문법

```
SET AUTOT[RACE] {OFF|ON|TRACE[ONLY]} [EXPLAIN] [STATISTICS]
```

●Query 및 결과 예제

```
SQL> SET AUTOTRACE ON;  
SQL> SELECT SYSDATE FROM DUAL;
```

SYSDATE

```
-----  
2009/10/13 17:37:23
```

1 row selected.

SQL ID: 57

Plan Hash Value: 2193767113

Explain Plan

```
1 DPV: _VT_DUAL (1,1)
```

VALUE EVENT_NAME

```
-----  
0 db block gets  
0 consistent gets  
0 physical reads  
0 redo size  
0 sorts (disk)  
0 sorts (memory)  
1 rows processed
```

Chapter 10장

DML 활용

- Data Manipulation Language
- 테이블의 데이터를 추가, 삭제, 변경 작업을 수행하는 명령어이다.

●SQL : 문법

- 기본 INSERT 문

INSERT INTO 테이블명(컬럼명1, 컬럼명2, ...) **VALUES** (값1, 값2, ...)

- SELECT문을 이용한 INSERT 문

INSERT INTO 테이블명(컬럼명1, 컬럼명2, ...) **SELECT** 구문

- Multi Table INSERT 문

INSERT [ALL ⚡ FIRST]

 WHEN 조건1 THEN

 INTO 테이블명1

 WHEN 조건2 THEN

 INTO 테이블명2

 ELSE

INTO 테이블명0

 SELECT 구문;

●SQL : 문법

- 기본 UPDATE 문

UPDATE 테이블명 **SET** 컬럼명 = 값1, 컬럼명 = 값2 ... [WHERE 조건]

- SUBQUERY를 이용한 UPDATE 문

UPDATE 테이블명

SET 컬럼명 = (SELECT 컬럼명 FROM 테이블명 [WHERE 조건])
[WHERE 조건]

- 테이블에 입력되어 있는 데이터를 삭제할 때 사용한다.
- SQL : 문법

DELETE FROM 테이블명 [WHERE 조건]

- UPDATE와 INSERT를 결합한 문장이다.

- SQL : 문법

MERGE INTO 테이블명

USING {table | view | subquery} **ON** (조건)

WHEN MATCHED THEN

 UPDATE SET 컬럼명1 = 값1 [, 컬럼명2 = 값2 ...]

WHEN NOT MATCHED THEN

 INSERT (컬럼리스트) VALUES (값 ...)

- Query 예제

```
MERGE INTO emp_history eh
```

```
USING employee e
```

```
ON (e.emp_id = eh.emp_id)
```

```
WHEN MATCHED THEN
```

```
    UPDATE SET eh.salary = e.salary
```

```
WHEN NOT MATCHED THEN
```

```
    INSERT VALUES (e.emp_id, sysdate, e.salary);
```

● Type

- 로우(row) 및 문장(statement)

타입	설명
로우	테이블에 INSERT, UPDATE, DELETE가 발생하는 로우마다 트리거의 내용이 실행되는 타입이다. 이타입의 트리거는 각 로우에 연산이 발생할 때마다 연산 직전 또는 직후에 트리거가 실행된다.
문장	로우의 갯수에 상관없이 문장 단위로 한 번만 실행되는 타입이다.

- BEFORE 및 AFTER

타입	설명
BEFORE	조건 문장이 실행되기 전에 트리거의 내용이 실행되는 타입이다.
AFTER	조건 문장이 실행된 후 트리거의 내용이 실행되는 타입이다.

●SQL : 문법

```
CREATE [OR REPLACE] TRIGGER 트리거_이름
{BEFORE | AFTER} {INSERT | UPDATE | DELETE} ON 테이블_이름
[FOR EACH ROW]
WHEN (조건_제약)
{[선언부]
BEGIN
...
END} |
CALL 함수_혹은_프로시저_이름;
```

●Trigger 생성 예제

```
CREATE OR REPLACE TRIGGER Log_overflow
AFTER UPDATE ON Deck_tbl
FOR EACH ROW
WHEN (new.count > 1000)
BEGIN
INSERT INTO Deck_log (Deck_id, Timestamp, New_count, Action)
VALUES (:new.Deck_no, SYSTIMESTAMP, :new.count, "overflow");
END;
```

Transaction 이란?

CHAPTER 10장. DML 활용

Transaction BEGIN

UPDATE 계좌1

SET 잔고 = 잔고 -10000

WHERE 계좌번호 = 1001;

UPDATE 계좌2

SET 잔고 = 잔고 +10000

WHERE 계좌번호 = 1002;

INSERT INTO 이체이력

VALUES(시퀀스,1001,1002,10000);

COMMIT/ROLLBACK;

Transaction END

●Query 및 결과 예제

Session 1

```
UPDATE employee  
SET salary = salary + salary * 0.1  
WHERE emp_id = 100
```

1

Session 2

```
UPDATE employee  
SET salary = salary + salary * 0.2  
WHERE emp_id = 100
```

2

DATA BASE

LOCK

Chapter 11장 tbPSM

● 개요

- SQL은 비절차적(Non-procedural) 언어로 데이터베이스의 모든 작업을 통제하며, 대체로 간단하고 적은 수의 명령어로 구성되어 있다. 따라서 데이터 구조와 알고리즘을 몰라도 자유롭게 SQL 명령을 실행할 수 있다는 장점을 갖고 있는 반면, 사용자가 원하는 데이터 처리 프로그램을 작성하려고 할 때는 불편함이 따른다.
- **tbPSM은 Tibero에서 제공하는 PSM 프로그램 언어 및 실행 시스템이다.** 순차적으로 원하는 결과를 얻어야 하는 프로그램을 SQL 문장만으로 작성할 수 없다는 제약 때문에 tbPSM이 필요하다

● tbPSM 란?

- Persistent Stored Modules
- 절차적 기능을 갖춘 확장된 SQL를 생성하는 언어이다.

● 절차형 언어(제3세대 언어)

- 변수와 타입
- 제어 구조
- 프로시저와 함수

● 변수와 상수

- 변수 - 프로그램을 작성할 때 값을 나타내는 문자나 문자들의 집합

변수이름 변수의 타입[제약조건] [디폴트 값 정의]

[할당 받는 대상] := [할당하는 대상]

-- 변수 선언

```
id NUMBER;  
productname VARCHAR2(20) := 'Tibero';
```

-- 변수 할당

```
grade := calculate_the_grade('Darwin');
```

- 상수 - 프로그램이 수행되는 동안은 변하지 않는 값을 나타내는 데이터

변수이름 CONSTANT 변수의 타입 := [디폴트 값 정의]

-- 상수 선언, 값 할당

```
PI CONSTANT NUMBER := 3.141592;
```

●제어 구조

tbPSM은 프로그램 실행 중에 발생되는 조건에 따라 특정 작업을 수행하거나 반복적인 작업을 수행하는 제어구조(Control Structure)를 제공한다.

- 선택적(selective) 제어 구조 - IF 문과 CASE 문
- 반복 제어 구조(iterative structure) - LOOP 문, WHILE 문, FOR 문
- IF문 예시

```
DECLARE
evaluation_score NUMBER;
BEGIN
SELECT comm into evaluation_score FROM employee WHERE emp_id = 2005287;

    IF evaluation_score > 80 THEN
        DBMS_OUTPUT.PUT_LINE('This person gets a salary bonus.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('This person does not get a salary
bonus.');
    END IF;
END;
```

● 서브프로그램과 패키지

- 서브프로그램

- 프로시저(procedure)와 함수(function)를 제공
- tbPSM의 프로그램 구조와 마찬가지로 선언부, 실행부, 예외 처리부로 구성된다.

- 서브프로그램 작성 예시

```
CREATE [OR REPLACE] PROCEDURE check_reg_number(name VARCHAR2) IS
    id NUMBER;
    invalid_id exception;
BEGIN
    SELECT emp_id INTO id FROM employee WHERE emp_name = name;
    IF id < 2000000 THEN
        raise invalid_id;
    END IF;
    DBMS_OUTPUT.PUT_LINE('Correct');
EXCEPTION
    WHEN invalid_id THEN
        DBMS_OUTPUT.PUT_LINE('Invalid registration number');
END;
```

● 서브프로그램과 패키지

- **패키지(Package)**는 서브프로그램과 변수, 상수의 집합이라 할 수 있다. 즉, 관련된 기능을 그룹으로 묶어 사용할 수 있어 프로그램을 관리하기가 쉬우며, 사용 시점에 일시에 메모리에 로드되므로 빠른 성능을 제공한다. 또한 모듈 간의 의존성 축소화를 위해서도 사용된다.

- 서브프로그램과 변수, 상수의 집합
- 선언부(specification)와 구현부(body)로 구성된다.

- Package 선언부 작성 예시

```
CREATE OR REPLACE PACKAGE book_manager IS
    book_cnt NUMBER;
    PROCEDURE add_new_book(v_author VARCHAR2, v_name
                           VARCHAR2,publish_year DATE);
    PROCEDURE remove_lost_book(v_author VARCHAR2, v_name VARCHAR2);
    FUNCTION search_book_position(v_author VARCHAR2,v_name VARCHAR2)
        RETURN NUMBER;
    FUNCTION get_total_book_cnt RETURN NUMBER;
END;
```

-Package 구현부 작성 예시

```
CREATE OR REPLACE PACKAGE BODY book_manager IS
    PROCEDURE add_new_book(v_author VARCHAR2, v_name VARCHAR2, publish_year DATE) IS
        BEGIN
            IF      substr(v_name, 1, 1) >= 'a' AND substr(v_name, 1, 1) < 'k'
            THEN   INSERT INTO books VALUES (1, v_author, v_name, publish_year);
            ELSE   INSERT INTO books VALUES (2, v_author, v_name, publish_year);
            END IF;
            COMMIT;
            book_cnt := book_cnt + 1;
        END;

    -----
    PROCEDURE remove_lost_book(v_author VARCHAR2, v_name VARCHAR2) IS
        BEGIN DELETE FROM books WHERE author = v_author AND name = v_name;
        COMMIT;
    END;

    -----
    FUNCTION search_book_position(v_author VARCHAR2, v_name VARCHAR2)
    RETURN NUMBER IS
        book_position NUMBER;
    BEGIN SELECT kind INTO book_position FROM books WHERE author = v_author AND name = v_name;
        RETURN book_position;
    EXCEPTION
        WHEN NO_DATA_FOUND
        THEN
            DBMS_OUTPUT.PUT_LINE('NOT EXIST...SORRY');
        RAISE;
    END;

    -----
    FUNCTION get_total_book_cnt RETURN NUMBER IS
    BEGIN RETURN book_cnt;
    END;

    BEGIN
        book_cnt := 0;
    END;
```

● 커서

- SQL 문장을 처리한 후 얻은 결과 집합에 대한 포인터이다.
- 커서 사용 예시

```
DECLARE
    CURSOR c1 IS SELECT * FROM employee;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO emp_id, emp_name, hiredate, salary, comm, dept_id;
        EXIT WHEN c1%NOTFOUND;
    END LOOP;
END;
```

● 에러처리

- tbPSM 프로그램은 실행 중에 에러가 발생할 수 있으며, 이러한 에러를 예외 상황이라 한다.
- 예외 상황(exception)을 처리하기 위한 루틴을 에러 처리 루틴(error handling routine)이라고 한다.
- 에러 처리 예시

```
DECLARE
    no_bonus exception;
    the_sales NUMBER := 12000;
BEGIN
    IF the_sales < 20000 THEN
        raise no_bonus;
    END IF;
    DBMS_OUTPUT.PUT_LINE('Your bonus is $' || the_sales * 0.1);
EXCEPTION
    WHEN no_bonus THEN
        DBMS_OUTPUT.PUT_LINE('Sorry, increase the sales');
END;
```

- 블록(block) 구조로 이루어져 있다.
- 하나의 블록은 크거나 작은 하나의 작업을 수행하는 모듈이다.
- 구성

- 선언부
- 실행 코드부
- 에러 처리부

● 예제

```
DECLARE
    -- 선언부
    name varchar(32);
BEGIN
    -- 실행 코드부
    SELECT ename INTO name FROM emp WHERE empno = 100;
EXCEPTION
    -- 에러 처리부
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('employee not exist');
END;
```

●식별자

- tbPSM 프로그램의 구성요소(변수, 상수, 레이블, 커서, 함수, 패키지 등)의 이름을 식별자라 한다.
- 특징
 - 알파벳 문자(대문자와 소문자), 한글, 숫자, \$, _, #를 사용하여 만든다.
 - 알파벳 문자 또는 한글로 시작해야 한다.
 - 대문자와 소문자를 구분하지 않는다.
 - 최대 길이는 255byte이다.

유효한 식별자의 예

x
employee_#
room_num_Local

유효하지 않은 식별자의 예

abc=xyz -- 허가되지 않은 기호(예: =)를 포함한 경우
_under_departs -- 알파벳 문자로 시작하지 않은 경우
Date Year -- 허가되지 않은 기호(예: 공백 문자)를 포함한 경우
000_name -- 알파벳 문자로 시작하지 않은 경우

●분리자

- 분리자는 식별자를 구분하기 위해 사용한다. 일부 분리자는 연산자의 역할도 수행한다.

기호	의미	기호	의미	기호	의미
+	덧셈	,	항목 구분	<=	크지않다(보다작거나같다)
-	뺄셈	.	컴포넌트구분	>=	작지않다(보다크거나같다)
*	곱셈	@	데이터베이스링크 표시	:=	대입
/	나눗셈	'	문자열분리자	..	범위
=	등호	"	인용된문자열분리자		문자열결합
<	부등호(보다작다)	:	바인드변수표시	<<	레이블(왼쪽)
>	부등호(보다크다)	**	지수	>>	레이블(오른쪽)
(괄호(왼쪽)	<>	같지않다	--	단일라인주석
)	괄호(오른쪽)	!=	같지않다	/*	다중라인(왼쪽)
:	문장의 끝 표시	~=	같지않다	*/	다중라인(오른쪽)
%	속성표시	^=	같지않다		

●상수

- 하나 또는 그 이상의 문자를 사용하여 어떤 값 자체를 표현하는 경우를 상수(literal)라 한다.
- 상수는 한 번 표시되면 변하지 않고, 숫자 상수, 문자 상수, 문자열 상수, 날짜 상수, 진리(boolean) 상수 등으로 분류된다.

●주석

- tbPSM이 인식하지 않는 문자열
- 단일 라인 주석 – ‘ -- ’ 사용한다.
- 다중 라인 주석 – ‘ /*...*/ ’ 사용한다.

```
DECLARE
    guest_ID      BINARY_INTEGER;    -- 고객을 식별하기 위한 ID
    guest_name    VARCHAR2(100);    -- 고객의 이름, 길이는 VARCHAR2 타입의 100자리

BEGIN
    INSERT INTO add_books (id, name) VALUES (guest_ID, guest_name);
    /* guest_ID와 guest_name으로 입력된 정보를
       add_books 테이블에 삽입한다. */
END;
```

● 스칼라 타입

그룹	서브타입
NUMERIC	NUMBER, DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NATURAL, NATURALN, NUMERIC, REAL, POSITIVE, POSITIVEN, SMALLINT, SIGNTYPE, PLS_INTEGER, BINARY_INTEGER, BINARY_FLOAT, BINARY_DOUBLE
CHARACTER / STRING	VARCHAR2, VARCHAR, CHAR, CHARACTER, LONG, STRING, RAW, ROWID, LONG RAW
DATETIME / INTERVAL	DATE, TIMESTAMP, INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND
BOOLEAN	BOOLEAN

● 복합 타입

그룹	서브타입
-	RECORD, VARRAY, TABLE

● 참조 타입

그룹	서브타입
-	REF CURSOR

● 대용량 객체형 타입

그룹	서브타입
-	CLOB, BLOB

● 기타 타입

그룹	서브타입
-	%TYPE, %ROWTYPE

● 명시적(explicit) 변환

- 명시적 변환은 시스템 변환 함수를 사용하여 변환하는 것을 말한다.
- TO_CHAR, TO_DATE, TO_CLOB, TO_TIMESTAMP, TO_NUMBER
- RAWTOHEX, HEXTORAW, CHARTOROWID, ROWIDTOCHAR

● 묵시적(implicit) 변환

- 묵시적 변환은 변수 사이의 대입 등에서 필요하다고 판단될 경우 자동으로 일어나는 변환을 말한다.
- 예시

```
SQL> CREATE TABLE emp (id NUMBER, current_credits NUMBER(3));
SQL> INSERT INTO emp VALUES (1004, 2);
...
SQL> DECLARE
cur_cred VARCHAR2(5);
BEGIN
SELECT current_credits INTO cur_cred
FROM emp WHERE id = 1004;
END;
```

● 변수 선언

- 변수 이름 입력 후 데이터 타입 정의
- 변수에 초기 값을 할당할 수 있으며, 상수로 초기 값을 선언할 수도 있다.

● 예시

```
radius REAL := 1.0;  
pi CONSTANT REAL := 3.141592654;  
area REAL NOT NULL := 0.0;  
SUBTYPE Single IS NUMBER(1, 0) NOT NULL;
```

● 변수의 참조 영역

- 변수의 참조 영역(scope)은 프로그램의 일부로서, 한 프로그램 내에서 해당 변수에 접근할 수 있는 영역을 의미한다.

```
DECLARE
    employee_num NUMBER;
BEGIN
    DECLARE
        employee_name VARCHAR2(100);
    BEGIN
        employee_num = '100';
        employee_name = 'Peter';
        ...
    END;
END;
```

```
DECLARE
    employee PLS_INTEGER;
BEGIN
    employee := 100;                                ... ① ...
    DECLARE
        employee VARCHAR2(255);                      ... ② ...
    BEGIN
        employee := 'John';                          ... ③ ...
    END;                                            ... ④ ...
    employee := employee + 10;                      ... ⑤ ...
END;                                              ... ⑥ ...
```

● 변수의 참조 영역

- 서브 블록 내에서는 외부 블록에서 선언된 employee 변수를 사용할 수 없다.
단 사용하고자 한다면 외부 블록에 레이블을 붙여서 사용해야 한다.

```
<<first_block>>
DECLARE
    employee PLS_INTEGER;
BEGIN
    employee := 100;
    DECLARE
        employee VARCHAR2(255);
    BEGIN
        employee := 'Susan';
        first_block.employee := 200;
    END;
    employee := employee + 10;
END;
```

● CASE 연산자

```
name := CASE order
WHEN 1 THEN 'Mercury'
WHEN 2 THEN 'Venus'
WHEN 3 THEN 'Earth'
...
ELSE 'No Such Planet'
END;
```

```
name := CASE
WHEN 1 <= order and order <= 2 THEN 'Inner Planet'
WHEN order = 3 THEN 'Earth'
WHEN 4 <= order and order <= 9 THEN 'Outer Planet'
ELSE 'No Such Planet'
END;
```

● 대입연산자

```
DECLARE
local_01 VARCHAR2(100);
BEGIN
local_01 := 'Seoul';
END;
```

- 조건 구조

- IF, CASE

- 반복 구조

- LOOP

- 단순 구조

- GOTO

● IF 문

형태	설명
IF-THEN	한 가지 경우만을 선택하여 사용하는 가장 단순한 형태
IF-THEN-ELSE	두 가지 중에서 하나를 선택하여 사용
IF-THEN-ELSEIF	다양한 경우에서 하나를 선택하여 사용

```
IF 조건식 THEN  
    실행문  
END IF;
```

```
If 조건식 THEN  
    실행문-1  
ELSE  
    실행문-2  
END IF;
```

```
If 조건식 THEN  
    실행문-1  
ELSIF 조건식 THEN  
    실행문-2  
...  
ELSIF 조건식 THEN  
    실행문-n  
END IF;
```

●CASE 문

CASE 표현식

```
WHEN 연산식 THEN 실행문-1  
WHEN 연산식 THEN 실행문-2  
...  
WHEN 연산식 THEN 실행문-n  
END CASE;
```

CASE

```
WHEN 조건식 THEN 실행문-1  
WHEN 조건식 THEN 실행문-2  
...  
WHEN 조건식 THEN 실행문-n  
END CASE;
```

CASE employee_grade

```
WHEN 'A' THEN pay_bonus_a(employee_num);  
WHEN 'B' THEN pay_bonus_b(employee_num);  
WHEN 'C' THEN pay_bonus_c(employee_num);  
ELSE check_grade(employee_num);  
END CASE;
```

DECLARE

```
value PLS_INTEGER := 0;  
result VARCHAR2(10);  
BEGIN  
CASE  
WHEN value = 0 THEN result := 'true';  
WHEN value != 0 THEN result := 'false';  
END CASE;  
DBMS_OUTPUT.PUT_LINE( 'Is the value Zero? ' || result );  
END;  
/
```

●LOOP 문

형태	설명
단순 LOOP	단순히 반복을 계속하는 LOOP문
FOR-LOOP	조건을 부여할 수 있는 LOOP문
WHILE-LOOP	조건을 부여할 수 있는 LOOP문

```
LOOP  
실행문  
END LOOP;
```

```
FOR loop_counter IN low_bound..high_bound  
LOOP  
실행문  
END LOOP;
```

```
WHILE 조건식 LOOP  
실행문  
END LOOP;
```

●LOOP 문 사용예

```
LOOP  
v_order := v_order + 1;  
IF v_order > 9 THEN  
EXIT;  
END IF;  
END LOOP;
```

```
LOOP  
v_order := v_order + 1;  
EXIT WHEN v_order > 9;  
END LOOP;
```

```
<<LOOP_OUT>>  
LOOP  
  <<LOOP_IN>>  
  LOOP  
    v_order := v_order + 1;  
    EXIT LOOP_OUT WHEN v_order > 9;      ... a ...  
    END LOOP <<LOOP_IN>>;  
  END LOOP <<LOOP_OUT>>;      ... b ...
```

●LOOP 문 사용예

```
FOR box_num IN REVERSE 1..100 LOOP  
box_weight := box_num * 10;  
END LOOP;
```

```
WHILE TRUE LOOP  
실행문 집합  
END LOOP;
```

```
DECLARE  
low_value BINARY_INTEGER := 10;  
high_value BINARY_INTEGER := 50;  
box_weight NUMBER;  
BEGIN  
FOR box_num IN low_value..high_value LOOP  
box_weight := box_num * 10;  
END LOOP;  
END;
```

●GOTO 문

- GOTO 문이 실행되면 GOTO 문에 명시된 레이블을 찾아서 해당 실행문이나 블록으로 제어를 이동한다.
- LABEL 문이 GOTO 문보다 먼저 올 수 있다.

```
GOTO lable;
```

●EXIT 문

```
EXIT [레이블];
```

```
BEGIN
    <<outer>>
    FOR out_index IN 1..100 LOOP
        <<inner>>
        FOR inner_index IN 1..1000 LOOP
            IF inner_index > out_index THEN
                EXIT outer;
            END IF;
        END LOOP inner;
    END LOOP outer;
END;
```

●NULL 문

- NULL 문은 프로그램상에 명시되어도 실제로는 아무런 일도 발생하지 않는다.
- 프로그램을 설계할 때 서브 프로그램의 내용을 일시적으로 NULL 문으로 대체 경우
- 제어 구조를 작성하면서 실행문을 NULL 문으로 대체하는 경우

```
DECLARE
    tmp_current_guest BINARY_INTEGER;
    tmp_max_guest BINARY_INTEGER;

    PROCEDURE raise_guest_max (current_guest BINARY_INTEGER) IS
    BEGIN
        NULL;
    END;
BEGIN
    SELECT current_guest, max_guest INTO tmp_current_guest, tmp_max_guest
    FROM guest WHERE local = 'SEOUL';

    IF tmp_current_guest > tmp_max_guest THEN
        raise_guest_max (tmp_current_guest);
    ELSE
        NULL;
    END IF;
END;
```

- tbPSM이 제공하는 스칼라 타입의 집합이다.
- 일반적인 프로그래밍 언어의 구조체에 해당한다.
- 종류
 - 컬렉션(collection) 타입
 - 테이블(nested table), 배열(varray)
 - 레코드(record)

● 컬렉션 타입

- 테이블

```
TYPE name IS TABLE OF type [NOT NULL];
```

```
DECLARE
    TYPE kind_tab IS TABLE OF VARCHAR2(10) NOT NULL;
    kinds kind_tab;
BEGIN
    kinds := kind_tab('math', 'physics', 'history', 'science');
END;
```

```
DECLARE
    TYPE class_tab IS TABLE OF VARCHAR2(10);
    classes class_tab := class_tab('math', 'physics', 'science');
BEGIN
    classes(3) := 'history';
    IF classes(1) IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('I hate this class');
    END IF;
    DBMS_OUTPUT.PUT_LINE(classes(3) || ' is my favorite class');
END;
/
history is my favorite class
PSM completed
```

● 컬렉션 타입

- 배열

```
TYPE name IS VARRAY(limit) OF type [NOT NULL];
```

```
DECLARE
  TYPE class_arr IS VARRAY(10) OF VARCHAR2(10) NOT NULL;
  classes class_arr;
BEGIN
  classes := class_arr('math', 'physics', 'history', 'science');
END;
```

```
DECLARE
  TYPE class_arr IS VARRAY(10) OF VARCHAR2(10);
  classes class_arr := class_arr('math', 'physics', 'art');
BEGIN
  classes(3) := 'history';
  IF classes(1) IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('I hate this class');
  END IF;
  DBMS_OUTPUT.PUT_LINE(classes(3) || ' is my favorite class');
END;
/
history is my favorite class
PSM completed
```

● 컬렉션 타입

- 컬렉션 함수와 프로시저

구분	이름	설명
컬렉션 함수	EXISTS COUNT LIMIT FIRST, LAST PRIOR, NEXT	컬렉션 함수는 항상 반환 값을 갖는다.
프로시저	EXTEND TRIM DELETE	프로시저는 반환 값을 갖지 않는다.

- 예외 상황

구분	설명
COLLECTION_IS_NULL	초기화되지 않은 컬렉션 변수에 접근하는 경우
NO_DATA_FOUND	DELETE 프로시저에 의해 제거된 구성요소에 접근하는 경우
SUBSCRIPT_BEYOND_COUNT	인덱스가 구성요소의 개수를 초과한 값으로 주어진 경우
SUBSCRIPT_OUTSIDE_LIMIT	인덱스가 허용 범위를 벗어난 경우
VALUE_ERROR	인덱스가 NULL이거나 숫자의 형태로 변환이 안 되는 경우

● 레코드

- 관련 있는 구성요소의 집합
- 일반적인 프로그래밍 언어의 구조체와 동일
- 레코드에 소속된 모든 구성요소의 타입이 서로 같을 필요가 없다.
- 필드가 다시 레코드를 포함할 수도 있다.

```
TYPE rec_name IS  
RECORD (field1 field1_type[, field2 field2_type...]);
```

```
DECLARE  
    music_info musics%ROWTYPE;  
BEGIN  
    SELECT * INTO music_info FROM musics WHERE kind = 'POP';  
    DBMS_OUTPUT.PUT_LINE(music_info.title);  
END;
```

- 서브프로그램(subprogram)은 다른 tbPSM 프로그램 내에서 호출할 수 있는 프로그램 블록이다.

● 장점

- 모듈성(modularity)
- 재사용성(reusability), 관리의 편의성(maintainability)
- 추상화(abstraction)

● 구분

- 프로시저(procedure)
- 함수(function)

● 프로시저

- 반환 값이 없다.
- 문법

```
[CREATE [OR REPLACE]] PROCEDURE 프로시저_이름 [(파라미터[, 파라미터])]  
[AUTHID {DEFINER | CURRENT_USER}] {AS | IS}  
[PRAGMA AUTONOMOUS_TRANSACTION;]  
[선언부]  
BEGIN  
[실행부]  
[예외 처리부]  
END;
```

● 프로시저

- 예제

```
CREATE OR REPLACE PROCEDURE NEW_EMP (ename VARCHAR2, deptno NUMBER)
IS
    salary NUMBER;
    deptno_not_found EXCEPTION;
BEGIN
    IF deptno = 5 THEN
        salary := 30000;
    ELSIF deptno = 6 THEN
        salary := 35000;
    ELSE
        RAISE deptno_not_found;
    END IF;

    INSERT INTO EMP (ENAME, SALARY, DEPTNO) VALUES (ename, salary, deptno);
EXCEPTION
    WHEN deptno_not_found THEN
        -- Report Unknown Deptno
        ROLLBACK; ... ④ ...
END NEW_EMP;
```

●함수

- 반환 값이 있다. 따라서 RETURN 문이 반드시 삽입되어야 한다.
- 문법

```
[CREATE [OR REPLACE]] FUNCTION 함수_이름 [(파라미터[, 파라미터])]  
RETURN 반환_타입 [AUTHID {DEFINER | CURRENT_USER}] {AS | IS}  
[PRAGMA AUTONOMOUS_TRANSACTION;]  
[선언부]  
BEGIN  
[실행부]  
RETURN 반환 값;  
[예외 처리부]  
END;
```

```
CREATE OR REPLACE FUNCTION NEW_EMP (ename VARCHAR, deptno INT)  
RETURN NUMBER IS  
...  
BEGIN  
...  
RETURN salary;  
EXCEPTION  
...  
END NEW_EMP;
```

- 패키지 (package)는 개념적으로 관련 있는 tbPSM의 변수나 타입, 서브프로그램을 그룹화하여 모아 놓은 객체이다.

● 구조

- 선언부

```
CREATE [OR REPLACE] PACKAGE 패키지_이름
[AUTHID {CURRENT_USER | DEFINER}] {IS | AS}
    [변수, 타입 선언...] -- 공개
    [커서 선언...]
    [함수 선언...]
    [프로시저 선언...]
END;
```

- 구현부

```
CREATE [OR REPLACE] PACKAGE BODY 패키지_이름
[AUTHID {CURRENT_USER | DEFINER}] {IS | AS}
    [변수, 타입 선언...] -- 비공개
    [커서 구현...]
    [함수 구현...]
    [프로시저 구현...]
[BEGIN
    초기화]
END;
```

● 패키지 예제

```
CREATE PACKAGE anniversary_manager IS
    month_counter PLS_INTEGER;
PROCEDURE compute_elapsed_days(start_day DATE);
END;
```

```
CREATE PACKAGE BODY anniversary_manager IS
    PROCEDURE compute_elapsed_days(start_day DATE) IS
BEGIN
    month_counter := months_between(sysdate(), start_day);
    DBMS_OUTPUT.PUT_LINE(month_counter * 30 || ' days...');
END;
BEGIN
    month_counter := 0;
    DBMS_OUTPUT.PUT_LINE('package is initialized');
END;
```

● 시스템 패키지

- Tibero에서 기본적으로 제공하는 패키지 라이브러리

시스템 패키지	설명
DBMS_LOB	BLOB, CLOB 타입의 대용량 데이터를 처리하기 위한 패키지이다.
DBMS_OBFUSCATION_TOOLKIT	DES, DES3 알고리즘을 이용한 데이터 암호화 및 복호화 패키지이다.
DBMS_OUTPUT	메시지 버퍼에 메시지를 저장하고 읽기 위한 패키지이다.
DBMS_STATS	데이터베이스 객체에 대한 통계 정보를 관리하기 위한 패키지이다.
DBMS_TRANSACTION	트랜잭션(transaction) 문장을 실행하고 트랜잭션을 관리하기 위한 패키지이다.
UTL_RAW	RAW 타입의 데이터를 처리하기 위한 패키지이다.
DBMS_JOB	JOB을 관리하기 위한 패키지이다.
DBMS_SQL	데이터베이스에 접근하는 Dynamic SQL을 사용하기 위한 패키지이다.
DBMS_ROWID	ROWID에 담긴 정보를 보거나 생성하기 위한 패키지이다.
DBMS_JAVA	데이터베이스에서 사용하는 Java 객체에 접근하기 위한 패키지이다.
UTL_FILE	운영체제에서 관리하는 파일에 접근하기 위한 함수와 프로시저를 제공하는 패키지이다.

● DCL

- COMMIT

```
COMMIT [task];
```

- ROLLBACK

```
ROLLBACK [task];
```

- SAVEPOINT

```
SAVEPOINT savepoint_name;  
ROLLBACK [task] TO SAVEPOINT savepoint_name;
```

- 자율 트랜잭션

```
PRAGMA AUTONOMOUS_TRANSACTION;
```

● Dynamic SQL

```
EXECUTE IMMEDIATE sql_stmt  
[INTO id_list USING id_list RETURNING INTO id_list];
```

```
DECLARE  
    sql_stmt VARCHAR2(2000);  
    emp_id NUMBER := 1;  
    emp_name VARCHAR2(10) := 'John';  
BEGIN  
    sql_stmt := 'INSERT INTO emp VALUES (:1, :2)';  
    EXECUTE IMMEDIATE sql_stmt USING emp_id, emp_name;  
    sql_stmt := 'SELECT name FROM emp WHERE id = :id';  
    EXECUTE IMMEDIATE sql_stmt INTO emp_name USING emp_id;  
    EXECUTE IMMEDIATE 'CREATE TABLE dept (id NUMBER)';  
END;
```

● 커서

- tbPSM에서 SQL을 수행하기 위해 하나의 문장마다 사용하는 내부 구조를 말한다.
- 일반적으로 커서에는 묵시적 커서와 명시적 커서, 커서 변수가 있다.
- %ISOPEN, %FOUND, %NOTFOUND, %ROWCOUNT 의 특별한 속성을 제공한다.

● 커서

- 묵시적 커서

```
BEGIN
    INSERT INTO emp VALUES (1, 'Susan');
    UPDATE emp SET name = 'Peter' WHERE id = 1;
    DELETE FROM emp WHERE id = 1;
END;
```

```
BEGIN
    FOR result IN (SELECT * FROM emp) LOOP
        DBMS_OUTPUT.PUT_LINE(result.id || '번 직원=' || result.name);
    END LOOP;
END;
```

● 커서

- 명시적 커서

```
CURSOR cursor_name IS SELECT ...;
CURSOR cursor_name (param1 TYPE [DEFAULT VALUE], ...) IS SELECT ...;
```

```
DECLARE
    CURSOR c1 IS SELECT * FROM emp;
BEGIN
    OPEN c1;
END;
```

```
DECLARE
    emp_rec emp%ROWTYPE;
    CURSOR c1 IS SELECT * FROM emp;
BEGIN
    OPEN c1;
    FETCH c1 INTO emp_rec;
    CLOSE c1;
END;
```

● 커서

- 커서 변수

```
TYPE csr_type_name IS REF CURSOR [RETURN type_name];  
csr_var_name csr_type_name;
```

```
DECLARE  
TYPE ref_csr IS REF CURSOR;  
c1 ref_csr;
```

```
OPEN csr_var_name FOR sql_query;
```

```
DECLARE  
TYPE ref_csr IS REF CURSOR;  
c1 ref_csr;  
BEGIN  
OPEN c1 FOR SELECT * FROM emp;  
OPEN c1 FOR SELECT part FROM dept;
```

● 커서

- 커서 속성

- 묵시적 커서

```
SQL%{ISOPEN | FOUND | NOTFOUND | ROWCOUNT}
```

- 명시적 커서

```
cursor_name%{ISOPEN | FOUND | NOTFOUND | ROWCOUNT}
```

```
BEGIN
    INSERT INTO emp VALUES (1, 'Susan');
    IF SQL%ISOPEN = FALSE THEN
        DBMS_OUTPUT.PUT_LINE('묵시적 커서는 항상 자동으로 닫힌다.');
    END IF;
END;
```

```
BEGIN
    UPDATE emp SET name = 'John' WHERE id = 1;
    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('1번 직원은 존재하지 않는다.');
    END IF;
END;
```

●OverView

- 에러 상황
 - 컴파일 중일 때
 - 프로그램이 실행 중일 때
- 에러 발생이 예상되는 부분을 예외 상황으로 정의하고 예외 처리 루틴을 만든다.

```
DECLARE
  a NUMBER := 5;
  b NUMBER := 0;
BEGIN
  a := a / b;
END;
```

결과

tbPSM : division by zero

```
DECLARE
  a NUMBER := 5;
  b NUMBER := 0;
BEGIN
  a := a / b;
EXCEPTION
  WHEN OTHERS THEN
    handle_numeric_error(b);
END;
```

● 시스템 정의 예외

```
DECLARE  
exception_identifier EXCEPTION;
```

예외 상황	설명
CASE_NOT_FOUND	CASE 문의 WHEN 절 중에서 조건을 만족하는 것이 없고 ELSE 절도 없는 경우
CURSOR_ALREADY_OPEN	이미 열려 있는 커서를 또다시 여는 경우
DUP_VAL_ON_INDEX	유일 키(UNIQUE) 제약조건이 선언되어 있는 컬럼에 중복된 값을 삽입하려는 경우
INVALID_CURSOR	열려 있지 않은 커서를 닫는 경우
NO_DATA_FOUND	SELECT INTO에 의한 질의에서 결과 로우가 하나도 없는 경우
TOO_MANY_ROWS	SELECT INTO에 의한 질의에서 결과 로우가 둘 이상인 경우
VALUE_ERROR	데이터 값의 변환(conversion), 절단(truncation), 제약조건 등과 관련된 에러가 발생한 경우
ZERO_DIVIDE	0으로 나누셈 연산을 수행하는 경우
COLLECTION_IS_NULL	초기화되지 않은 컬렉션 변수의 요소에 값을 대입하려는 경우 초기화되지 않은 컬렉션 변수에 EXISTS를 제외한 서브프로그램을 사용하는 경우
SUBSCRIPT_BEYOND_COUNT	컬렉션 변수에 있는 요소의 개수보다 큰 인덱스를 사용하는 경우
SUBSCRIPT_OUTSIDE_LIMIT	컬렉션 변수를 접근하는 인덱스가 유효하지 않은 경우 예:-1

● 시스템 정의 예외 예제

```
DECLARE
    employee_num NUMBER := 980180;
    employee_grade VARCHAR2(10) := 'S';
BEGIN
    CASE employee_grade
    WHEN 'A' THEN pay_bonus_a(employee_num);
    WHEN 'B' THEN pay_bonus_b(employee_num);
    WHEN 'C' THEN pay_bonus_c(employee_num);
    WHEN 'D' THEN pay_bonus_d(employee_num);
    END CASE;
EXCEPTION
    WHEN case_not_found THEN
        pay_bonus_special(employee_num);
END;
```

● 사용자 정의 예외

```
DECLARE  
    exception name EXCEPTION;
```

- RAISE 문의 사용

```
DECLARE  
    too_many_guest EXCEPTION;  
    cur_guest BINARY_INTEGER;  
    max_guest BINARY_INTEGER;  
BEGIN  
    /* guest_info 테이블에서 local이 New York인 현재 고객 수와 최대 고객 수를 찾는다. */  
    SELECT current_guest, max_guest INTO cur_guest, max_guest  
    FROM guest_info WHERE local = 'New York';  
  
    IF cur_guest > max_guest THEN  
        /* 만약 현재 고객 수가 최대 고객 수를 초과하면 예외 상황을 발생시킨다. */  
        RAISE too_many_guest;  
    END IF;  
END;
```

● 사용자 정의 예외

- RAISE_APPLICATION_ERROR 프로시저의 사용

```
RAISE_APPLICATION_ERROR
(
    error_number, error_message[, {true|false}]
);
```

```
DECLARE
    man_cnt NUMBER;
    woman_cnt NUMBER;
BEGIN
    SELECT count(*) INTO man_cnt FROM class WHERE gender = 'M';
    SELECT count(*) INTO woman_cnt FROM class WHERE gender = 'F';
    IF man_cnt != woman_cnt THEN
        RAISE_APPLICATION_ERROR(-20101, '남자와 여자의 비율이 맞지 않는다.');
    END IF;
END;
```

● 예외 처리 루틴

- 예외 상황이 발생하면 예외 처리 루틴으로 이동한다.
- 예외 처리 루틴이 실행된다.
- 실행이 끝나면, 외부 블록으로 이동한다.
- 외부 블록을 실행한다. 만약 외부 블록이 없으면 해당 프로그램은 종료된다.
- 프로그램을 종료한다.

● 예외 처리 루틴의 형식

```
BEGIN  
...  
EXCEPTION  
WHEN A THEN  
    실행문  
WHEN B THEN  
    실행문  
WHEN C THEN  
    실행문  
...  
END;
```

● 예외 상황의 전파

- 선언부에서 예외 상황이 발생한 경우

```
DECLARE
    tmp_number NUMBER(10) := 'STRING';
BEGIN
    ...
EXCEPTION
    /* VALUE_ERROR나 OTHERS를 포함하고 있으나 에러가 선언부에서 발생했으므로
       예외 처리 루틴은 실행되지 않는다. */
    WHEN VALUE_ERROR THEN
        ...
    WHEN OTHERS THEN
        ...
        /* 외부 블록이 없으므로 블록은 예외 상황을 처리하지 못한다.
           프로그램이 비정상적으로 종료된다. */
END;
```

● 예외 상황의 전파

- OTHERS 문을 이용한 예외 상황 처리

```
BEGIN
    DECLARE
        tmp_number VARCHAR2(10) := 'STRING';
    BEGIN
        ...
    EXCEPTION
        WHEN VALUE_ERROR THEN
            ...
        WHEN OTEHRS THEN
            ...
        END;
    EXCEPTION
        /* 내부 블록에서 전파된 예외 상황이 OTHERS 문에 의해 처리된다. */
        WHEN OTHERS THEN
            ...
            /* 프로그램은 정상적으로 종료된다. */
    END;
```

● 예외 상황의 전파

- RAISE 문을 이용한 예외 상황 처리

```
DECLARE
    a EXCEPTION;
    b EXCEPTION;
BEGIN
    RAISE a;
EXCEPTION
    WHEN a THEN
        /* 예외 상황 A를 처리하는 중에 예외 상황 B가 발생되었으므로
        예외 상황 B가 다시 외부 블록으로 전파된다. */
        RAISE b;
    WHEN b THEN
        ...
        /* 외부 블록이 없으므로, 예외 상황을 처리하지 못한 채로
        프로그램이 비정상적으로 종료된다. */
END;
```

● 에러 정보

-SQLCODE 함수

- 에러 코드 반환

-SQLERRM 함수

- 에러 메시지 반환

예외 상황	SQLCODE	SQLERRM
에러가 발생하지 않은 경우	0	normal, successful completion
사용자 정의 에러	1	user-defined exception
NO_DATA_FOUND	100	no data found
기타 예외 상황	해당 에러 코드	해당 에러 메시지

```
BEGIN
    copy_dept_info(20090430, 'Tibero');
EXCEPTION
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLCODE);
    /* 에러 코드를 반환한다. */

    DBMS_OUTPUT.PUT_LINE(SQLERRM(SQLCODE));
    /* 에러 코드에 대한 메시지를 반환한다. */
END;
```

Chapter 12장 tbJDBC 활용

- Java 프로그램 안에서 SQL 문장을 실행하기 위해 데이터베이스를 연결해주는 Application 프로그램의 인터페이스를 tbJDBC(Tibero의 Java database connectivity)라 한다.

- tbJDBC를 사용하기 위해서는 JDK1.4 이상이 반드시 설치되어 있어야 한다.

- JDK 다운로드 및 설치 방법

- <http://java.sun.com/javase/downloads/index.jsp>

● JDBC 1.0 및 JDBC 2.0

지원 기능	설명
✓ 인터페이스 메소드	- 노트부분 참조
✓ 결과 집합 기능	<ul style="list-style-type: none">- Scrolling 기능- 결과 집합 타입 : 다음과 같이 3가지 타입으로 지정할 수 있다.<ul style="list-style-type: none">▪ Forward-only▪ Scroll-insensitive▪ Scroll-sensitive- Concurrency 타입 : 결과 집합에 대해 다음과 같이 2가지 타입을 지정할 수 있다.<ul style="list-style-type: none">▪ Read-only▪ Updatable
✓ Batch update	<ul style="list-style-type: none">- 여러 개의 DML 문장을 한꺼번에 처리.- 개별로 INSERT 문을 수행하거나 준비된 문장(Prepared Statement)을 사용하여 파라미터만 바꿔가며 수행함으로써, 시스템 성능향상을 지원함.
✓ 데이터 타입	<ul style="list-style-type: none">- BLOB, CLOB 데이터 타입을 지원 (getBlob(), getBlob(), setBlob(), setBlob() 메소드 지원)

● JDBC 3.0

지원 기능	설명
✓ 인터페이스 메소드	- 노트부분 참조
✓ 저장점	- Savepoint 인터페이스를 구현하여 임의의 트랜잭션에 대한 저장점 설정과 커밋 및 롤백 기능을 제공
✓ 접속 풀링 / 문장 풀링	- ConnectionPoolDataSource 인터페이스에서 설정할 수 있는 내용을 지원한다. 사용자는 DataSource 객체에 의해 PooledConnection이 어떠한 특성을 갖고 생성될지를 결정할 수 있다.
✓ 파라미터 메타데이터	- Prepared Statement)에서 사용한 파라미터 개수의 정보를 제공
✓ 자동 생성 키	- SQL 문장을 실행한 후 결과 집합으로부터 getGeneratedKeys 함수를 사용. 결과 로우에 대한 키 또는 자동으로 생성된 컬럼의 값을 얻는다.
✓ 이름 있는 파라미터	- CallableStatement 객체의 파라미터를 저장할 때 파라미터 이름으로 구별할 수 있는 이름 있는 파라미터(named parameter) 기능을 지원
✓ 결과 집합의 유지성	- 결과 집합이 열려있는 동안에 커밋이 발생했을 때, 결과 집합을 그대로 유지할지 아니면 닫을지를 설정
✓ 복수 개의 결과 집합 반환	- 한 SQL 문장이 여러 개의 결과 집합을 열 수 있도록 지원
✓ BLOB와 CLOB 객체에 존재하는 데이터의 수정	- updateXXX API를 통해 BLOB와 CLOB 객체에 포함된 데이터를 수정하는 기능을 제공한다.

● JDBC 4.0

지원하는 기능	설명
✓ 인터페이스 메소드	- 노트부분 참조
✓ XML 데이터 타입	- SQLXML 인터페이스를 통해 SQL:2003에 추가된 XML 데이터 타입을 사용할 수 있다.
✓ 자동 드라이버 검출	- 자동으로 드라이버를 로딩할 수 있게 되어 Class.forName 를 사용한 java.sql.Driver 클래스의 로드 없이도 드라이버 객체를 사용할 수 있다.
✓ 국가별 캐릭터 세트 지원	- 데이터베이스에서 별도로 지정해 사용하는 국가별 캐릭터 세트를 지원하기 위한 API가 추가되었다.
✓ 향상된 SQLException	- 연쇄적으로 연결된 예외를 생성할 수 있게 되어 보다 자세한 원인을 전달해 줄 수 있으며, 새로운 종류의 예외 타입이 추가되었다.
✓ 향상된 Blob/Clob 기능	- Blob/Clob 객체를 생성/해제할 수 있는 API를 지원한다.
✓ SQL ROWID 데이터 타입의 지원	- RowId 인터페이스를 이용하여 SQL ROWID 타입을 사용할 수 있다.
✓ 실제 JDBC 객체에 대한 접근 허용	- Wrapper 인터페이스를 이용하여 어플리케이션 서버나 접속 풀링 환경에서도 실제 JDBC 객체에 접근해 사용할 수 있다.
✓ 접속 풀링 환경에서 실제 접속 상태에 대한 통지	- 접속 풀링 환경에서 실제 접속이 닫히거나 유효하지 않게 되었을 때, 그 상태를 풀링된 문장에 통지해 준다.

패키지 임포트

데이터베이스 연결

Statement 객체 생성

질의문 수행과 ResultSet 객체받기

ResultSet 객체 처리

커밋 또는 롤백 수행

ResultSet 객체와 Statement 객체 소멸

데이터베이스 연결 해제

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;

public class JdbcTest
{
    Connection conn;
    public static void main(String[] args) throws Exception
    {
        JdbcTest test = new JdbcTest();
        test.connect();
        test.executeStatement();
        test.executePreparedStatement();
        test.executeCallableStatement();
        test.disconnect();
    }
    /* ... 기능별 멤버 함수를 구현한다. */
}
```

● 접속

```
private void connect() throws SQLException, ClassNotFoundException
{
    Class.forName("com.tmax.tibero.jdbc.TbDriver");

    conn = DriverManager.getConnection(
        "jdbc:tibero:thin:@localhost:8629:tibero", "tibero", "tmax");

    if (conn == null)
    {
        System.out.println("connection failure!");
        System.exit(-1);
    }
    System.out.println("Connection success!");
}
```

● 실행1

```
private void executeStatement() throws SQLException
{
    String dropTable = "drop table emp";

    String createTable = "create table emp (id number, "+
                          " name varchar(20), salary number)";

    String InsertTable = "insert into emp values(1000, 'Park', 5000)";

    Statement stmt = conn.createStatement();

    try {
        stmt.executeUpdate(dropTable);
    } catch(SQLException e) {
        // if there is not the table
    }
    stmt.executeUpdate(createTable);
    stmt.executeUpdate(InsertTable);
    stmt.close();
}
```

● 실행2

```
private void executePreparedStatement() throws SQLException
{
    PreparedStatement pstmt = conn
        .prepareStatement("select name from emp where id = ?");

    pstmt.setString(1, "1000");

    ResultSet rs = pstmt.executeQuery();

    while (rs.next()) {
        System.out.println(rs.getString(1));
    }
    pstmt.close();
}
```

●호출

```
private void executeCallableStatement() throws SQLException
{
    String callSQL =
        " CREATE PROCEDURE testProc "+"(ID_VAL IN NUMBER, SAL_VAL IN OUT NUMBER) as "
        +
        " BEGIN" + " update emp" + " set salary = SAL_VAL" + " where id = ID_VAL;" +
        " select salary into SAL_VAL" + " from emp" + " where id = ID_VAL;" + " END;";

    String dropProc = "DROP PROCEDURE testProc";
    Statement stmt = conn.createStatement();
    try {
        stmt.executeUpdate(dropProc);
    } catch(SQLException e) {
        // if there is not the procedure
    }
    stmt.executeUpdate(callSQL);
    CallableStatement cstmt = conn.prepareCall("{call testProc(?, ?)}");
    cstmt.setInt(1, 1000);
    cstmt.setInt(2, 7000);
    cstmt.registerOutParameter(2, Types.INTEGER);
    cstmt.executeUpdate();
    int salary = cstmt.getInt(2);
    System.out.println(salary);
    stmt.close();
    cstmt.close();
}
```

● COMMIT & ROLLBACK

```
conn.setAutoCommit(false);
conn.rollback();
conn.commit();
conn.setTransactionIsolation(Connection.TRANSACTION_READ_UNCOMMITTED)
```

● 접속 해제

```
private void disconnect() throws SQLException
{
    if (conn != null)
        conn.close();
}
```

Chapter 13장 tbCLI 활용

● tbCLI란?

- tbCLI는 Tibero가 제공하는 Call Level Interface(CLI) 이다.

● 특징

- 일반적인 프로그램 언어와 SQL 문장의 장점을 융합한 인터페이스
- 모듈 효율적 관리 및 가독성 향상
- 애플리케이션 패키지를 바인딩 할 필요 없음
- 접근할 데이터베이스의 통계 사용 가능
- 쓰레드의 안정성 보장

● 클라이언트 / 서버 환경에 유용



●tbCLI 핸들

- 주요 데이터 구조에 대한 포인터(pointer)

●tbCLI 함수

- tbCLI 프로그램에서 데이터베이스 작업을 수행하기 위해서 사용
- SQLExecDirect 함수 프로토 타입 예시

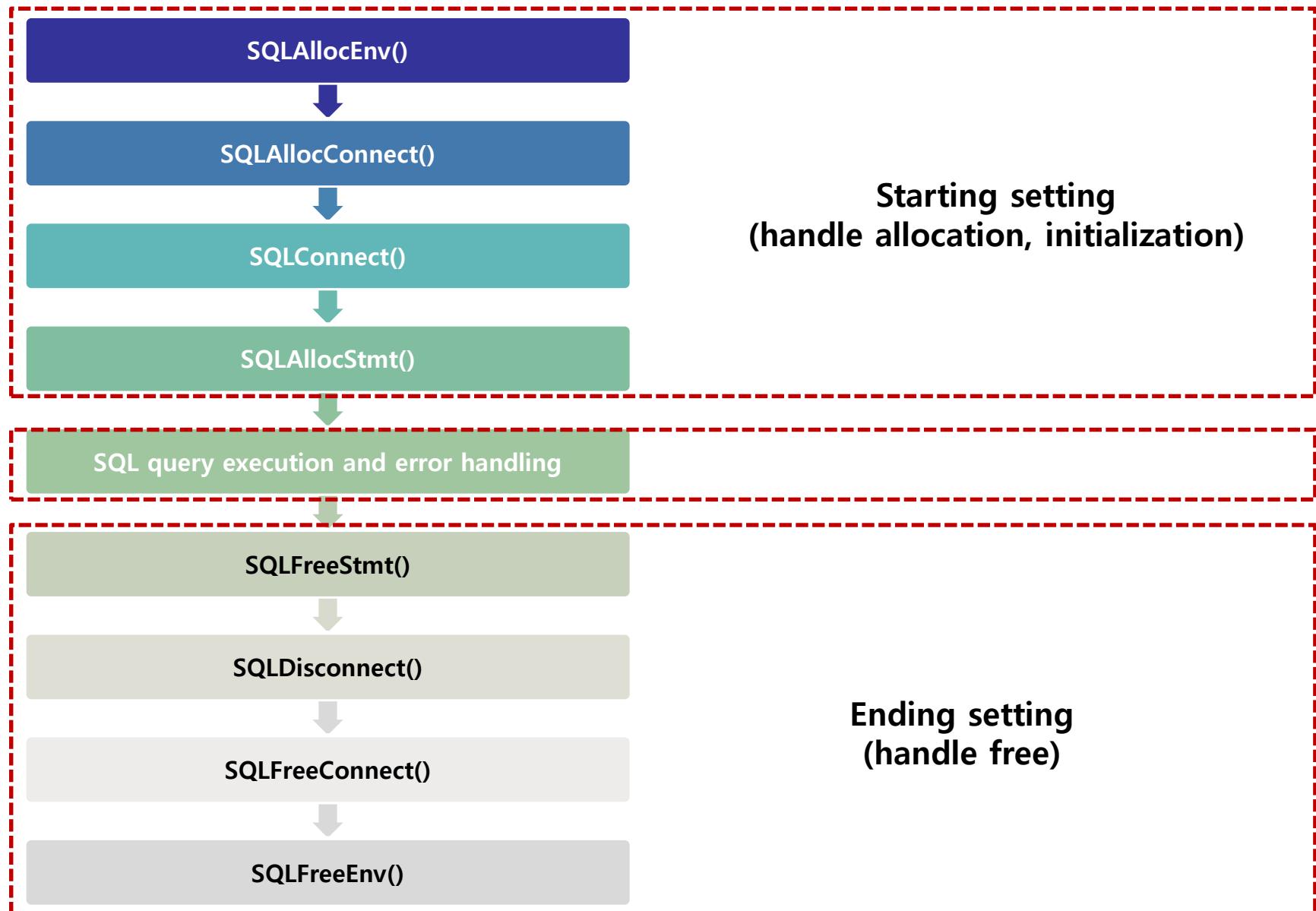
```
SQLRETURN SQLExecDirect(SQLHSTMT StatementHandle, SQLCHAR  
*SQLString,SQLINTEGER SQLStringSize);
```

●tbCLI Error Message

- 진단 레코드를 생성하여 다양한 정보 제공

Program 구조 (1/4)

CHAPTER 13장. tbCLI 활용



● 시작 설정 부분 (Starting setting)

- 환경 핸들 할당
- 연결 핸들 할당
- 데이터 소스와 연결 수행

● 예시

```
SQLHENV h_env;
SQLHDBC hdbc;
SQLRETURN rc = SQL_SUCCESS;

...
rc = SQLAllocHandle(SQL_HANDLE_ENV, NULL, &h_env); 1
if (rc != SQL_SUCCESS) ...
rc = SQLAllocHandle(SQL_HANDLE_DBC, h_env, &hdbc); 2
if (rc != SQL_SUCCESS) ...
rc = SQLConnect(hdbc, (SQLCHAR *)ds_name, SQL_NTS, (SQLCHAR *)user,SQL_NTS,
(SQLCHAR *)passwd, SQL_NTS); 3
if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO) ...
```

```
SQLHSTMT h_stmt;
...
rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &h_stmt); 4
if (rc != SQL_SUCCESS) ...
```

● SQL 문장 실행 및 에러 처리 부분

- 직접 실행
- 준비된 실행

● 예시

```
SQLCHAR *update = "UPDATE EMPLOYEE SET SALARY = SALARY * 1.05  
                      WHERE DEPTNO = 5";  
rc = SQLExecDirect(h_stmt, update, SQL_NTS);  
if (rc != SQL_SUCCESS) ...
```

```
SQLCHAR *update = "UPDATE EMP SET SALARY = SALARY * ? "  
"WHERE DEPTNO = ?";  
double ratio = 0.0;  
short deptno = 0;  
...  
rc = SQLPrepare(h_stmt, update, SQL_NTS); ①  
if (rc != SQL_SUCCESS) ...  
rc = SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,  
                      SQL_DOUBLE, 5, 2, &ratio, 0, NULL);  
if (rc != SQL_SUCCESS) ...  
rc = SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SHORT,  
                      SQL_SMALLINT, 0, 0, &deptno, 0, NULL);  
if (rc != SQL_SUCCESS) ...  
ratio = 1.05;  
deptno = 5;  
SQLExecute(h_stmt); ②  
if (rc != SQL_SUCCESS) ...
```

● 종료 설정 부분 (Ending setting)

- 시작 설정 부분에서 수행한 작업과 반대 작업 수행

● 예시

```
rc = SQLDisconnect(h_dbc); 1
if (rc != SQL_SUCCESS) ...
SQLFreeHandle(SQL_HANDLE_DBC, h_dbc);
SQLFreeHandle(SQL_HANDLE_ENV, h_env);
```

2

3

- SQL 문장에 값을 입력하고, 질의 결과를 얻기 위해 사용

- Tibero Data Type

Database에 저장된 데이터에 접근할 때 사용

- tbCLI Data Type

Application Program에서 데이터 조작할 때 사용

● Tibero Data Type

구분	Data Type	설명
문자형	CHAR, VARCHAR, RAW	문자열을 표현하는 데이터 타입이다.
숫자형	NUMBER, INTEGER, FLOAT	정수나 실수의 숫자를 저장하는 데이터 타입이다.
날짜형	DATE, TIME, TIMESTAMP	시간이나 날짜를 저장하는 데이터 타입이다.
대용량 객체형	BLOB, CLOB	LOB 타입을 의미한다. 다른 데이터 타입이 지원하는 최대 길이(8KB 이하) 보다 훨씬 큰 길이를 가질 수 있는 객체이다. 4GB까지 가능하다.
내재형	ROWID	사용자가 명시적으로 선언하지 않아도 Tibero가 자동으로 삽입되는 로우마다 포함하는 컬럼의 타입이다.

●tbCLI Data Type

C의 typedef 이름	C의 데이터 타입
SQLCHAR	unsigned char
SQLSCHAR	signed char
SQLSMALLINT	short int
SQLUSMALLINT	unsigned short int
SQLINTEGER	long int
SQLUINTEGER	unsigned long int
SQLREAL	float
SQLDOUBLE, SQLFLOAT	double
DATE_STRUCT, SQL_DATE_STRUCT	
TIME_STRUCT, SQL_TIME_STRUCT	
TIME_STAMP_STRUCT, SQL_TIMESTAMP_STRUCT	

● SQL-99 표준 API

- SQLAllocHandle, SQLBindParameter 등

● Tibero API

- SQLAllocHandle2, SQLLobClose 등

● 반환 코드

- 모든 tbCLI 함수는 실행 후 반환 코드를 반환한다.
- 반환 코드는 SQLRETURN 타입이며 미리 정해진 값 중의 하나이다.

반환 코드	설명
SQL_SUCCESS	함수가 성공적으로 완료된 상태
SQL_SUCCESS_WITH_INFO	함수가 성공적으로 완료되었으나, 경고 메시지가 있는 상태
SQL_NO_DATA	함수가 성공적으로 완료되었으나, 관련된 데이터를 찾을 수 없는 상태
SQL_INVALID_HANDLE	입력 파라미터에 주어진 핸들이 유효하지 않은 상태
SQL_NEED_DATA	SQL 문장을 실행하기 위해 데이터가 더 필요한 상태
SQL_STILL_EXECUTING	이전에 실행한 SQL 문장이 완료되지 않은 상태
SQL_ERROR	치명적인 에러가 발생한 상태

● ODBC(Open Database Connectivity)란?

- Call-Level Interface(CLI) 명세와 표준을 따르는 데이터베이스의 API.

The Open Group CAE Specification "Data Management: SQL Call-Level Interface (CLI)" ISO/IEC 9075-3:1995 (E) Call-Level Interface (SQL/CLI)

● What Useful?

- 동일한 Application Program 으로 다양한 벤더의 DBMS에 접근 가능.
- 단, 벤더별 모듈이나 드라이버 필요

- tbCLI와 ODBC 연동

- Windows 계열
- UNIX 계열(LINUX 포함)

- Tibero에서는 ODBC와의 연동을 위해 운영체제별로 ODBC 드라이버를 제공한다.

● Tbnet_alias.tbr과 ODBC 데이터 원본 관리자

- 데이터베이스 접속 정보 갖고 오는 방법

1. tbdsn.tbr

```
tibero=(  
    (INSTANCE=(HOST=localhost)  
     (PORT=8629)  
     (DB_NAME=tibero)  
 )  
)
```

2. ODBC 데이터 원본 관리자의 DSN

● Windows 계열에서의 설치

Tibero ODBC Driver 설치

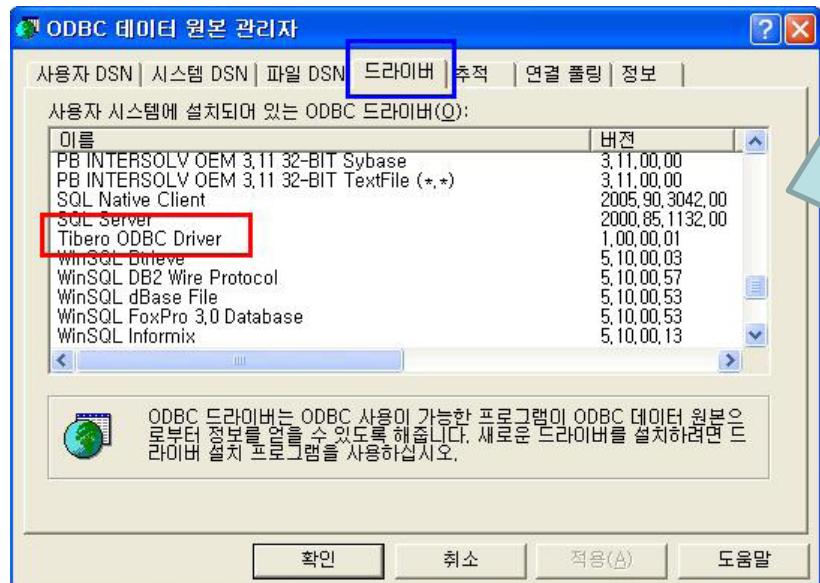
1. \$TB_HOME/client/lib 디렉터리에 있는 libtbcli.dll 파일을 설치하려는 폴더에 복사한다.
2. 명령 프롬프트(cmd.exe) 창을 실행한 후, 다음과 같은 명령을 실행한다.

```
$TB_HOME/client/bin/odbc_driver_install.exe -i [설치 경로]
```

설치 경로를 입력하지 않으면 디폴트로 설정된 %SystemRoot%\System32 폴더에 설치된다.

● Windows 계열에서의 설치

설치 확인

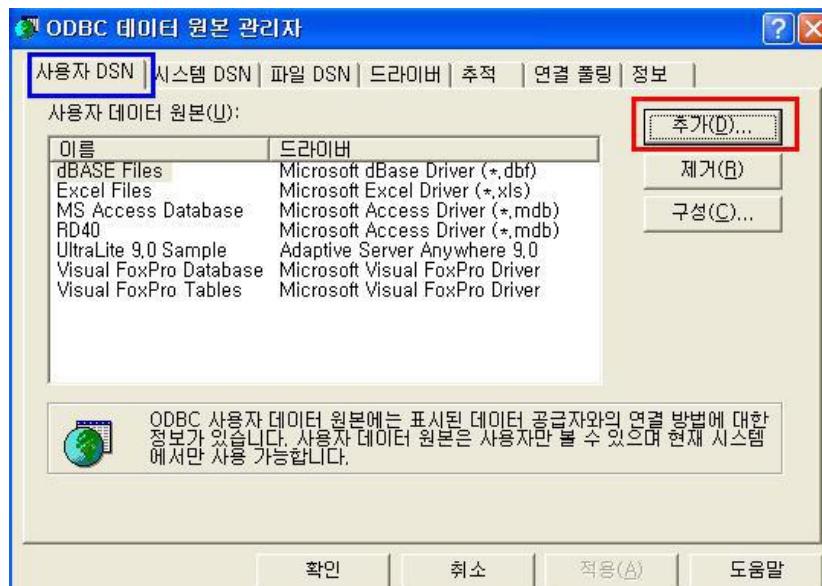


1. ODBC 데이터 원본 관리자 실행
 - 제어판 > 관리도구 > 데이터 원본 (ODBC) 메뉴 선택
2. Tibero ODBC Driver 확인

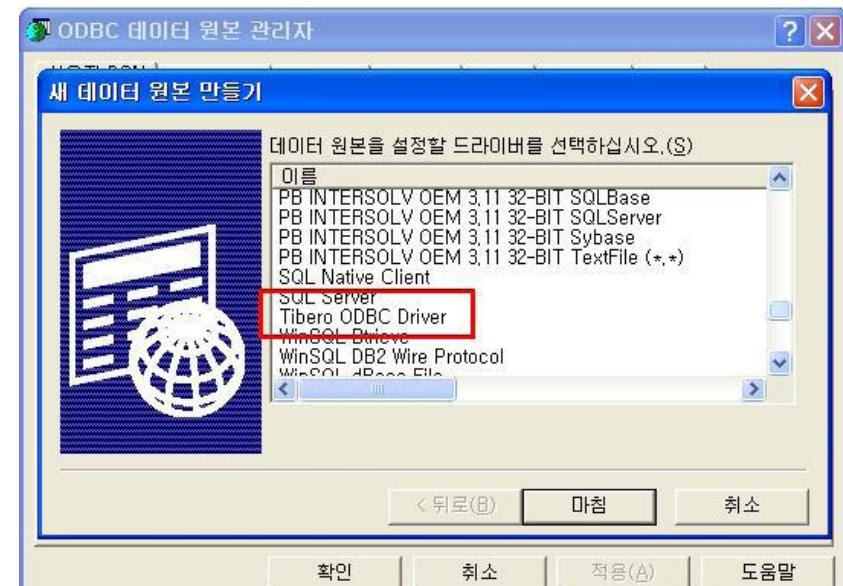
● Windows 계열에서의 설치

DSN 등록

- ◆ DSN(Data Source Name)를 등록하여 데이터베이스 접속 정보를 저장한다.



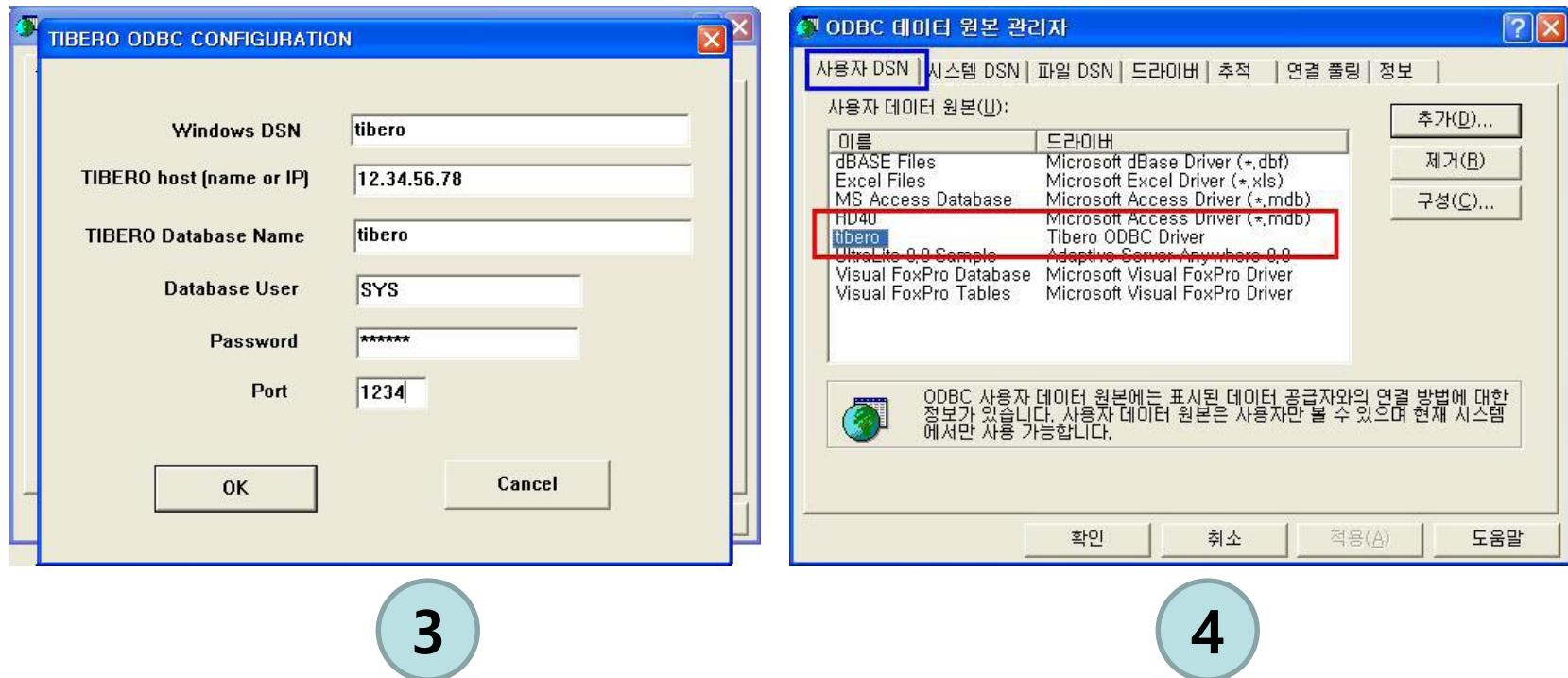
1



2

●Windows 계열에서의 설치

DSN 등록



●UNIX 계열에서의 설치

ODBC 드라이버 관리자 다운로드

1. <http://iodbc.org>
2. Libiodbc-3.52.6.tar.gz 형태의 파일 다운로드

기본 환경 설정

1. \$TB_HOME/client/lib (또는 \$TB_HOME/client/lib32) 디렉터리에 libtbodbc.so 파일이 존재하는지 확인한다.
2. LD_LIBRARY_PATH 환경 변수에 libtbodbc.so 파일이 존재하는 경로를 추가한다.

●UNIX 계열에서의 설치

iODBC 설치

1. 32-bit

```
export CC=cc  
.configure [--prefix=/usr/local] --sysconfdir=/etc --disable-gui  
make  
make install
```

2. 64-bit

```
export CFLAGS=+DD64  
export CC=cc  
.configure [--prefix=/usr/local] --sysconfdir=/etc --disable-gui  
make  
make install
```

HP

```
export CFLAGS=-q64  
export CC=cc  
.configure [--prefix=/usr/local] --sysconfdir=/etc --disable-gui  
make  
make install
```

AIX

●UNIX 계열에서의 설치

iODBC 설치 확인

```
$ file libtbodbc.so  
libtbodbc.so: ELF 32-bit LSB shared object, Intel 80386,  
version 1 (SYSV), not stripped
```

```
$ file libiodbcinst.so.2.1.18  
libiodbcinst.so.2.1.18: ELF 32-bit LSB shared object, Intel 80386,  
version 1 (SYSV), not stripped
```

●UNIX 계열에서의 설치

Tibero를 위한 iODBC 환경설정

1. ODBC 드라이버를 설정한다. (아래의 예를 참고하여 /etc 디렉터리에 odbc.ini 파일을 생성한다.)

```
[ODBC Data Source]
Tibero6 = Tibero6 ODBC Driver
[ODBC]
Trace = 1
TraceFile = /tmp/odbc.trace
[Tibero6]
Driver = /home/tibero6/client/lib/libtbodbc.so
Description = Tibero6 ODBC Datasource
SID = tibero
User = tibero
Password = tmax
```

●UNIX 계열에서의 설치

Tibero를 위한 iODBC 환경설정

2. ODBC 데이터 소스를 설정한다. (아래의 예를 참고하여 /etc 디렉터리에 odbcinst.ini 파일을 생성한다.)

[Tibero6 ODBC Driver]

Description = ODBC Driver for Tibero6

Driver = /home/tibero6/client/lib/libtbodbc.so

UsageCount = 1

3. 1, 2번 과정을 통해 생성된 ODBC 드라이버와 ODBC 데이터 소스를 iODBC 관리자에 등록한다.

```
iodbc-config --odbconfig --odbconfigini
```

●UNIX 계열에서의 설치

데이터베이스 접속 확인

```
iodbctest "DSN={dsn};UID={user};PWD={pwd}
```

```
$ iodbctest "DSN=Tibero6;UID=tibero;PWD=tmax"  
iODBC Demonstration program  
This program shows an interactive SQL processor  
Driver Manager: 03.52.0607.1008  
Driver: 04.00.0000 (libtbodbc.so)  
SQL>select * from dual;  
DUMMY  
-----  
X  
result set 1 returned 1 rows.  
SQL>
```

Chapter 14장

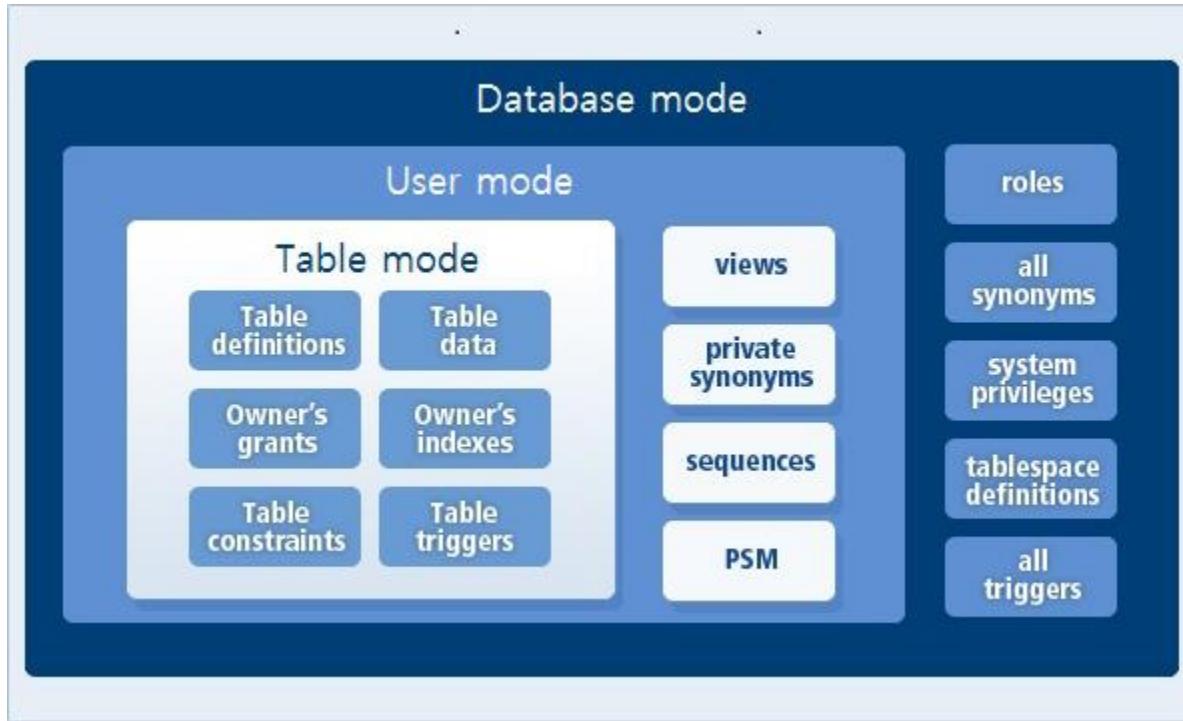
유틸리티

- 데이터 베이스에 저장데이터베이스에 저장된 스키마 객체의 전체 또는 일부를 추출해 고유 형식의 파일로 저장하므로, 데이터베이스의 백업과 다른 머신 간의 데이터베이스를 전송할 때 유용.

- 특징

- 데이터 베이스, 사용자, 테이블 모드로 사용 가능
- 테이블 정의를 저장 가능
- 테이블 재구성 가능
- 논리적인 백업 가능
- 서로 다른 시점의 데이터

● Export 모드



● 실행 (\$TB_HOME/client/bin)

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

● 파라미터 설명

항목	기본값	설명
CFGFILE		환경설정 파일의 이름이다.
USERNAME		Export를 수행하는 사용자의 계정을 입력한다.
PASSWORD		Export를 수행하는 사용자의 패스워드를 입력한다.
IP	LOCALHOST	Export 대상 Tibero 서버의 IP를 입력한다.
PORT	8629	Export 대상 Tibero 서버의 포트를 입력한다.
SID		Export 대상 Tibero 서버의 SID를 입력한다.
FILE	DEFAULT.DAT	Export를 수행할 때 생성되는 파일의 이름이다. 바이너리 파일의 형태로 운영체제에서 생성되며, 이름을 지정하지 않으면 기본값으로 생성된다.
NO_PACK_DIR		압축을 해제한 덤프 파일이 저장되는 디렉터리이다. 이옵션이 지정되면, FILE 파라미터에 설정된 값은 무시된다.
OVERWRITE	N	Export를 수행할 때 생성되는 파일과 동일한 이름의 파일이 이미 존재하는 경우 파일을 덮어쓸지 지정한다. – Y: 파일을 덮어쓴다. – N: 파일을 덮어쓰지 않는다.
LOG		Export의 로그가 기록될 파일의 이름을 입력한다.
FULL	N	전체 데이터베이스 모드로 Export를 수행할지 지정한다. – Y: 전체 데이터베이스 모드로 Export를 수행한다. – N: 사용자 또는 테이블 모드로 Export를 수행한다. (둘중 하나의 모드는 있어야 함)
USER		사용자 모드로 Export를 수행할 때 Export될 객체의 소유자를 지정한다. USER=userlist의 형태로 사용한다.
TABLE		테이블 모드로 Export를 수행할 때 Export할 대상 테이블의 이름을 지정한다. TABLE=tablelist의 형태로 사용한다.

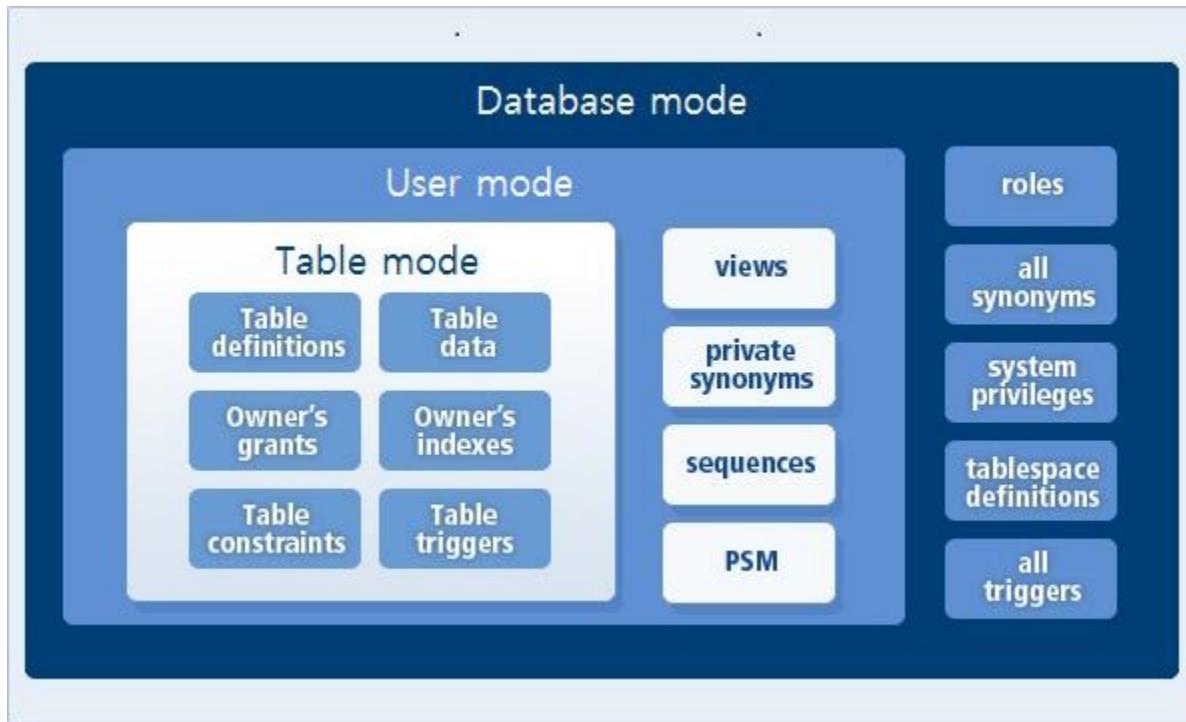
항목	기본값	설명
QUERY		<p>Export될 데이터에 필터 조건을 지정한다.</p> <ul style="list-style-type: none"> - 모드에 상관없이 동작하지만, 원하지 않는 테이블에도 적용될 수 있으므로 주의한다. - Where 조건 앞과 뒤를 "W"로 감싸주어야 한다. - 지정된 조건에 의해 SQL 문장에서 문법(Syntax) 에러가 발생할 경우, 조건을 적용하지 않고 다시 시도한다.
INDEX		<p>Export를 수행할 때 인덱스 정보의 Export 여부를 지정한다.</p> <ul style="list-style-type: none"> - Y: 인덱스를 Export한다. - N: 인덱스를 Export하지 않는다.
GRANT	Y	<p>Export를 수행할 때 권한의 Export 여부를 지정한다.</p> <ul style="list-style-type: none"> - Y: 권한을 Export한다. - N: 권한을 Export하지 않는다.
CONSTRAINT	Y	<p>Export를 수행할 때 제약조건의 Export 여부를 지정한다.</p> <ul style="list-style-type: none"> - Y: 제약조건을 Export한다. - N: 제약조건을 Export하지 않는다.
ROWS	Y	<p>Export를 수행할 때 테이블의 데이터를 Export 할지 여부를 지정한다.</p> <ul style="list-style-type: none"> - Y: 테이블의 데이터를 Export한다. - N: 테이블의 데이터를 Export하지 않는다.
SCRIPT	N	<p>Export를 수행할 때 스키마 객체를 생성하는 DDL 스크립트의 표시 여부를 지정한다.</p> <ul style="list-style-type: none"> - Y: 스키마 객체를 생성하는 DDL 스크립트를 표시한다. - N: 스키마 객체를 생성하는 DDL 스크립트를 표시하지 않는다.
THREAD_CNT	4	테이블의 데이터를 Export하기 위해 사용하는 스레드의 개수를 입력한다.
SERVER_VER	5	Export의 대상이 되는 Tiberio의 버전을 지정한다.
PARALLEL_DEGREE	0	테이블의 데이터를 Export하기 위해 수행하는 질의의 parallel hint를 입력한다.
PACK_TYPE	TAR	<p>패키징에 사용할 알고리즘을 지정한다.</p> <ul style="list-style-type: none"> - TAR: TAR 형식으로 패키징한다. - ZIP: ZIP 형식으로 패키징한다.

- 외부 파일에 저장된 스키마 객체를 Tibero 데이터베이스에 다시 저장하므로, tbExport 유틸리티와 함께 데이터베이스의 백업과 다른 머신 간의 데이터베이스 전송 등을 할 때 유용.

● 특징

- 데이터 베이스, 사용자, 테이블 모드로 사용 가능
- 논리적인 백업 가능
- 서로 다른 시점의 데이터
- JVM(Java Virtual Machine) 설치 필요

● Import 모드



● 실행 (\$TB_HOME/client/bin)

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

● 파라미터 설명

항목	기본값	설명
CFGFILE		환경설정 파일의 이름이다.
USERNAME		Import를 수행하는 사용자의 계정을 입력한다.
PASSWORD		Import를 수행하는 사용자의 패스워드를 입력한다.
IP	LOCALHOST	Import 대상 Tibero 서버의 IP를 입력한다.
PORT	8629	Import 대상 Tibero 서버의 포트를 입력한다.
SID		Import 대상 Tibero 서버의 SID를 입력한다.
FILE	DEFAULT.DAT	Import를 수행할 때 생성되는 파일의 이름이다. 바이너리 파일의 형태로 운영체제에서 생성되며, 이름을 지정하지 않으면 기본값으로 생성된다.
NO_PACK_DIR		압축을 해제한 덤프 파일이 저장되는 디렉터리이다. 이옵션이 지정되면, FILE 파라미터에 설정된 값은 무시된다.
LOG		Export의 로그가 기록될 파일의 이름을 입력한다.
FULL	N	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)
USER		사용자 모드로 Export를 수행할 때 Export될 객체의 소유자를 지정한다. USER=userlist의 형태로 사용한다.
FROMUSER		From to User 모드에서 사용하며 Export할 때 사용된 객체의 원래 소유자를 지정한다. FROMUSER=userlist의 형태로 사용한다.
TOUSER		From to User 모드에서 사용하며 Import를 수행할 때 Import할 소유자를 지정한다. TOUSER=userlist의 형태로 사용한다.
TABLE		테이블 모드로 Export를 수행할 때 Export할 대상 테이블의 이름을 지정한다. TABLE=tablelist의 형태로 사용한다.

항목	기본값	설명
INDEX		Export를 수행할 때 인덱스 정보의 Export 여부를 지정한다. – Y: 인덱스를 Export한다. – N: 인덱스를 Export하지 않는다.
GRANT	Y	Export를 수행할 때 권한의 Export 여부를 지정한다. – Y: 권한을 Export한다. – N: 권한을 Export하지 않는다.
CONSTRAINT	Y	Export를 수행할 때 제약조건의 Export 여부를 지정한다. – Y: 제약조건을 Export한다. – N: 제약조건을 Export하지 않는다.
ROWS	Y	Export를 수행할 때 테이블의 데이터를 Export 할지 여부를 지정한다. – Y: 테이블의 데이터를 Export한다. – N: 테이블의 데이터를 Export하지 않는다.
DPL	N	DPL 방법으로 Import할지 여부를 지정한다. – Y: DPL 방법을 사용한다. – N: DPL 방법을 사용하지 않는다.
PIPELINING	N	PIPELINING(with DPL) 기능을 사용하여 Import할지 여부를 지정한다. – Y: PIPELINING(with DPL) 기능을 사용한다. – N: PIPELINING(with DPL) 기능을 사용하지 않는다.
SCRIPT	N	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.
THREAD_CNT	4	테이블의 데이터를 Export하기 위해 사용하는 스레드의 개수를 입력한다.
SERVER_VER	5	Export의 대상이 되는 Tibero의 버전을 지정한다.
IGNORE	N	Import를 수행할 때 이미 존재하는 스키마 객체로 인한 생성에러를 무시한다. – Y: 이미 존재하는 스키마 객체로 인한 생성 에러를 무시한다. – N: 이미 존재하는 스키마 객체로 인한 생성 에러를 무시하지 않는다.
IO_BUF_SIZE	16M	Import를 실행할 때 파일의 입/출력에 사용되는 버퍼의 크기를 조절한다.
BIND_BUF_SIZE	1M	Import를 DPL 모드로 실행할 때 stream에서 사용하는 bind buffer의 크기를 조절한다.

감사합니다.



Korea First, World Best, **TIBERO !!**