Tibero RDBMS

관리자 안내서

Tibero RDBMS 4 SP1

안내서 정보

안내서 제목: Tibero RDBMS 관리자 안내서

발행일: 2013-02-25

소프트웨어 버전: Tibero RDBMS 4 SP1

안내서 버전: 2.1.4

내용 목차

그림 목차

예 목차

안내서에 대하여

안내서의 대상

본 안내서는 Tibero RDBMS®(이하 Tibero RDBMS)를 사용하여 데이터베이스를 생성하고 원활한 Tibero RDBMS의 동작을 보장하려는 데이터베이스 관리자(Database Administrator, 이하 DBA)를 대상으로 기술한다.

안내서의 전제 조건

본 안내서는 Tibero RDBMS를 관리하는 방법을 설명한 안내서이다.

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

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

안내서의 제한 조건

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

참고

Tibero RDBMS의 설치 및 환경 설정에 관한 내용은 "Tibero RDBMS 설치 안내서"를 참고한다.

안내서 구성

Tibero RDBMS 관리자 안내서는 총 12개의 장과 Appendix로 구성되어 있다.

각 장의 주요 내용은 다음과 같다.

- 제1장: Tibero RDBMS 소개

  DBA 측면에서 Tibero RDBMS의 기본 개념과 이를 구성하는 프로세스와 디렉터리 구조를 기술한다.

- 제2장: 관리의 기본

  Tibero RDBMS를 관리하기 위한 기본적인 사항을 기술한다.

- 제3장: 파일과 데이터의 관리

  Tibero RDBMS의 파일과 데이터를 관리하는 방법을 기술한다.

- 제4장: 스키마 객체의 관리

  Tibero RDBMS의 데이터베이스를 구성하는 데 필요한 스키마 객체를 관리하는 방법을 기술한다.

- 제5장: 사용자 관리와 데이터베이스 보안

  데이터베이스 보안을 위한 사용자, 특권, 역할을 생성하고 이를 관리하는 방법을 기술한다.

- 제6장: 백업과 복구

  시스템의 예상치 못한 오류로부터 대비하기 위해 다양한 백업 및 복구 과정을 기술한다.

- 제7장: 분산 트랜잭션

  분산 트랜잭션을 지원하기 위해 Tibero RDBMS가 제공하는 기능을 기술한다.

- 제8장: Tibero Standby Cluster

  고가용성, 자료 보호, 재난 복구를 위한 Tibero Standby Cluster를 기술한다.

- 제9장: Tibero Cluster Manager

  클러스터 관리를 편리하게 하고, 가용성을 높이기 위해 제공하는 Tibero RDBMS의 부가 기능인 Tibero Cluster Manage를 기술한다.

- 제10장: Tibero Active Cluster

  확장성, 고가용성을 목적으로 하는 Tibero Active Cluster를 기술한다.

- 제11장: 데이터 암호화

  데이터 암호화 기능을 사용하고 이를 관리하는 방법을 기술한다.

- 제12장: 통신 암호화

  Tibero RDBMS에서 통신 암호화 기능을 사용하고 관리하기 위한 방법을 설명한다.

- 제13장: Parallel Execution

  Parallel Execution의 기본개념과 동작원리를 소개하고, 이를 유형별로 실행하는 방법을 기술한다.

- 제14장: Automatic Performance Monitoring

  Tibero의 성능 진단을 위해서 제공되는 APM(Automatic Performance Monitoring)에 대해서 설명한다.

- Appendix A: tbdsn.tbr

  클라이언트가 Tibero RDBMS에 접속하기 위해 필요한 tbdsn.tbr 환경설정 파일을 기술한다.

- Appendix B: V$SYSSTAT

  동적 뷰 V$SYSSTAT에 포함된 시스템의 각종 통계 정보를 기술한다.

- Appendix C: 클라이언트 환경 변수

  클라이언트에서 설정할 수 있는 환경 변수에 대한 설명이다.

- Appendix D: 문제 해결

  Tibero RDBMS를 사용할 때 발생할 수 있는 문제를 해결하는 방법을 기술한다.

안내서 규약

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

시스템 사용 환경

	요구 사항
Platform	HP-UX 11i (PA-RISC, ia64)
	Solaris (SPARC 9/Solaris 9)
	AIX (PPC 5L/AIX 5.3)
	GNU (X86, 64, IA64)
	Linux kernel 2.6 이상
Hardware	최소 1.5GB 하드디스크 공간
	512MB 이상 메모리 공간
Compiler	PSM (C99 지원 필요)
	tbESQL/C (C99 지원 필요)

관련 안내서

안내서	설명
Tibero RDBMS 설치 안내서	설치 시 필요한 시스템 요구사항과 설치 및 제거 방법을 기술한 안내서이다.
Tibero RDBMS tbCLI 안내서	Call Level Interface인 tbCLI의 개념과 구성요소, 프로그램 구조를 소개하고 tbCLI 프로그램을 작성하는 데 필요한 데이터 타입, 함수, 에러 메시지를 기술한 안내서이다.
Tibero RDBMS 애플리케이션 개발자 안내서	각종 애플리케이션 라이브러리를 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero RDBMS External Procedure 안내서	External Procedure를 소개하고 이를 생성하고 사용하는 방법을 기술한 안내서이다.
Tibero RDBMS JDBC 개발자 안내서	Tibero RDBMS에서 제공하는 JDBC 기능을 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero RDBMS tbESQL/C 안내서	C 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero RDBMS tbESQL/COBOL 안내서	COBOL 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero RDBMS tbPSM 안내서	저장 프로시저 모듈인 tbPSM의 개념과 문법, 구성요소를 소개하고, tbPSM 프로그램을 작성하는 데 필요한 제어 구조, 복합 타입, 서브프로그램, 패키지와 SQL 문장을 실행하고 에러를 처리하는 방법을 기술한 안내서이다.
Tibero RDBMS tbPSM 참조 안내서	저장 프로시저 모듈인 tbPSM의 패키지를 소개하고, 이러한 패키지에 포함된 각 프로시저와 함수의 프로토타입, 파라미터, 예제 등을 기술한 참조 안내서이다.
Tibero RDBMS tbAdmin 안내서	SQL/PSM 처리와 DBA를 위한 시스템 관리 기능을 제공하는 GUI 기반의 툴인 tbAdmin을 소개하고, 설치 및 사용 방법을 기술한 안내서이다.
Tibero RDBMS 유틸리티 안내서	데이터베이스와 관련된 작업을 수행하기 위해 필요한 유틸리티의 설치 및 환경설정, 사용 방법을 기술한 안내서이다.
Tibero RDBMS	Tibero RDBMS를 사용하는 도중에 발생할 수 있는 각종 에러의 원인과 해결 방법을 기술한 안내서이다.

안내서	설명
에러 참조 안내서	
Tibero RDBMS 참조 안내서	Tibero RDBMS의 동작과 사용에 필요한 초기화 파라미터와 데이터 사전, 정적 뷰, 동적 뷰를 기술한 참조 안내서이다.
Tibero RDBMS SQL 참조 안내서	데이터베이스 작업을 수행하거나 애플리케이션 프로그램을 작성할 때 필요한 SQL 문장을 기술한 참조 안내서이다.

참고 문헌

제품	안내서
Tibero MMDBMS	관리자 안내서

연락처

Korea

TIBERO Co., Ltd
272-6 Tmax Building 3th floor, Seohyeon-dong, Bundang-gu,
Seongnam-si, Gyeonggi-do, 463-824
South Korea
Tel: +82-31-779-7113
Fax: +82-31-779-7119
Email: `tibero@tibero.com`
Web (Korean): http://www.tibero.com
기술지원: http://technet.tmaxsoft.com

USA

TmaxSoft, Inc.
560 Sylvan Avenue Englewood Cliffs, NJ 07632
U.S.A
Tel: +1-201-567-8266
Fax: +1-201-567-7339
Email: `info@tmaxsoft.com`
Web (English): http://www.tmaxsoft.com

Japan

TmaxSoft Japan Co., Ltd.
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073
Japan
Tel: +81-3-5765-2550
Fax: +81-3-5765-2567
Email: `info@tmaxsoft.co.jp`
Web (Japanese): http://www.tmaxsoft.co.jp

China

TmaxSoft China Co., Ltd.
Beijing Silver Tower, RM 1508, 2# North Rd Dong San Huan,
Chaoyang District, Beijing, China, 100027
China
Tel: +86-10-6410-6145~8
Fax: +86-10-6410-6144
Email: info.cn@tmaxsoft.com
Web (Chinese): http://www.tmaxsoft.com.cn

제1장 Tibero RDBMS 소개

현재 기업의 비즈니스는 폭발적인 데이터의 증가와 다양한 환경 및 플랫폼의 등장으로 빠르게 확장되고 있다. 새로운 비즈니스 환경이 도래함에 따라 보다 더 효율적이고 유연한 데이터 서비스와 정보의 처리, 데이터 관리 기능이 필요하게 되었다.

Tibero RDBMS는 이러한 변화에 맞춰 기업 비즈니스 구현의 기반이 되는 데이터베이스 인프라 구성을 지원하며 고성능, 고가용성 및 확장성의 문제를 해결하는 엔터프라이즈 데이터베이스 관리 시스템이다.

기존 RDBMS의 단점을 보완하기 위해 Tibero RDBMS는 독자적인 Tibero Thread Architecture를 채택, 구현하였다. 한정된 서버 프로세스의 CPU 및 메모리 등의 시스템 리소스를 효율적으로 사용하면서 뛰어난 성능과 안정성 및 확장성을 보장하고 편리한 개발 환경과 관리 기능을 제공한다.

Tibero RDBMS는 초기 설계부터 대규모 사용자, 대용량 데이터, 강화된 안정성, 향상된 호환성 측면 등에서 타 DBMS와 차별화를 고려하여 개발되었다.

Tibero RDBMS는 이처럼 기업이 원하는 최적의 데이터베이스 환경을 제공하는 대표적인 RDBMS이다.

본 장에서는 **Tibero RDBMS의 주요 기능**과 **프로세스 구조**, **디렉터리 구조**를 간단히 소개한다.

1.1. 주요 기능

대용량의 데이터를 관리하고 안정적인 비즈니스의 연속성을 보장하는 데이터 관리 솔루션인 Tibero RDBMS는 RDBMS 환경에서 요구되는 주요 기능을 다음과 같이 갖추고 있다.

- **분산 데이터베이스 링크**(Distributed Database Link)

  데이터베이스 인스턴스별로 각각 서로 다른 데이터를 저장하는 기능이다. 이 기능을 통해 원격 데이터베이스에 저장된 데이터를 네트워크를 통해 읽기 및 쓰기를 수행할 수 있다.

  또한, 이 기능은 다양한 벤더의 RDBMS 제품을 연결하여 읽기 및 쓰기를 수행할 수 있다.

- **데이터 이중화**(Data Replication)

  현재 운영 중인 데이터베이스에서 변경된 모든 내용을 Standby DB로 복제하는 기능이다. 즉 네트워크를 통해서 변경 로그(Change log)만 전송하면 Standby DB에서 데이터에 적용하는 방식이다.

- **데이터베이스 클러스터**(Database Cluster)

  기업용 RDBMS의 최대 이슈인 고가용성과 고성능을 모두 해결하는 기능이다. Tibero RDBMS는 고가용성과 고성능을 보장하기 위해 **Tibero Active Cluster** 기술을 보유하고 있다.

이 기술로 인해 여러 개의 데이터베이스 인스턴스가 공유 디스크를 이용하여 동일한 데이터베이스를 공유할 수 있다. 이때 각 데이터베이스 인스턴스는 내부의 데이터베이스 캐시(Database cache) 사이의 일관성을 유지하는 기술이 매우 중요하다. 따라서 이러한 기술도 Tibero Active Cluster에 포함하여 제공하고 있다. 보다 자세한 내용은 "제10장 Tibero Active Cluster"를 참고한다.

- **병렬 쿼리 처리**(Parallel Query Processing)

  기업의 데이터 크기는 계속적으로 증가하고 있다. 대용량 데이터를 처리하기 위해 서버의 리소스를 최대한 활용할 수 있는 병렬 처리 기술이 필수적으로 요구되고 있다.

  Tibero RDBMS는 이러한 요구사항에 맞추어 온라인 트랜잭션 처리(OLTP, on-line transaction processing) 환경에 최적화된 기능을 제공할 뿐만 아니라 OLAP(Online Analytical Processing) 환경에 최적화된 SQL 병렬 처리 기능을 제공하고 있다. 이로 인해 쿼리는 빠른 응답 속도로 수행되며, 기업의 빠른 의사 결정을 돕는다.

- **쿼리 최적화기**(Optimizer)

  쿼리 최적화기는 스키마 객체의 통계 정도를 바탕으로 다양한 데이터 처리 경로들을 고려하여 어떤 실행 계획이 가장 효율적인지를 결정한다.

  쿼리 최적화기는 논리적으로 다음과 같은 단계를 통해 수행된다.

  1. 주어진 SQL 문을 처리하는 다양한 실행 계획들을 만들어 낸다.

  2. 데이터의 분산도에 대한 통계 정보와 테이블, 인덱스, 파티션 등의 특징을 고려하여 각각의 실행 계획의 비용을 계산한다. 여기서 비용이란 특정 실행 계획을 수행하는 데 필요한 상대 시간을 나타내고, 최적화기는 I/O, CPU, 메모리 등의 컴퓨터 자원을 고려하여 그 비용을 계산한다.

  3. 실행 계획들의 비용을 비교하여 가장 비용이 작은 계획을 선택한다.

  쿼리 최적화기의 주요 기능은 다음과 같다.

  – **최적화 목표**

    사용자는 최적화기의 최종 목표를 바꿀 수도 있는데, 예를 들어 다음의 두 가지 목표를 선택할 수 있다.

구분	설명
전체 처리 시간	ALL_ROWS 힌트를 사용하면 최적화기는 마지막 row까지 얻어내는 시간을 최대한 단축하도록 최적화한다.
최초 반응 시간	FIRST_ROWS 힌트를 사용하면 최적화기는 첫번째 row를 얻어내는 시간을 최대한 단축하도록 최적화한다.

  – **질의 변형**

질의의 형태를 바꿔서 더 좋은 실행 계획을 만들 수 있도록 한다. 질의 변형의 예에는 뷰 병합, 부질의 언네스트, 실체화 뷰의 사용 등이 있다.

- **데이터 접근 경로 결정**

  데이터를 데이터베이스로부터 꺼내오는 작업은 전체 테이블 스캔, 인덱스 스캔, rowid 스캔 등의 다양한 방법을 통해서 수행될 수 있다. 각 방법마다 필요한 데이터의 양이나 필터링의 형태 등에 따라 장단점이 있어서 질의에 따라 최적의 접근 방식이 다르다.

- **조인 처리 방식 결정**

  여러 테이블에서 데이터를 꺼내오게 되는 조인의 경우, 최적화기는 조인의 순서와 방법을 결정해야 한다. 여러 테이블간의 조인일 때 어떤 테이블을 먼저 조인할지에 대한 순서와, 각각의 조인에 있어서 중첩 루프 조인, 합병 조인, 해시 조인 등의 다양한 방법 중 어떤 것을 사용할 지가 실행 속도에 큰 영향을 미치게 된다.

- **비용 추정**

  각각의 실행 계획에 대해 비용을 추정한다. 비용 추정을 위해서 필요한 predicate의 선택도나 각 실행 단계에서 데이터의 row 수 등을 통계 정보를 사용해서 추정하고, 이를 바탕으로 각 단계에서의 비용을 추정한다.

1.2. 데이터베이스로서의 기본 기능

Tibero는 데이터베이스의 영속성과 일관성을 유지하기 위하여 SQL 문장의 묶음인 트랜잭션을 다음의 4가지 성질을 통해 보장한다.

- Atomicity

  All-or-nothing 즉 트랜잭션이 행한 모든 일이 적용되던가, 아니면 모두 적용 되지 않아야 함을 의미한다. Tibero에서는 이를 위하여 undo data를 사용한다.

- Consistency

  트랜잭션이 데이터베이스의 consistency를 깨뜨리는 일은 여러 방면에서 생겨날 수 있다. 간단한 예는 테이블과 인덱스간에 서로 다른 내용을 담고 있어서 consistency가 깨지는 것이다. 이를 막기 위해 Tibero에서는 트랜잭션이 적용한 일들 중 일부만 자신이나 남에게 적용되는 것을 막고 있다. 즉, 테이블만 수정했고 아직 인덱스를 수정하지 않은 상태라고 해도 다른 트랜잭션에서는 이를 예전 모습으로 돌려 보아서 테이블과 인덱스가 항상 consistency가 맞는 형태로 보이게 된다.

- Isolation

  트랜잭션은 혼자만 돌고 있는 것처럼 보이게 된다. 물론 다른 트랜잭션이 수정한 데이터에 접근시에는 이를 기다릴 수는 있지만 다른 트랜잭션이 수정 중이므로 접근할 수 없다고 에러가 나지는 않는다. 이를 위해 Tibero에서는 Multi version concurrency control 기법과 Row-level locking 기법을 사용한다.

데이터를 참조하는 경우에는 MVCC 기법을 이용하여 다른 트랜잭션과 무관하게 참조 가능하며, 데이터를 수정할 때도 row level의 fine-grained lock control을 통하여 최소한의 충돌만을 일으키고 만약 같은 데이터에 접근한다고 해도 단지 기다림으로써 이를 해결한다.

- Durability

  Tibero에서는 이를 위하여 Redo 로그와 write-ahead logging 기법을 사용한다. 트랜잭션이 commit될 때 해당 Redo 로그가 디스크에 기록되어 트랜잭션의 영속성을 보장해 준다. 또한 블럭이 디스크에 내려가기 전에 항상 Redo 로그가 먼저 내려가서 데이터베이스 전체가 일관성을 지니게 한다.

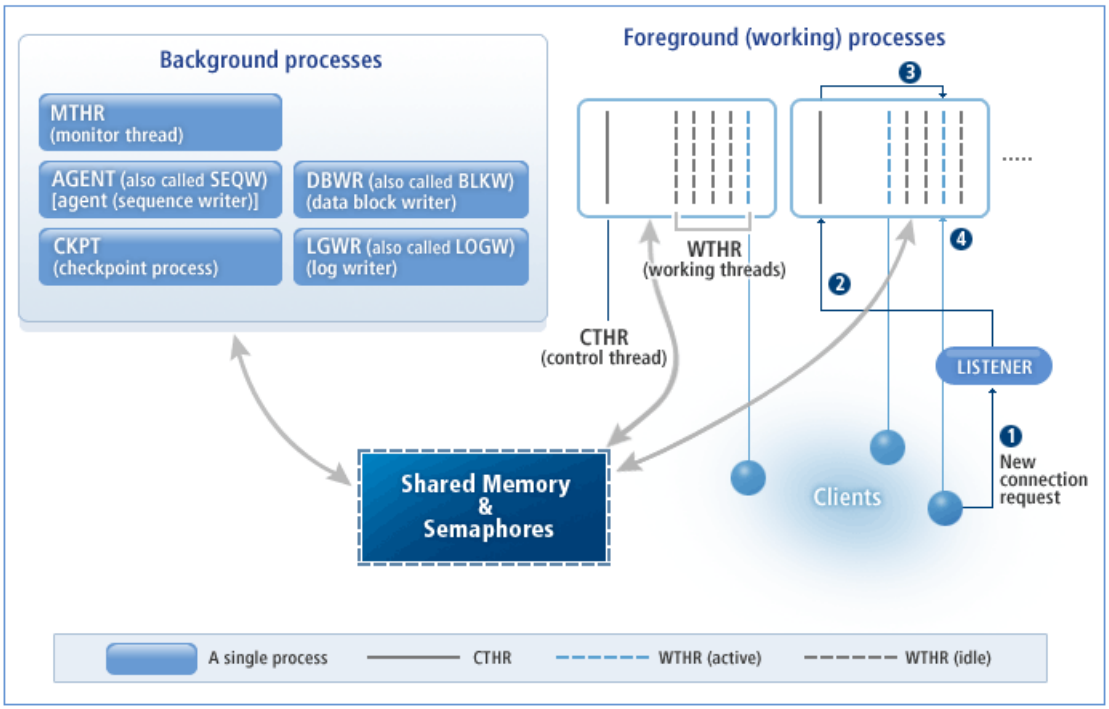
1.3. Row level locking

Tibero RDBMS는 fine-grained lock control을 보장해 주기 위하여 Row level locking을 사용한다. 즉, 데이터 최소 단위인 Row 단위의 locking을 통하여 최대한의 concurrency를 보장해 준다. 많은 Row들을 수정한다고 해도 테이블에 lock이 걸려서 동시에 발생하는 DML이 수행되지 못하는 상황은 발생하지 않는다. 이러한 기법을 통해 OLTP 환경에서 더욱 강력한 성능을 발휘하고 있다.

1.4. 프로세스 구조

Tibero RDBMS는 대규모 사용자 접속을 수용하는 다중 프로세스 및 다중 스레드 기반의 아키텍처를 갖추고 있다. 다음은 Tibero RDBMS의 프로세스 구조를 나타내는 그림이다.

[그림 1.1] Tibero RDBMS의 프로세스 구조



Tibero RDBMS의 프로세스는 크게 3가지로 구성된다.

- 리스너(Listener)
- 워킹 프로세스(Working Process 또는 Foreground Process)
- 배경 프로세스(Background Process)

1.4.1. 리스너

리스너는 클라이언트의 새로운 접속 요청을 받아 이를 유휴한 워킹 프로세스에 할당한다. 즉, 클라이언트와 워킹 프로세스간의 중계 역할을 담당하며, 이는 별도의 실행 파일인 **tblistener**를 사용하여 작업을 수행한다.

다음은 [그림 1.1]를 기준으로 클라이언트의 새로운 접속 요청이 이루어지는 순서이다.

1. 현재 유휴한 워킹 스레드가 있는 워킹 프로세스를 찾아서 클라이언트의 접속 요청(①)을 한다.
2. 이때 File descriptor와 함께 할당되므로, 클라이언트는 서버의 내부 동작과 상관없이 마치 처음부터 워킹 스레드에 접속한 것처럼 동작하게 된다.
3. 리스너의 요청을 받은 컨트롤 스레드 (CTHR: control thread)는 자기 자신에 속한 워킹 스레드의 상태를 검사(②)하여 현재 유휴한 워킹 스레드에 클라이언트의 접속을 할당(③)한다.
4. 할당된 워킹 스레드는 클라이언트와 인증 절차를 거친 후 세션을 시작(④)한다.

1.4.2. 워킹 프로세스

워킹 프로세스는 클라이언트와 실제로 통신을 하며 사용자의 요구 사항을 처리하는 프로세스이다.

이 프로세스의 개수는 `WTHR_PROC_CNT` 초기화 파라미터로 조절할 수 있으며, 일단 Tibero RDBMS가 기동된 뒤에는 변경할 수 없다. 따라서 시스템 환경을 고려하여 적절한 값을 설정해야 한다.

참고

초기화 파라미터에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

Tibero RDBMS는 효율적인 리소스의 활용을 위해 **스레드** (Thread) 기반으로 작업을 수행한다. Tibero RDBMS를 설치하면 기본적으로 하나의 워킹 프로세스 안에는 1개의 컨트롤 스레드와 10개의 워킹 스레드가 존재한다.

워킹 프로세스는 컨트롤 스레드와 워킹 스레드를 통해 다음과 같은 작업을 수행한다.

컨트롤 스레드

워킹 프로세스마다 하나씩 존재하며 다음과 같은 역할을 담당한다.

- Tibero RDBMS가 기동될 때 초기화 파라미터에 설정된 수만큼 워킹 스레드를 생성한다.
- 클라이언트의 새로운 접속 요청이 오면 현재 유휴한 워킹 스레드에 클라이언트의 접속을 할당한다.
- 시그널 처리를 담당한다.

워킹 스레드

워킹 스레드는 클라이언트와 1:1로 통신하며, 클라이언트가 보내는 메시지를 받아 처리하고 그 결과를 돌려준다. 주로 SQL 파싱, 최적화 수행 등 DBMS가 하는 작업 대부분이 워킹 프로세스에서 일어난다. 그리고 워킹 스레드는 하나의 클라이언트와 접속하므로 Tibero RDBMS에 동시 접속이 가능한 클라이언트 수는 `WTHR_PROC_CNT * _WTHR_PER_PROC`이다. Tibero RDBMS는 세션 멀티플렉싱 (Session multiplexing)을 지원하지 않으므로 하나의 클라이언트 접속은 곧 하나의 세션과 같다. 그러므로 최대 세션이 생성될 수 있는 개수는 `WTHR_PROC_CNT * _WTHR_PER_PROC`를 연산한 값과 같다.

워킹 스레드는 클라이언트와 접속이 끊긴다고 해도 없어지지 않으며, Tibero RDBMS가 기동될 때 생성된 이후부터 종료할 때까지 계속 존재하게 된다. 이러한 구조에서는 클라이언트와 접속을 빈번하게 발생하더라도 매번 스레드를 생성하거나 제거하지 않으므로 시스템의 성능을 높일 수 있다.

반면 실제 클라이언트의 수가 적더라도 초기화 파라미터에 설정된 개수만큼 스레드를 생성해야 하므로 운영체제의 리소스를 계속 소모하는 단점은 있으나, 운영체제 대부분이 유휴한 스레드 하나를 유지하는 데 드는 리소스는 매우 적으므로 시스템을 운영하는 데는 별 무리가 없다.

주의

많은 수의 워킹 스레드가 동시에 작업을 수행하려고 할 때 심한 경우 운영체제에 과도한 부하를 일으켜 시스템 성능이 크게 저하될 수 있다. 그러므로 대규모 시스템을 구축할 경우에는 Tibero RDBMS와 클라이언트의 애플리케이션 프로그램 사이에 미들웨어를 설치하여 3-Tier 구조로 시스템을 구축할 것을 권장한다.

1.4.3. 배경 프로세스

배경 프로세스(Background Process)는 클라이언트의 접속 요청을 직접 받지 않고, 워킹 스레드나 다른 배경 프로세스가 요청할 때 혹은 정해진 주기에 따라 동작하는 주로 시간이 오래 걸리는 디스크 작업을 담당하는 독립된 프로세스이다.

배경 프로세스에 속해 있는 프로세스는 다음과 같다.

- 감시 프로세스 (**MTHR**: Monitor thread)

감시 프로세스의 영문 약자를 보면 프로세스가 아닌 스레드로 나타나 있지만, 실제로는 하나의 독립된 프로세스이다. 리스너를 제외하고 Tibero RDBMS가 기동할 때 최초로 생성되며, Tibero RDBMS가 종료하면 맨 마지막에 프로세스를 끝마친다.

Tibero RDBMS가 기동할 때 다른 프로세스를 생성하거나 주기적으로 각 프로세스의 상태를 점검하는 역할을 담당한다. 또한, 교착 상태 (deadlock)도 검사한다.

- 시퀀스 프로세스 **(AGENT** 또는 **SEQW**: Sequence writer)

  시퀀스 캐시 (sequence cache)의 값을 디스크에 저장하고, 그 외에 시스템 유지를 위해 주기적으로 처리해야 하는 Tibero RDBMS 내부의 작업을 담당한다.

  시퀀스를 사용하는 방법에 대한 내용은 “4.6.1. 시퀀스의 생성, 변경, 제거”를 참고한다.

- 데이터 블록 쓰기 프로세스 (**DBWR** 또는 **BLKW**: Data block writer)

  사용자가 수정한 데이터 블록을 주기적으로 디스크에 기록한다. 기록된 데이터 블록은 주로 워킹 스레드가 직접 읽어온다.

- 체크포인트 프로세스 (**CKPT**: Checkpoint process)

  체크포인트는 주기적으로 혹은 클라이언트의 요청에 따라 메모리에 있는 변경된 모든 데이터 블록을 디스크에 기록하는 작업이다. Tibero RDBMS에 장애가 발생하면 이를 복구하기 위해 걸리는 시간이 한계 수치를 넘지 않도록 예방하는 효과가 있다. 이러한 체크포인트를 관리하는 프로세스가 체크포인트 프로세스이다.

  클라이언트가 직접 체크포인트를 요청하는 방법은 "Tibero RDBMS SQL 참조 안내서"의 ALTER SYSTEM CHECKPOINT 문을 참고한다.

- 로그 쓰기 프로세스(**LGWR** 또는 **LOGW**: Log writer)

  Redo 로그 파일을 디스크에 기록하는 프로세스이다. 로그 파일에는 데이터베이스의 데이터에 대한 모든 변경 사항을 저장하고 있으며 빠른 트랜잭션 처리와 복구를 위해 사용된다.

  로그 파일에 대한 자세한 내용은 “3.4. 로그 파일”을 참고한다.

1.5. 디렉터리 구조

Tibero RDBMS가 설치되면 다음과 같은 디렉터리가 생성된다.

```
$TB_HOME
    +- bin
    |
    +- client
    |   |
    |   +- bin
```

```
|   +- config
|   +- include
|   +- lib
|   +- ssl
|   |   |
|   |   +- man
|   |   |  +- man1
|   |   |  +- man3
|   |   |  +- man5
|   |   |  +- man7
|   |   +- misc
|   +- tbepa
|   |   |
|   |   +- java
|   |       |
|   |       +- config
|   |       +- lib
|   + win32
|   + win64
+- config
|
+- database
|   +- $TB_SID
|       |
|       +- psm
+- instance
|   |
|   +- $TB_SID
|       |
|       +- audit
|       +- log
|       |  +- dbmslog
|       |  +- lsnr
|       |  +- tracelog
|       +- path
+- lib
+- license
+- scripts
     +- pkg
```

위의 디렉터리 구조에서 $TB_SID라고 보이는 부분은 각각의 시스템 환경에 맞는 서버의 SID로 바꿔서 읽어야 한다.

Tibero RDBMS에서 사용하는 고유의 디렉터리는 다음과 같다.

bin

Tibero RDBMS의 실행 파일과 서버 관리를 위한 유틸리티가 위치한 디렉터리이다. 이 디렉터리에 속한 파일 중에서 tbsvr과 tblistener는 Tibero RDBMS를 구성하는 실행 파일이며, tbboot와 tbdown은 각각 Tibero RDBMS를 기동하고 종료하는 역할을 담당한다.

tbsvr과 tblistener 실행 파일은 반드시 tbboot 명령어를 이용하여 실행되어야 하며, 절대로 직접 실행해서는 안 된다.

client/bin

Tibero RDBMS의 클라이언트 실행 파일이 있는 디렉터리이다. 이 디렉터리에는 다음과 같은 유틸리티가 있다.

유틸리티	설명
tbSQL	기본적인 클라이언트 프로그램으로 사용자가 직접 SQL 질의를 하고 그 결과를 확인할 수 있는 유틸리티이다.
tbMigrator	다른 데이터베이스의 내용을 Tibero RDBMS의 데이터베이스로 옮기는 것을 지원하는 유틸리티이다.
tbExport	논리적 백업이나 데이터베이스간에 데이터 이동을 위해 데이터베이스의 내용을 외부 파일로 저장하는 유틸리티이다.
tbImport	외부 파일에 저장된 내용을 데이터베이스로 가져오는 유틸리티이다.
tbLoader	대량의 데이터를 데이터베이스로 한꺼번에 읽어 들이는 유틸리티이다.
tbpc	C 언어로 작성된 프로그램 안에서 내장 SQL(Embedded SQL)을 사용하는 프로그램을 개발할 때 이를 C 프로그램으로 변환하는 유틸리티이다. 이렇게 변환된 프로그램을 C 컴파일러를 통해 컴파일할 수 있도록 도와주는 역할도 담당한다.

유틸리티에 대한 내용은 "Tibero RDBMS 유틸리티 안내서"를 참고한다. 단, **tbpc** 유틸리티는 "Tibero RDBMS tbESQL/C 안내서"를 참고한다.

client/config

Tibero RDBMS의 클라이언트 프로그램을 실행하기 위한 설정 파일이 위치하는 디렉터리이다.

client/include

Tibero RDBMS의 클라이언트 프로그램을 작성할 때 필요한 헤더 파일이 위치하는 디렉터리이다.

client/lib

Tibero RDBMS의 클라이언트 프로그램을 작성할 때 필요한 라이브러리 파일이 위치하는 디렉터리이다. 이에 대한 자세한 내용은 "Tibero RDBMS 애플리케이션 개발자 안내서"와 "Tibero RDBMS tbESQL/C 안내서"를 참고한다.

client/ssl

서버 보안을 위한 인증서와 개인 키를 저장하는 디렉터리이다.

client/tbepa

External Procedure와 관련된 설정 파일과 로그 파일이 있는 디렉터리이다. 이에 대한 자세한 내용은 "Tibero RDBMS External Procedure 안내서"를 참고한다.

client/win32

32Bit Windows 용 ODBC/OLE DB 드라이버가 위치하는 디렉터리이다.

client/win64

64Bit Windows 용 ODBC/OLE DB 드라이버가 위치하는 디렉터리이다.

config

Tibero RDBMS의 환경설정 파일이 위치하는 디렉터리이다. 이 위치에 존재하는 $TB_SID.tip 파일이 Tibero RDBMS의 환경설정을 결정한다.

database/$TB_SID

Tibero RDBMS의 데이터베이스 정보를 별도로 설정하지 않는 한, 모든 데이터베이스 정보가 이 디렉터리와 그 하위 디렉터리에 저장된다.

이 디렉터리에는 데이터 자체에 대한 메타데이터(metadata)뿐만 아니라 다음과 같은 종류의 파일이 있다.

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.

database/$TB_SID/psm

tbPSM 프로그램을 컴파일드 모드(Compiled mode)로 컴파일하는 경우, 컴파일된 파일이 저장되는 디렉터리이다. 하지만, 현재 Tibero RDBMS에서는 인터프리터 모드만을 지원하고 있다.

이에 대한 자세한 내용은 "Tibero RDBMS 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

Tibero RDBMS의 트레이스(trace) 로그 파일과 DBMS 로그 파일이 저장되는 디렉터리이다.

파일	설명
트레이스 로그 파일	디버깅을 위한 파일이다. 서버가 하는 모든 일이 자세하게 기록되는 파일이며, 서버 성능이 저하되는 원인을 찾거나 Tibero RDBMS 자체의 버그를 해결하는 데 사용될 수 있다.
DBMS 로그 파일	트레이스 로그 파일에 기록되는 정보보다 좀 더 중요한 정보가 기록되는 파일이며, 서버 기동 및 종류, DDL 문장의 수행 등이 기록되는 파일이다.

트레이스 로그 파일과 DBMS 로그 파일은 데이터베이스를 사용할수록 계속 누적되어 저장된다. 또한, 전체 디렉터리의 최대 크기를 지정할 수 있으며, Tibero RDBMS는 그 지정된 크기를 넘어가지 않도록 오래된 파일을 삭제한다.

DBMS 로그 파일을 설정하는 초기화 파라미터는 다음과 같다.

초기화 파라미터	설명
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.

instance/$TB_SID/path

Tibero RDBMS의 프로세스간에 통신을 위한 소켓 파일이 있는 디렉터리이다. Tibero RDBMS가 운영 중일 때 이 위치에 존재하는 파일을 읽거나 수정해서는 안 된다.

lib

Tibero RDBMS 서버에서 Spatial과 관련된 함수를 사용하기 위한 라이브러리 파일이 있는 디렉터리이다.

license

Tibero RDBMS의 라이선스 파일(license.xml)이 있는 디렉터리이다. XML 형식이므로 일반 텍스트 편집기로도 라이선스의 내용을 확인할 수 있다.

scripts

Tibero RDBMS의 데이터베이스를 생성할 때 사용하는 각종 SQL 문장이 있는 디렉터리이다. 또한, Tibero RDBMS의 현재 상태를 보여주는 각종 뷰의 정의도 이 디렉터리에 있다.

scripts/pkg

Tibero RDBMS에서 사용하는 패키지의 생성문이 저장되는 디렉터리이다.

제2장 관리의 기본

본 장에서는 DBA가 Tibero RDBMS를 관리하기에 앞서 기본적으로 알아야 할 사항을 설명한다.

2.1. 사용자의 정의

Tibero RDBMS를 크게 사용하는 목적과 역할에 따라 사용자의 유형을 다음과 같이 정의한다.

- DBA
- SYS
- 시스템 관리자 (sysadmin)
- 애플리케이션 프로그램 개발자 (Application Developers)
- 데이터베이스 사용자 (Database Users)

2.1.1. DBA

DBMS는 적어도 1명의 DBA가 필요하다. 데이터베이스를 사용하는 사용자가 많으면 많을수록 이를 관리하는 개인이나 그룹이 필요하다.

DBA는 데이터베이스 환경을 유지하는 데 필요한 제반 활동을 감독하거나 직접 수행하는 개인 또는 그룹이다.

데이터베이스 내용의 정확성이나 통합성을 결정하고 데이터베이스의 내부 저장 구조와 접근 관리 대책을 결정하며, 데이터의 보안 정책을 수립하고 점검하는 등 데이터베이스의 성능을 감시하여 변화하는 요구에 대응하는 책임을 진다. 또한, 다양한 데이터베이스 제품에 관한 깊은 경험이 요구된다.

다음은 DBA가 관리해야 할 항목을 간략히 소개한 표이다.

항목	설명
DBMS	현재 사용하고 있는 DBMS의 특성을 파악한다. - 디스크 용량이 충분한가? - 물리적으로 독립된 디스크인 경우 시스템 부하가 제대로 분산이 되고 있는가? - 데이터 배치는 잘 되었는가?

항목	설명
	- CPU와 메모리 사용량, 클라이언트의 응답 시간은 어떠한가? 문제가 발생하면, H/W를 확장하거나 데이터베이스를 튜닝한다.
데이터베이스	- 자주 사용되는 스키마 객체는 무엇인가? - 자주 사용되는 SQL 문장은 무엇인가? - 자주 사용되는 SQL 문장이 효율적으로 실행되고 있는가?
유지보수	- Tibero RDBMS의 설치 및 패치를 수행한다. - 데이터베이스의 설계 및 분석, 구현을 한다. - 데이터베이스의 생성 및 권한을 설정한다. - 사용자의 권한을 설정한다.
정책 및 절차 확립	데이터베이스의 관리, 보안, 유지보수 등에 속하는 정책 및 절차를 확립한다.
보안	데이터 누출이나 유실을 막기 위해 보안 정책을 수립한다.
백업 및 복구	주기적으로 데이터를 백업하며, 문제 발생시 복구한다.

Tibero RDBMS는 DBA가 위와 같은 항목을 효율적이고 유연하게 관리할 수 있도록 유틸리티를 제공하고 있다.

참고

자세한 내용은 "Tibero RDBMS 유틸리티 안내서"를 참고한다.

2.1.2. SYS

Tibero RDBMS를 설치하고 나면, 데이터베이스 자체의 메타데이터를 관리하는 **SYS**라는 사용자가 생성된다. SYS 사용자에 DBA 역할이 부여되며, 이는 UNIX 계열(LINUX 포함)의 루트 사용자와 비슷한 역할을 담당한다.

Tibero RDBMS의 데이터 사전, 기반 테이블, 뷰 등은 모두 SYS 사용자의 스키마에 저장된다. 특히 기반 테이블, 뷰는 Tibero RDBMS가 동작하는 데 매우 중요한 역할을 한다. 따라서 SYS 사용자 외 다른 사용자가 이를 수정하거나 조작해서는 안 된다.

SYS 사용자의 접속 계정은 다음과 같다.

항목	설명
ID	ID는 **'SYS'**이다.

항목	설명
	SYS 사용자는 하나의 스키마를 가진다. 스키마의 이름은 사용자 이름인 SYS와 동일하다.
패스워드	패스워드는 Tibero RDBMS를 설치할 때 사용자가 입력한 값이다. 패스워드는 설치 후에 변경할 수 있다.

2.1.3. 시스템 관리자

일부 사이트는 1명 이상의 시스템 관리자가 있다. 시스템 관리자 (또는 sysadmin, 네트워크 관리자 포함)는 컴퓨터 시스템이나 네트워크를 운영하고 유지 보수하는 개인이나 그룹이다. 서버나 다른 컴퓨터에 운영체제를 설치하고, 유지 보수하며 서비스 정지 등의 문제에 대해 관리 책임이 있다.

또한, 약간의 프로그래밍 실력 그리고 시스템과 관련된 프로젝트에 대한 관리, 감시, 운영 기술 등을 가지고 있어야 하며 컴퓨터 문제에 대해 기술적 지원을 하는 역할도 수행해야 한다.

2.1.4. 애플리케이션 프로그램 개발자

애플리케이션 프로그램 개발자는 보통 넓은 영역의 컴퓨터 프로그래밍이나 전문적인 프로젝트 관리 분야에서 소프트웨어 개발 작업을 하는 개인이다. 애플리케이션 프로그램 개발자는 일반적으로 개별 프로그램 작업보다는 애플리케이션 프로그램의 수준에서 전반적인 프로젝트에 기여한다.

애플리케이션 프로그램 개발자는 데이터베이스 사용 측면에서 데이터베이스 애플리케이션 프로그램을 설계하고 구현하는 역할을 수행한다. 역할은 다음과 같다.

- 데이터베이스 애플리케이션 프로그램의 설계와 개발
- 애플리케이션 프로그램을 위한 데이터베이스 구조 설계 및 명세
- 애플리케이션 프로그램을 위한 저장 공간 요청
- DBA와 데이터베이스 정보를 공유
- 개발 중에 애플리케이션 프로그램 튜닝
- 개발 중에 애플리케이션 프로그램의 보안 정책 수립

2.1.5. 데이터베이스 사용자

데이터베이스 사용자는 Tibero RDBMS를 사용하는 DBA, 업무 분석가, 애플리케이션 프로그램 개발자, 사용자 모두를 뜻한다.

2.2. 설치 환경

Tibero RDBMS를 정상적으로 설치했다면, 시스템에 다음과 같은 환경 변수가 설정된다.

환경 변수	설명
$TB_HOME	Tibero RDBMS가 설치된 홈 디렉터리로 Tibero RDBMS 서버, 클라이언트 라이브러리, 다양한 부가 기능을 수행하는 파일이 설치된다.
$TB_SID	한 머신에서 Tibero RDBMS의 인스턴스를 여러 개로 운영할 때 필요한 서비스 ID이다.
$PATH	파일 시스템을 통해 특정 파일에 접근하기 위한 디렉터리 경로를 설정하는 환경 변수이다.

환경 변수를 제대로 설정하지 않으면 Tibero RDBMS를 사용할 수 없다. 따라서 환경 변수를 확인하는 절차가 필요하다.

참고

본 안내서에서는 UNIX 셸 명령어를 실행할 때 **GNU Bash**(http://www.gnu.org/software/bash/) 문법을 따른다. 사용하는 셸의 종류에 따라 문법이 다를 수 있으며, 자세한 내용은 현재 사용 중인 운영체제의 안내서를 참고한다.

UNIX 셸 프롬프트에서 환경 변수를 확인하는 방법은 다음과 같다.

- $TB_HOME

```
$ echo $TB_HOME
/home/tibero/tibero4
```

이 디렉터리 안에는 Tibero RDBMS 서버, 클라이언트 라이브러리, 부가 기능을 지원하는 파일이 있다.

- $TB_SID

```
$ echo $TB_SID
Tb4
```

서비스 ID는 데이터베이스의 이름과 동일하게 설정할 것을 권장한다.

- $PATH

```
$ echo $PATH
...:/home/tibero/tibero4/bin:/home/tibero/tibero4/client/bin:...
```

$PATH는 다음의 디렉터리를 포함하고 있어야 한다.

디렉터리	설명
$TB_HOME/bin	Tibero RDBMS의 실행 파일과 서버 관리를 위한 유틸리티가 위치한 디렉터리이다.
$TB_HOME/client/bin	Tibero RDBMS의 클라이언트 실행 파일이 있는 디렉터리이다.

2.3. tbSQL 유틸리티의 사용

tbSQL은 Tibero에서 제공하는 대화형 SQL 명령어 처리 유틸리티이다. SQL 질의, 데이터 정의어, 트랜잭션과 관련된 SQL 문장 등을 실행할 수 있다.

본 절에서는 **tbSQL** 유틸리티를 이용하여 데이터베이스에 접속하는 방법과 그 이후에 간단한 SQL 문장을 실행하는 방법을 설명한다.

tbSQL 유틸리티

tbSQL 유틸리티를 실행하는 방법은 다음과 같다.

[예 2.1] tbSQL 유틸리티의 실행

```
$ tbsql

tbSQL 4 SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

SQL>
```

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

참고

SQL 프롬프트가 나타나지 않고 데이터베이스 접속에 실패한 경우 “Appendix C. 문제 해결”을 참고하여 해결한다.

데이터베이스 접속

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

데이터베이스에 접속하는 방법은 다음과 같다.

[예 2.2] tbSQL 유틸리티를 이용한 데이터베이스 접속

```
$ tbsql SYS/tibero

tbSQL 4 SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Connected to Tibero.

SQL>
```

본 예제에서는 UNIX 셸 프롬프트에서 tbSQL 유틸리티의 실행과 함께 사용자명과 패스워드를 입력한다.

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

항목	설명
사용자명	스키마 객체의 이름과 마찬가지로 대소문자를 구분하지 않는다. 단, 큰따옴표(“”)에 사용자명을 입력하는 경우는 예외다.
패스워드	패스워드는 대소문자를 구분하므로 입력할 때 주의해야 한다.

tbSQL 유틸리티의 명령어

tbSQL 유틸리티에서 사용할 수 있는 명령어는 대소문자를 구분하지 않고 사용할 수 있으며, SQL 문장을 실행하거나 데이터베이스 관리에 필요한 명령어가 포함되어 있다.

tbSQL 유틸리티에서 사용할 수 있는 명령어는 총 35개로 다음과 같다.

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.
ACC[EPT]	사용자의 입력을 받아 바인드 변수의 속성을 설정하는 명령어이다.
C[HANGE]	SQL 버퍼의 현재 라인에서 패턴 문자를 찾아 주어진 문자로 변환하는 명령어이다.
CL[EAR]	설정된 옵션을 초기화하거나 지우는 명령어이다.

명령어	설명
COL[UMN]	컬럼의 출력 속성을 설정하는 명령어이다.
CONN[ECT]	특정 사용자 ID로 데이터베이스에 접속하는 명령어이다.
COUNT	지정된 테이블의 컬럼 개수를 세는 명령어이다.
DEFINE	바인드 변수를 정의하거나 출력하는 명령어이다.
DEL	SQL 버퍼에 저장된 라인을 지우는 명령어이다.
DESC[RIBE]	지정된 객체의 컬럼 정보를 출력하는 명령어이다.
DISC[ONNECT]	현재 데이터베이스로부터 접속을 해제하는 명령어이다.
ED[IT]	특정 파일 또는 SQL 버퍼의 내용을 외부 편집기를 이용하여 편집하는 명령어이다.
EXEC[UTE]	단일 tbPSM 문장을 수행하는 명령어이다.
EXIT	tbSQL 유틸리티를 종료하는 명령어이다. QUIT 명령어와 동일하다.
H[ELP]	도움말을 출력하는 명령어이다.
HIS[TORY]	실행한 명령어의 히스토리를 출력하는 명령어이다.
HO[ST]	운영체제 명령어를 실행하는 명령어이다. ! 명령어와 동일하다.
I[NPUT]	SQL 버퍼의 현재 라인 뒤에 새로운 라인을 추가하는 명령어이다.
L[IST]	SQL 버퍼 내의 특정 내용을 출력하는 명령어이다.
LOAD[FILE]	Tibero RDBMS의 테이블을 Oracle의 SQL*Loader 툴이 인식할 수 있는 형식으로 저장하는 명령어이다.
LS	현재 사용자가 생성한 데이터베이스 객체를 출력하는 명령어이다.
PAU[SE]	사용자가 <Enter> 키를 누를 때까지 실행을 멈추는 명령어이다.
PRI[NT]	사용자가 정의한 바인드 변수의 값을 출력하는 명령어이다.
PRO[MPT]	사용자가 정의한 SQL 문장이나 빈 라인을 그대로 화면에 출력하는 명령어이다.
Q[UIT]	tbSQL 유틸리티를 종료하는 명령어이다. EXIT 명령어와 동일하다.
R[UN]	현재 SQL 버퍼 내의 SQL 문장이나 tbPSM 프로그램을 실행하는 명령어이다. / 명령어와 동일하다.
SET	tbSQL 유틸리티의 시스템 변수를 설정하는 명령어이다.
SHO[W]	tbSQL 유틸리티의 시스템 변수를 출력하는 명령어이다.
SPO[OL]	화면에 출력되는 내용을 모두 외부 파일에 저장하는 과정을 시작하거나 종료하는 명령어이다.

명령어	설명
STA[RT]	스크립트 파일을 실행하는 명령어이다. @ 명령어와 동일하다.
[TB]DOWN	Tibero RDBMS를 종료하는 명령어이다.
UNDEF[INE]	하나 이상의 바인드 변수를 삭제하는 명령어이다.
VAR[IABLE]	사용자가 정의한 바인드 변수를 선언하는 명령어이다.

위 표에서 대괄호([ ])에 포함된 내용은 입력하지 않아도 명령어를 실행할 수 있다.

참고

tbSQL 유틸리티에 대한 자세한 내용은 "Tibero RDBMS 유틸리티 안내서"를 참고한다.

기본적으로 자신의 스키마에 어떤 객체들이 있는지 알아보기 위해서는 **LS** 명령어를 사용해야 한다.

다음은 SYS 사용자로 데이터베이스에 접속하여 **LS** 명령어를 실행하는 예이다.

[예 2.3] LS 명령어의 실행

```
SQL> LS

NAME                           OBJECT_TYPE
------------------------------ --------------------
SYS_CON100                     INDEX
SYS_CON400                     INDEX
SYS_CON700                     INDEX
_DD_CCOL_IDX1                  INDEX
        ......중간 생략......
UTL_RAW                        PACKAGE
DBMS_STATS                     PACKGE BODY
TB_HIDDEN2                     PACKGE BODY

SQL>
```

LS 명령어는 tbSQL 유틸리티를 사용할 때 사용자의 편의를 위해 제공되는 명령어이며, SQL 문장을 실행할 때 지원하는 명령어는 아니다. 오직 tbSQL 유틸리티에서만 사용할 수 있다. 즉, tbSQL 유틸리티가 아닌 JDBC, CLI 등을 이용하여 접속한 경우, 이 명령어를 사용할 수 없다.

[예 2.3]에서는 SYS 사용자의 스키마에 존재하는 모든 객체가 출력된다. SYS 스키마에는 Tibero RDBMS가 스스로를 관리하기 위해 내부적으로 사용되는 객체가 많다. 따라서 객체가 많이 출력된다.

다음은 LS 명령어를 실행하여 현재 시스템에 접속하고 있는 **사용자**를 조회하는 예이다.

[예 2.4] LS 명령어의 실행 - 사용자 조회

```
SQL> LS USER

USERNAME
------------------------------
SYS
```

다음은 LS 명령어를 실행하여 현재 데이터베이스에 존재하는 **테이블스페이스**를 조회하는 예이다.

[예 2.5] LS 명령어의 실행 - 테이블스페이스 조회

```
SQL> LS TABLESPACE

TABLESPACE_NAME
------------------------------
SYSTEM
UNDO
TEMP
USER
```

SQL 문장의 실행

다음은 **VT_SESSION** 이라는 뷰를 이용하여 TYPE 컬럼이 **'WTHR'**인 데이터를 조회하는 SQL 문장을 실행하는 예이다.

[예 2.6] SQL 문장의 실행 (1)

```
SQL> SELECT SESS_ID, STATUS, TYPE, WTHR_ID FROM VT_SESSION WHERE TYPE = 'WTHR';

 SESS_ID    STATUS                              TYPE          WTHR_ID
----------  ----------------------------------  ----------    ----------
       18                          RUNNING          WTHR             1

1 row selected.
```

VT_SESSION은 각각의 세션 ID를 나열하는 뷰이다.

다음은 **SQL 표준**에 대한 간략한 설명이다.

일반적으로 SQL 문장을 실행할 때 지켜야 할 규칙이 있다. 이 규칙은 ANSI(American National Standard Institute)와 ISO/IEC(International Standard Organization/International Electrotechnical Commission)에서 공동으로 제정한 관계형(relational) 또는 객체 관계형(object-relational) 데이터베이스 언어 즉 SQL 표준을 따른다.

SQL 표준은 1992년과 1999년에 각각 버전 2와 버전 3가 제정되었다. 1992년에 발표된 SQL 표준은 SQL2 또는 SQL-92라고 불리며, 관계형 데이터베이스를 위한 언어로 정의되었다.

1999년에 제정된 SQL 표준은 SQL3 또는 SQL-99라고 불리며, SQL2에 객체지향 개념을 추가하여 확장한 객체관계형 데이터베이스 언어이다.

SQL 표준에서 정의하고 있는 SQL 문장은 크게 다음과 같이 나뉜다.

- 데이터 조작어 (Data Manipulation Language, 이하 DML)
- 데이터 정의어 (Data Definition Language, 이하 DDL)
- 데이터 제어어 (Data Control Language, 이하 DCL)

참고

본 안내서에서는 DCL에 포함되는COMMIT, ROLLBACK 등의 SQL 명령어 일부를 트랜잭션 및 세션 언어로 재구성하였다. 따라서 전체 DCL에 대한 내용은 관련 문서를 참고하기 바란다.

DML에 포함되는 **SELECT** 문을 다음과 같이 실행한다.

[예 2.7] SQL 문장의 실행 (2)

```
SQL> select SESS_ID, STATUS, TYPE, WTHR_ID from vt_session where type = 'WTHR';

......[예 2.6]의 실행 결과와 동일......

SQL> select SESS_ID, STATUS, TYPE, WTHR_ID from VT_SESSION where type = 'WTHR';

......[예 2.6]의 실행 결과와 동일......

SQL> Select SESS_ID, STATUS, TYPE, WTHR_ID From Vt_session Where Type = 'WTHR';

......[예 2.6]의 실행 결과와 동일......
```

SQL 표준은 대소문자를 구분하지 않으므로 큰따옴표(“”) 또는 작은따옴표(‘’)로 묶은 부분을 제외하고는 대소문자를 혼용하여 사용할 수 있다. 위 예제는 모두 [예 2.6]를 실행한 결과와 동일한 결과가 나타나며, 그 의미 또한 같다.

그러나, 다음과 같은 SQL 문장을 실행하면 그 결과와 의미는 달라진다.

```
SQL> SELECT SESS_ID, STATUS, TYPE, WTHR_ID FROM VT_SESSION WHERE TYPE = 'wthr';

0 row selected.
```

2.4. 사용자 및 테이블의 생성

본 절에서는 데이터베이스를 사용하기 위해 사용자와 테이블을 생성하는 방법을 설명한다.

사용자의 생성

사용자를 생성하려면 **CREATE USER** 문을 사용해야 한다.

다음은 **ADMIN** 이라는 사용자를 생성하고 세션(CREATE SESSION)과 테이블을 생성(CREATE TABLE)할 수 있는 특권을 부여하는 예이다.

[예 2.8] 사용자의 생성

```
SQL> CREATE USER ADMIN IDENTIFIED BY 'password123';
      ..."ADMIN" 이라는 이름의 사용자를 생성하고 패스워드는 'password123'으로 한다.
User 'ADMIN' created.

SQL> GRANT CREATE SESSION TO ADMIN;
      ...ADMIN 사용자에게 세션을 시작할 수 있는 특권을 부여한다.
Granted.

SQL> GRANT CREATE TABLE TO ADMIN;
      ...ADMIN 사용자에게 테이블을 생성할 수 있는 특권을 부여한다.
Granted.

SQL> CONN ADMIN/PASSWORD123
      ...방금 만든 ADMIN 사용자로 데이터베이스에 접속한다.
TBR-17001: Login failed: invalid user name or password.

No longer connected to server.
      ...패스워드는 대소문자를 구분하므로 데이터베이스 접속에 실패한다.

SQL> CONN ADMIN/password123
      ...패스워드를 올바르게 입력한 후, 데이터베이스에 다시 접속한다.
Connected.

SQL> LS
      ...방금 생성된 사용자이므로 스키마 객체가 없다.
NAME                           OBJECT_TYPE
------------------------------ --------------------

SQL>
```

[예 2.8]의 과정을 모두 완료하면, Tibero RDBMS에 ADMIN 이라는 새로운 사용자가 추가된다.

테이블의 생성

테이블을 생성하려면 **CREATE TABLE** 문을 사용해야 한다.

다음은 **PRODUCT** 라는 테이블을 생성하고 4개의 로우 데이터를 삽입(INSERT)하는 예이다.

[예 2.9] CREATE TABLE 문을 이용한 테이블의 생성

```
SQL> CREATE TABLE "PRODUCT"
    (
      PROD_ID NUMBER(3) NOT NULL CONSTRAINT PROD_ID_PK PRIMARY KEY,
      PROD_NAME VARCHAR(50) NULL,
      PROD_COST NUMBER(10) NULL,
      PROD_PID NUMBER(3) NULL,
      PROD_DATE DATE NULL
    );
Table 'PRODUCT' created.

SQL> SELECT TABLE_NAME FROM USER_TABLES;
      ...USER_TABLES은 현재 데이터베이스에 접속한 사용자의 모든 테이블을 나열하는 정적 뷰이다.

TABLE_NAME
------------------------
PRODUCT

1 row selected.


SQL> DESC "PRODUCT"
      ...DESC는 PRODUCT 테이블에 어떤 컬럼이 있는지 출력하는 tbSQL 유틸리티의 명령어이다.

COLUMN_NAME              TYPE             CONSTRAINT
------------------------ ---------------- --------------------------
PROD_ID                  NUMBER(3)        PRIMARY KEY
                                          NOT NULL
PROD_NAME                VARCHAR(50)
PROD_COST                NUMBER(10)
PROD_PID                 NUMBER(3)
PROD_DATE                DATE

INDEX_NAME               TYPE    COLUMN_NAME
------------------------ ------- ------------------------------
PROD_ID_PK               NORMAL  PROD_ID


SQL> INSERT INTO "PRODUCT" VALUES(601,'TIBERO',7000,'',
to_date('2004-12-31 09:00:00', 'yyyy-mm-dd hh24:mi:ss'));
      ...SQL 표준에 따라 큰따옴표("")로 PRODUCT에 설정하면 스키마 객체나 사용자명을
```

```
          영문 대문자가 아닌 임의의 글자로 사용할 수 있다.

      ...SQL 표준에 따라 작은따옴표('')는 데이터베이스에 직접 삽입되는 문자열 데이터에
          사용한다.
1 row inserted.

SQL> INSERT INTO "PRODUCT" VALUES(602,'TIBERO2',8000,'601',
to_date('2005-06-21 09:00:00', 'yyyy-mm-dd hh24:mi:ss'));
1 row inserted.

SQL> INSERT INTO "PRODUCT" VALUES(603,'TIBERO3',9000,'601',
to_date('2007-01-01 09:00:00', 'yyyy-mm-dd hh24:mi:ss'));
1 row inserted.

SQL> INSERT INTO "PRODUCT" VALUES(604,'TIBERO4',10000,'601',
to_date('2009-04-30 09:00:00', 'yyyy-mm-dd hh24:mi:ss'));
1 row inserted.

SQL> SELECT * FROM "PRODUCT";
      ...PRODUCT 테이블의 모든 데이터를 출력하는 SQL 문장이다.

   PROD_ID PROD_NAME             PROD_COST    PROD_PID  PROD_DATE
---------- -------------------  ----------  ----------  ----------
       601 TIBERO                     7000              2004/12/31
       602 TIBERO2                    8000         601  2005/06/21
       603 TIBERO3                    9000         601  2007/01/01
       604 TIBERO4                   10000         601  2009/04/30

4 rows selected.
```

Tibero RDBMS는 **USER_TABLES**를 비롯한 여러 정적 뷰를 제공한다. 이 뷰를 통해 현재 데이터베이스에 접속한 사용자가 접근할 수 있는 스키마 객체의 다양한 정보를 볼 수 있다.

참고

정적 뷰에 대한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

테이블 생성과 데이터 삽입을 완료하였으면, 이 콘솔 창을 **그대로 놔두고** 또 다른 콘솔 창을 실행한다.

본 예제에서는 tbSQL 유틸리티를 이용하여 [예 2.8]에서 생성한 ADMIN으로 Tibero RDBMS에 다음과 같이 접속한다.

```
$ tbsql ADMIN/password123

tbSQL 4 SP1
```

```
Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Connected to Tibero.

SQL> SELECT * FROM "PRODUCT";

0 row selected.
```

위 예를 보면, [예 2.9]에서처럼 4개의 로우 데이터가 출력되지 않음을 알 수 있다. 그 이유는 Tibero RDBMS는 트랜잭션을 지원하기 때문에 한 세션에서 입력한 데이터라도 커밋을 하기 전까지는 다른 세션에서 보이지 않는 현상이다. 따라서 4개의 로우 데이터를 확인하려면 이전 콘솔 창([예 2.9] 화면)으로 이동하여 트랜잭션의 커밋 명령을 실행해야 한다.

커밋 명령을 실행하는 방법은 다음과 같다.

```
SQL> COMMIT;
Commit succeeded.
```

커밋이 완료되면 두 번째 콘솔 창에서 SELECT 문을 실행하여 4개의 로우 데이터가 출력되는지 확인한다.

사용 예제

다음은 가상의 시나리오를 설정하여 SQL 문장을 실행하는 예이다.

시나리오는 다음과 같다.

- 8,500원 미만의 모든 제품의 가격(PROD_COST 컬럼)을 10% 인상했는데, 다시 이전 상태로 복구해야 한다.
- TIBERO2 제품은 더 이상 판매하지 않는다.

시나리오를 기준으로 SQL 문장을 실행하는 과정은 다음과 같다.

```
SQL> UPDATE "PRODUCT" SET PROD_COST = PROD_COST * 1.1
    WHERE PROD_COST < 8500;
2 rows updated.

SQL> SELECT PROD_NAME, PROD_COST FROM "PRODUCT";

PROD_NAME                                          PROD_COST
-------------------------------------------------- ----------
TIBERO                                                   7700
TIBERO2                                                  8800
TIBERO3                                                  9000
TIBERO4                                                 10000

4 rows selected.
```

```
      ...8,500원 미만의 모든 제품의 가격(PROD_COST 컬럼)을 10% 인상한 SQL 문장이다.

SQL> ROLLBACK;
      ...다시 이전 상태로 복구한다.

Rollback succeeded.


SQL> DELETE FROM "PRODUCT" WHERE PROD_NAME = 'TIBERO2';
      ...TIBERO2 제품은 더는 판매하지 않는다.따라서 PRODUCT 테이블에서 이 제품을 삭제한다.

1 row deleted.

SQL> SELECT PROD_NAME, PROD_COST FROM "PRODUCT";
      ...PRODUCT 테이블을 조회한다. TIBERO2 제품은 삭제되었고,
         10% 인상됐던 TIBERO 제품(8,500원 미만)은 이전 상태의 가격으로 롤백 되었다.

PROD_NAME                                          PROD_COST
-------------------------------------------------- ----------
TIBERO                                               7000
TIBERO3                                              9000
TIBERO4                                             10000

3 rows selected.

SQL> quit
      ...quit 명령어는 현재 진행중인 트랜잭션을 먼저 커밋하고 데이터베이스 접속을 종료한다.
         따라서 TIBERO2 제품은 PRODUCT 테이블에서 완전히 제거되었다.
Disconnected.

$
```

SQL 문장을 실행하는 데 있어 사용자에게 부여된 특권은 매우 중요하다. DBA 역할을 부여 받은 사용자라면, 데이터베이스를 관리할 때 편리하게 SQL 문장을 실행할 수 있다. 부여되지 않은 특권 때문에 매번 SYS 사용자나 DBA 역할을 가진 다른 사용자로 접속하여 특권을 부여해줘야 하기 때문이다. 이를 해결하기 위해 여러 특권을 모아 하나의 역할로 생성하는 방법을 사용할 수 있다.

참고

특권과 역할에 대한 자세한 내용은 “5.2. 특권”과 “5.3. 역할”을 참고한다.

다음은 SYS 사용자로 데이터베이스에 접속한 후, [예 2.8]에서 생성한 ADMIN 사용자에게 DBA 역할을 부여하는 예이다.

```
$ tbsql SYS/tibero

tbSQL 4 SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Connected to Tibero.

SQL> GRANT DBA TO ADMIN;
Granted.
```

ADMIN 사용자를 더 이상 사용하지 않을 경우에는 다음과 같은 SQL 명령을 실행한다.

```
SQL> DROP USER ADMIN;
TBR-7139: cascade is required to remove this user from the system

SQL> DROP USER ADMIN CASCADE;
User 'ADMIN' dropped.
```

위 예제에서 보듯이, ADMIN 사용자의 스키마에 생성된 모든 객체를 완전히 삭제하려면 **CASCADE** 옵션을 반드시 사용해야 한다. 그렇지 않으면 TBR-7139 에러가 발생한다. ADMIN 사용자는 더 이상 데이터베이스에 접속할 수 없게 된다.

2.5. 기동과 종료

본 절에서는 Tibero RDBMS를 기동하고 종료할 때 사용하는 명령어와 이를 사용하는 방법을 설명한다.

2.5.1. tbboot

tboot는 Tibero RDBMS가 설치된 머신에서 실행해야 한다. 또한, “2.2. 설치 환경” 에서 설명했듯이 tbboot를 실행하기 전에 반드시 환경 변수가 제대로 설정되어 있어야 한다. 이 명령어와 관련된 환경 변수는 $TB_HOME과 $TB_SID이다.

또한, tbboot는 실행 파일을 실행할 수 있는 권한이 있는 사용자라면 어느 누구든 Tibero RDBMS를 기동할 수 있다. 따라서 이는 보안 문제가 발생할 수 있어, 정책 상 Tibero RDBMS를 설치한 사용자만 Tibero RDBMS의 실행 파일에 접근할 수 있도록 권한을 설정할 것을 권장한다.

파일의 권한(permission)을 설정하는 방법은 다음과 같다.

```
$ cd $TB_HOME/bin
$ chmod 700 tbsvr tblistener tbboot tbdown tbctl
$ ls -alF
```

```
total 56
drwxr-xr-x  4 tibero tibero 4096 Dec 28 18:12 ./
drwxr-xr-x 13 tibero tibero 4096 Dec 20 11:59 ../

        ..... 중간 생략.....

-rwx------  1 ... tbboot*
-rwx------  1 ... tbctl*
-rwx------  1 ... tbdown*
lrwxrwxrwx    1 ... tblistener -> .build/dflt/tblistener*
lrwxrwxrwx    1 ... tbsvr -> .build/dflt/tbsvr*

$ ls -alF .build/dflt        ... 디렉터리 이름은 시스템 환경에 따라 다를 수 있다. ...
total 98332
drwxr-xr-x 2 tibero tibero     4096 Dec 28 18:12 ./
drwxr-xr-x 3 tibero tibero     4096 Oct 30 13:52 ../
-rwx------ 1 tibero tibero   712592 Dec 28 18:12 tblistener*
-rwx------ 1 tibero tibero 30613251 Dec 28 18:12 tbsvr*
```

tboot의 사용법은 다음과 같다.

```
tbboot
tbboot -v
tbboot -h
tbboot [-t] [ normal | mount | nomount | resetlogs |
              NORMAL | MOUNT | NOMOUNT | RESETLOGS ]
```

옵션	설명
	옵션이 없는 경우 Tibero RDBMS를 부트 모드 중 NORMAL로 기동하는 옵션이다.
-h	tbboot를 사용하기 위한 간단한 도움말을 보여주는 옵션이다.
-v	Tibero RDBMS의 버전 정보를 보여주는 옵션이다.
-t	Tibero RDBMS 서버를 기동할 수 있는 옵션이다. 이 옵션은 생략이 가능하다.

Tibero RDBMS에서는 tbboot에 부트 모드(bootmode)를 제공한다.

다음은 각 부트 모드의 사용 예이다.

NORMAL

정상적으로 데이터베이스의 모든 기능을 사용할 수 있는 모드이다.

사용 예는 다음과 같다.

```
$ tbboot NORMAL
listener port = 8629
```

```
Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NORMAL mode).
```

참고

데이터베이스를 비정상적으로 종료한 경우 Tibero RDBMS가 기동할 때 자동으로 파손 복구(crash recovery)를 실행한다. 자세한 내용은 "6.5.2. 파손 복구"를 참고한다.

NOMOUNT

Tibero RDBMS의 프로세스만 기동 시키는 모드이다.

일반적으로는 이 모드를 사용하는 경우는 거의 없고 Tibero RDBMS가 기동한 다음에 CREATE DATABASE 문을 이용하여 데이터베이스를 생성하는 것밖에 없다.

사용 예는 다음과 같다.

```
$ tbboot NOMOUNT
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started suspended at NOMOUNT mode.
```

MOUNT

미디어 복구를 위해 사용하는 모드이다.

사용 예는 다음과 같다.

```
$ tbboot MOUNT
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started suspended at MOUNT mode.
```

RESETLOGS

Tibero RDBMS 서버를 기동하는 과정에서 로그 파일을 초기화하며, 미디어 복구 이후에 사용하는 모드이다.

사용 예는 다음과 같다.

```
$ tbboot RESETLOGS
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started suspended at NORMAL mode.
```

참고

RESETLOGS 부트 모드는 직전에 데이터베이스가 비정상적으로 종료되었을 경우에는 사용할 수 없으며, 일단 Tibero RDBMS가 기동되면 NORMAL 모드와 동일하게 동작한다. 자세한 내용은 "6.5.3. 미디어 복구"를 참고한다.

2.5.2. tbdown

tbdown은 현재 동작 중인 Tibero RDBMS를 종료하는 역할을 수행한다.

tbdown의 사용법은 다음과 같다.

```
tbdown
tbdown -h
tbdown [-t] [ normal | post_tx | immediate | abort | switchover | abnormal |
               NORMAL | POST_TX | IMMEDIATE | ABORT | SWITCHOVER | ABNORMAL ]
tbdown clean
```

옵션	설명
	옵션이 없는 경우 Tibero RDBMS를 정상 모드로 종료하는 옵션이다.
-h	tbdown을 사용하기 위한 간단한 도움말을 보여주는 옵션이다.
-t	Tibero RDBMS 서버를 종료할 수 있는 옵션이다. 이 옵션은 생략이 가능하다.
clean	Tibero RDBMS 서버가 비정상 종료된 상태에서 운영 중에 사용하였던 공유 메모리나 세마포어 자원들을 해제하는 옵션이다. Tibero RDBMS 서버가 운영 중일 때는 사용할 수 없는 옵션이다.

참고

만약 Tibero RDBMS 서버가 kill과 같은 시스템 내부 명령어에 의해서 비정상적으로 종료된 경우, 운영 중에 사용하였던 공유 메모리나 세마포어 자원들이 해제가 안 될 수 있다. 이런 경우에 재부팅을 시도하는 경우 실패를 하게 되고, 관리자는 에러 메시지를 통해서 서버가 비정상 종료 되었다는 것을 인지하게 된다. 이와 같은 서버의 비정상 종료 후, 재부팅을 하기 위해서는 먼저 반드시 tbdown clean으로 기존 자원을 해제시켜야 한다.

비정상적으로 종료되더라도 초기화 파라미터 BOOT_WITH_AUTO_TBDOWN_CLEAN를 Y로 설정하면 자동으로 이전 운영 중에 사용하였던 자원을 해제시켜 부팅을 시킬 수는 있다. 하지만 관리자가 서버의 비정상 종료 상황을 제대로 인지하지 못 하고 서버 운영을 할 수 있으며, 기존 자원이나 프로세스가 제대로 정리가 안 되는 예외적이 상황이 발생하여 충돌이 날 수 있으므로 BOOT_WITH_AUTO_TBDOWN_CLEAN 옵션을 켜는 것을 권장하지 않는다.

Tibero RDBMS에서는 tbdown에 다운 모드(downmode)를 제공한다.

다운 모드	설명
NORMAL	일반적인 종료 모드이다.
POST_TX	모든 트랜잭션이 끝날 때까지 기다리고 나서 Tibero RDBMS를 종료하는 모드이다.
IMMEDIATE	현재 수행 중인 모든 작업을 강제로 중단시키며, 진행 중인 모든 트랜잭션을 롤백하고 Tibero RDBMS를 종료하는 모드이다.
ABORT	Tibero RDBMS의 프로세스를 강제로 종료하는 모드이다.
SWITCHOVER	Standby DB와 Primary DB를 동기화시킨 후, Primary DB를 NORMAL 모드처럼 종료하는 모드이다.
ABNORMAL	Tibero RDBMS 서버에 접속하지 않고 서버 프로세스를 무조건 강제로 종료시키는 모드이다.

참고

1. 만약 현재 Tibero RDBMS 서버에 접속하고 있는 세션이 존재하는 경우에 다운 모드 옵션이 없는 tbdown 명령을 하면, NORMAL 모드로 세션이 끝날 때까지 기다렸다가 종료할건지, IMMEDIATE 모드로 바로 종료할건지, 아니면 서버 종료를 취소할 건지를 선택해야 한다.

2. 이 장에서 설명하는 tbdown 명령과 다운 모드는 SYS 사용자로 접속한 tbSQL의 SQL 프폼프트에서도 가능하다. 이를 통하여 Tibero RDBMS 서버가 설치되지 않은 원격에서 tbSQL를 이용하여 서버를 종료시킬 수 있다. 단, ABNORMAL 모드의 tbdown 명령은 tbSQL에서 사용할 수 없다.

3. Windows에서 서비스 관리자를 통해서 Tibero RDBMS 서비스를 중지시킬 경우, 기본적으로 immediate 모드로 Tibero RDBMS 서버가 종료된다. 하지만 사용자가 원한다면 초기화 파라미터 SERVICE_CONTROL_STOP_DOWN를 이용하여 종료 모드를 설정할 수 있다.

다음은 각 다운 모드의 사용 예이다.

NORMAL

일반적인 종료 모드이다.

Tibero RDBMS에 SYS 사용자로 접속한 다음 다른 모든 세션의 접속이 끊어질 때까지 기다린 후, 그 다음 서버를 종료시킨다. 일단 tbdown을 실행하고 나면 어떤 사용자도 더 이상 데이터베이스에 접속할 수 없게 된다. 하지만, tbdown이 실행되기 전에 이미 데이터베이스에 접속한 사용자는 스스로 접속을 끊을 때까지 제한 없이 데이터베이스를 계속 사용할 수 있다.

사용 예는 다음과 같다.

```
$ tbdown

Tibero instance terminated (NORMAL mode).
```

POST_TX

모든 트랜잭션이 끝날 때까지 기다리고 나서 Tibero RDBMS를 종료하는 모드이다.

POST_TX는 Tibero RDBMS에 SYS 사용자로 접속한 다음 모든 **트랜잭션**이 끝날 때까지 기다린다. 그 다음 Tibero RDBMS를 종료시킨다. tbdown 실행이 시작되면 더는 데이터베이스에 접속할 수 없고, 이미 열려 있던 세션에서도 새로운 트랜잭션을 시작할 수 없게 된다. 다만, 현재 수행 중인 트랜잭션은 커밋 혹은 롤백할 때까지 제한 없이 수행할 수 있으며, 커밋이나 롤백을 하는 순간 자동으로 데이터베이스 접속을 종료하게 된다.

또한, tbdown 실행이 시작되면 데이터베이스에 접속한 클라이언트에게 **서버 종료**를 알리는 메시지를 보내지 않는다. tbSQL 유틸리티 등에서는 클라이언트가 서버 종료를 즉시 알지 못하고 그 다음 명령을 실행할 때 비로소 Tibero RDBMS가 종료되었음을 알게 된다.

예를 들면 다음과 같다.

```
$ tbsql admin/password123

tbSQL 4 SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Connected to Tibero.

SQL> CREATE TABLE T1 (COL1 NUMBER);
Table 'T1' created.

SQL> INSERT INTO T1 VALUES(10);
1 row inserted.

SQL> SELECT * FROM T1;
      COL1
```

```
----------
        10

1 row selected.

        ...이 시점에서 tbdown POST_TX 명령을 실행한다.
        ...tbdown 명령은 트랜잭션이 끝나기를 기다린다.

SQL> COMMIT;
Commit succeeded.
        ...이 시점에서 실제로 서버가 종료되고 tbdown 실행이 끝난다.

SQL> SELECT * FROM T1;
TBR-2069: I/O read error
        ...서버가 종료되었으므로 tbSQL 유틸리티의 프롬프트에서는 TBR-2069 에러가 발생된다.
```

IMMEDIATE

현재 수행 중인 모든 작업을 강제로 중단시키며, 진행 중인 모든 트랜잭션을 롤백하고 Tibero RDBMS를 종료하는 모드이다.

IMMEDIATE는 Tibero RDBMS에 SYS 사용자로 접속한 다음 현재 수행 중인 모든 작업을 강제로 종료하고 진행 중이던 모든 트랜잭션을 롤백한 후, Tibero RDBMS를 종료시킨다. 클라이언트에서 Tibero RDBMS 종료를 알지 못하는 것은 POST_TX 모드와 같다.

트랜잭션이 오래 걸리는 작업 중에 있었다면, 이를 모두 롤백하기 위해서 다소 시간이 걸릴 수 있다.

사용 예는 다음과 같다.

```
$ tbdown immediate

Tibero instance terminated (IMMEDIATE mode).
```

ABORT

Tibero RDBMS의 프로세스를 강제로 종료하는 모드이다.

ABORT는 Tibero RDBMS에 SYS 사용자로 접속한 다음 Tibero RDBMS의 MTHR 프로세스가 모든 프로세스를 OS의 강제 종료 시그널을 전달하여 강제로 종료시키는 모드이다. 따라서 이 모드는 비상 시에 사용하며. 다음 번에 Tibero RDBMS를 기동할 때 파손 복구 과정이 필요하다.

사용 예는 다음과 같다.

```
$ tbdown abort

Tibero RDBMS instance terminated (ABORT mode).
```

ABORT는 Tibero RDBMS가 강제로 종료시키므로 사용하던 시스템 리소스를 해제할 기회가 없다.

따라서 서버가 종료된 다음에도 공유 메모리(Shared Memory), 세마포어(Semaphore) 등의 시스템 리소스가 남아있을 수 있으며, 로그 파일이나 데이터 파일도 마찬가지이다.

또한, 다음 번에 Tibero RDBMS를 기동할 때 파손 복구에 많은 시간이 걸릴 수도 있다.

ABORT는 다음과 같은 경우에만 제한적으로 사용할 것을 권장한다.

- Tibero RDBMS의 내부 에러로 인한 정상적인 종료가 불가능한 경우
- H/W에 문제가 발생하여 Tibero RDBMS를 즉시 종료해야 하는 경우
- 해킹 등의 비상 상태가 발생하여 Tibero RDBMS를 즉시 종료해야 하는 경우

SWITCHOVER

SWITCHOVER는 Standby DB와 Primary DB를 동기화시킨 후, Primary DB를 NORMAL 모드처럼 종료하는 모드이다. 이와 관련된 자세한 내용은 “제8장 Tibero Standby Cluster”의 “8.4.1. Switchover”를 참고한다.

ABNORMAL

Tibero RDBMS 서버에 접속하지 않고 서버 프로세스를 무조건 강제로 종료시키는 모드이다.

ABNORMAL은 Tibero RDBMS 서버에 접속하지 않은 채 현재 Tibero RDBMS 서버 상태와 상관없이 OS의 강제 종료 시그널을 사용하여 무조건 서버 프로세스를 강제로 종료시키는 모드이다. 따라서 이 모드는 비상 시에 사용하며, 다음 번에 Tibero RDBMS를 기동할 때 파손 복구 과정이 필요하다.

사용 예는 다음과 같다.

```
$ tbdown abnormal

Tibero instance terminated (ABNORMAL mode).
```

ABNORMAL는 Tibero RDBMS가 강제로 종료시키므로 사용하던 시스템 리소스를 해제할 못 할 수 있다.

따라서 서버가 종료된 다음에도 공유 메모리(Shared Memory), 세마포어(Semaphore) 등의 시스템 리소스가 남아있을 수 있으며, 로그 파일이나 데이터 파일도 마찬가지이다.

또한, 다음 번에 Tibero RDBMS를 기동할 때 파손 복구에 많은 시간이 걸릴 수도 있다.

ABNORMAL은 다음과 같은 경우에만 제한적으로 사용할 것을 권장한다.

- Tibero RDBMS의 내부 에러로 인한 정상적인 종료가 불가능한 경우

- ABNORMAL외 다른 종료 모드로 종료 명령을 내린 후 종료가 지연되고 있을 때, 강제로 즉시 종료해야 하는 경우
- H/W나 OS등의 외부적인 요인으로 인하여 문제가 발생하여 ABNORMAL외 다른 종료 모드의 실행이 실패한 경우
- H/W에 문제가 발생하여 Tibero RDBMS를 즉시 종료해야 하는 경우
- 해킹 등의 비상 상태가 발생하여 Tibero RDBMS를 즉시 종료해야 하는 경우

제3장 파일과 데이터의 관리

본 장에서는 Tibero RDBMS의 파일과 데이터를 관리하는 방법을 설명한다.

3.1. 데이터의 저장 구조

Tibero RDBMS의 데이터를 저장하는 구조는 다음과 같이 두 가지 영역으로 나뉜다.

- 논리적 저장 영역

  데이터베이스의 스키마 객체를 저장하는 영역이다.

  논리적 저장 영역은 다음과 같은 포함 관계가 있다.

  ```
  데이터베이스 > 테이블스페이스 > 세그먼트 > 익스텐트
  ```

- 물리적 저장 영역

  운영체제와 관련된 파일을 저장하는 영역이다.

  물리적 저장 영역은 다음과 같은 포함 관계가 있다.

  ```
  데이터 파일 > 운영체제의 데이터 블록
  ```

3.2. 데이터의 저장 구조

Tibero RDBMS의 데이터를 저장하는 구조는 다음과 같이 두 가지 영역으로 나뉜다.

- 논리적 저장 영역

  데이터베이스의 스키마 객체를 저장하는 영역이다.

  논리적 저장 영역은 다음과 같은 포함 관계가 있다.

  ```
  데이터베이스 > 테이블스페이스 > 세그먼트 > 익스텐트
  ```

- 물리적 저장 영역

  운영체제와 관련된 파일을 저장하는 영역이다.

  물리적 저장 영역은 다음과 같은 포함 관계가 있다.

  ```
  데이터 파일 > 운영체제의 데이터 블록
  ```

3.3. 테이블스페이스

테이블스페이스는 논리적 저장 영역과 물리적 저장 영역에 공통적으로 포함된다. 논리적 저장 영역에는 Tibero RDBMS의 모든 데이터가 저장되며, 물리적 저장 영역에는 데이터 파일이 하나 이상 저장된다.

테이블스페이스는 논리적 저장 영역과 물리적 저장 영역을 연관시키기 위한 단위이다.

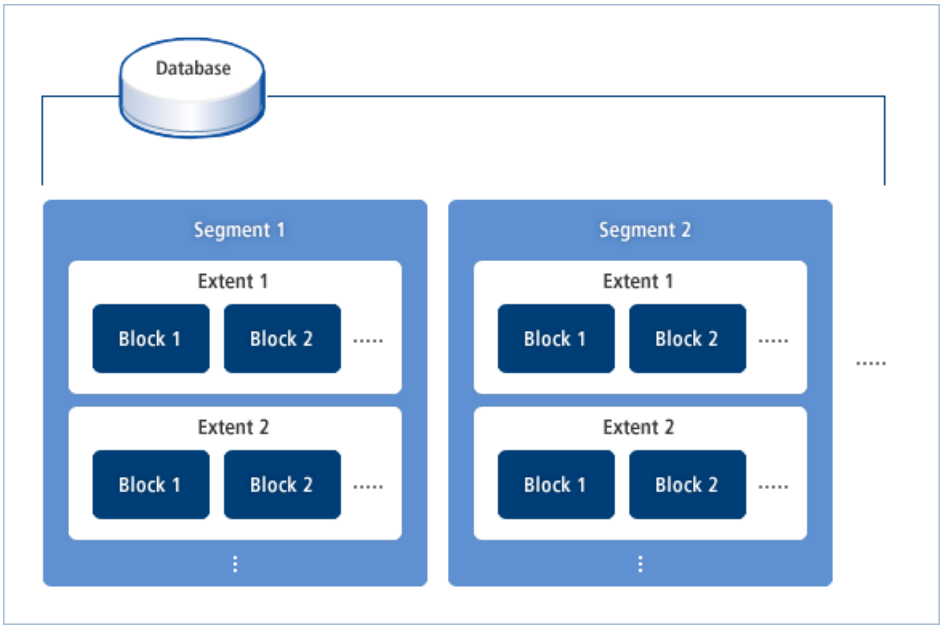
3.3.1. 테이블스페이스의 구성

테이블스페이스는 크게 두 가지 구성으로 Tibero RDBMS의 데이터를 저장한다.

테이블스페이스의 논리적 구성

다음은 테이블스페이스의 논리적 구성을 나타내는 그림이다.

[그림 3.1] 테이블스페이스의 논리적 구성



테이블스페이스는 [그림 3.1]과 같이 세그먼트(Segment), 익스텐트(Extent), 데이터 블록(Block)으로 구성된다.

구성요소	설명
세그먼트	익스텐트의 집합이다. 하나의 테이블, 인덱스 등에 대응되는 것으로, CREATE TABLE 등의 문장을 실행하면 생성된다.
익스텐트	연속된 데이터 블록의 집합이다.

구성요소	설명
	세그먼트를 처음 만들거나, 세그먼트의 저장 공간이 더 필요한 경우, Tibero RDBMS는 테이블스페이스에서 연속된 블록의 주소를 갖는 데이터 블록을 할당 받아 세그먼트에 추가한다.
데이터 블록	데이터베이스에서 사용하는 데이터의 최소 단위이다. Tibero RDBMS는 데이터를 블록(Block) 단위로 저장하고 관리한다.

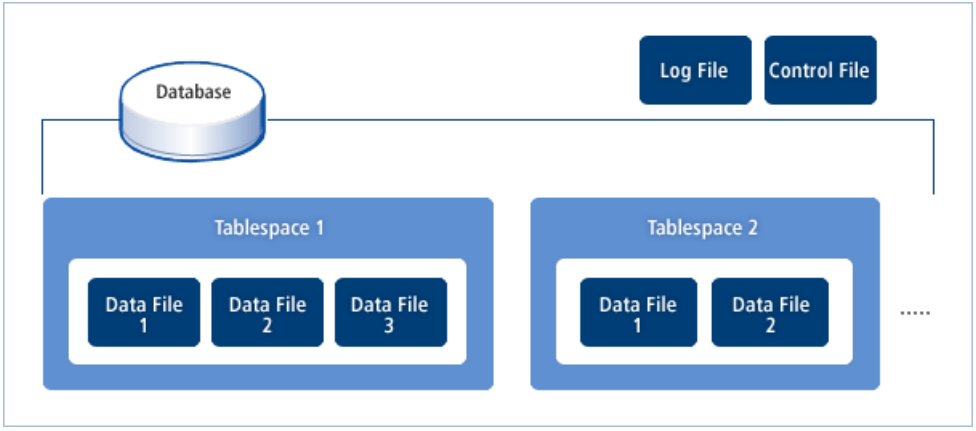
참고

논리적 저장 영역을 관리하는 방법에 대한 자세한 내용은 "제4장 스키마 객체의 관리"를 참고한다.

테이블스페이스의 물리적 구성

다음은 테이블스페이스의 물리적 구성을 나타내는 그림이다.

[그림 3.2] 테이블스페이스의 물리적 구성



테이블스페이스는 [그림 3.2]와 같이 물리적으로 여러 개의 데이터 파일로 구성된다. Tibero RDBMS는 데이터 파일 외에도 컨트롤 파일과 로그 파일을 이용하여 데이터를 저장할 수 있다.

빈번하게 사용되는 두 테이블스페이스(예: 테이블과 인덱스)는 물리적으로 서로 다른 디스크에 저장하는 것이 좋다. 왜냐하면 한 테이블스페이스를 액세스하는 동안에 디스크의 헤드가 그 테이블스페이스에 고정되어 있기 때문에, 다른 테이블스페이스를 액세스할 수 없다.

따라서, 서로 다른 디스크에 각각의 테이블스페이스를 저장하여 동시에 액세스하는 것이 데이터베이스 성능을 향상시키는 데 도움이 된다.

참고

테이블스페이스 안에서 특정한 데이터 파일을 사용할 수 있도록 임의로 지정할 수 없다. 또한, 테이블스페이스 내의 모든 데이터 블록은 구분되지 않고 저장 공간에 할당된다.

3.3.2. 테이블스페이스의 생성, 제거

테이블스페이스는 생성되는 유형에 따라 **시스템 테이블스페이스**(System Tablespace)와 **비시스템 테이블스페이스**(Non System Tablespace)로 구분된다.

시스템 테이블스페이스는 데이터베이스가 생성될 때 자동으로 생성되는 테이블스페이스고, 비시스템 테이블스페이스는 일반 사용자가 생성한 테이블스페이스이다.

본 절에서는 비시스템 테이블스페이스를 생성하고 제거하는 방법을 설명한다.

테이블스페이스의 생성

테이블스페이스를 생성하기 위해서는 **CREATE TABLESPACE** 문을 사용해야 한다. 테이블스페이스의 이름, 테이블스페이스에 포함되는 데이터 파일과 데이터 파일의 크기, 익스텐트의 크기 등을 설정할 수 있다. 다음은 하나의 데이터 파일 my_file.tdf로 구성되는 테이블스페이스 my_space를 생성하는 예이다.

```
CREATE TABLESPACE my_space
    DATAFILE '/usr/tibero/tdf/my_file.tdf' SIZE 50M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;
```

데이터 파일 my_file.tdf는 SQL 문장을 실행함과 동시에 생성된다. 만약 동일한 이름의 파일이 이미 사용중이라면 에러를 반환하게 된다. 데이터 파일 my_file.tdf의 크기는 50MB이며, 테이블스페이스의 크기도 50MB가 된다.

참고

Tibero RDBMS에서는 데이터 파일마다 데이터 블록을 2^22개까지 관리한다. 따라서 데이터 파일의 최대 크기는 데이터 블록의 크기 * 2^22 이다. 예를 들어, 데이터 블록의 크기가 8KB라고 한다면 데이터 파일의 최대 크기는 32GB이다.

Tibero RDBMS에서는 하나의 테이블스페이스 내의 모든 익스텐트의 크기를 항상 일정하게 관리한다. 예를 들어, 하나의 익스텐트의 크기가 256KB이고, 데이터 블록의 크기가 4KB라고 한다면 하나의 익스텐트에는 총 64개의 데이터 블록이 포함된다.

또한, 하나의 테이블스페이스를 두 개 이상의 데이터 파일로 구성할 수도 있다.

예를 들면 다음과 같다.

```
CREATE TABLESPACE my_space2
    DATAFILE '/usr/tibero/tdf/my_file21.tdf' SIZE 20M,   ... ① ...
             '/usr/tibero/tdf/my_file22.tdf' SIZE 30M    ... ② ...
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 64K;            ... ③ ...
```

① 20MB 크기의 데이터 파일 my_file21.tdf를 정의한다.

② 30MB 크기의 데이터 파일 my_file22.tdf를 정의한다. 테이블스페이스 my_space2의 전체 크기는 총 50MB가 된다.

③ 테이블스페이스 my_space2는 하나의 익스텐트의 크기가 64KB로 설정한다.

하나의 테이블스페이스에 포함되는 데이터 파일의 개수는 데이터베이스와 시스템 환경에 따라 달라진다. 하나의 테이블스페이스에 많은 데이터가 저장되면 여러 개의 데이터 파일로 테이블스페이스를 생성해야 한다. 단, 운영체제에 따라 동시에 처리할 수 있는 데이터 파일의 최대 개수가 달라질 수 있으므로, 범위 내에서 데이터 파일의 개수를 조정해야 한다.

데이터 파일의 크기는 데이터베이스의 크기를 추정하여 설정해야 한다. 테이블스페이스를 생성할 때 설정된 크기보다 더 많은 공간이 필요할 것에 대비하여 데이터 파일의 크기가 자동으로 확장되도록 설정할 수도 있다.

다음은 CREATE TABLESPACE 문의 DATAFILE 절에 AUTOEXTEND 절을 추가하여 저장 공간이 더 필요할 것에 대비하여 1MB씩 확장하도록 설정하는 예이다.

```
CREATE TABLESPACE my_space
    DATAFILE '/usr/tibero/tdf/my_file.tdf' SIZE 50M
    AUTOEXTEND ON NEXT 1M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;
```

Tibero RDBMS에서는 데이터 블록을 할당한 정보를 테이블스페이스에 비트맵 형태로 저장한다. 따라서, 테이블스페이스 내의 익스텐트의 최대 개수는 (테이블스페이스의 크기 / 익스텐트의 크기)보다 작은 값이 된다.

테이블스페이스의 제거

테이블스페이스를 제거하기 위해서는 **DROP TABLESPACE** 문을 사용해야 한다. 단, 테이블스페이스를 제거하면 그 안에 포함되어 있는 모든 스키마 객체가 제거되므로, 주의해야 한다.

다음은 테이블스페이스를 제거하는 예이다.

```
DROP TABLESPACE my_space;
```

이 SQL 문장을 실행하면 데이터베이스에서 테이블스페이스는 제거되지만, 테이블스페이스를 구성하는 데이터 파일은 삭제되지 않는다. 데이터 파일까지 제거하려면 다음과 같이 **INCLUDING** 절을 삽입하여 DROP TABLESPACE 문을 실행해야 한다.

```
DROP TABLESPACE my_space INCLUDING CONTENTS AND DATAFILES;
```

참고

테이블스페이스를 생성하거나 제거하면 그러한 내용이 컨트롤 파일에 동시에 반영된다.

CREATE TABLESPACE, DROP TABLESPACE 문에 대한 자세한 내용은 "Tibero RDBMS SQL 참조 안내서"를 참고한다.

3.3.3. 테이블스페이스의 변경

테이블스페이스의 저장 공간이 더 필요한 경우 데이터 파일이 자동으로 확장하도록 설정하는 방법도 있지만, **ALTER TABLESPACE** 문에서 **ADD DATAFILE** 절을 삽입하여 새로운 데이터 파일을 테이블스페이스에 추가하는 방법도 있다.

다음은 테이블스페이스 my_space에 새로운 데이터 파일 my_file02.tdf를 추가하는 예이다.

```
ALTER TABLESPACE my_space ADD DATAFILE 'my_file02.tdf' SIZE 20M;
```

데이터 파일을 추가할 때 위의 예처럼 절대 경로를 명시하지 않으면, 디폴트로 설정된 디렉터리에 데이터 파일이 생성된다. 이때 생성되는 데이터 파일의 개수는 하나 이상이 될 수 있으며, 각 데이터 파일에 대한 크기를 자동으로 확장하도록 설정할 수 있다.

참고

데이터 파일의 절대 경로를 명시하지 않았을 때, 디폴트로 생성되는 위치는 초기화 파라미터 파일($TB_HOME/config/$TB_SID.tip)에 설정된 DB_CREATE_FILE_DEST이다. 단, 해당 파라미터가 정의되지 않았으면 디폴트 위치는 $TB_HOME/database/$TB_SID이다.

또한, 데이터 파일은 ALTER DATABASE 문을 통해 크기를 변경할 수도 있다. ALTER DATABASE 문을 사용하면 데이터 파일의 크기를 늘리거나 줄이는 것이 모두 가능하다. 단, 데이터 파일의 크기를 줄이는 경우, 데이터 파일 안에 저장되어 있는 스키마 객체의 전체 크기보다 작을 경우에는 에러가 발생된다.

다음은 데이터 파일 my_file01.tdf의 크기를 변경하는 예이다.

```
ALTER DATABASE DATAFILE 'my_file01.tdf' RESIZE 100M;
```

3.3.4. 테이블스페이스의 정보 조회

Tibero RDBMS에서는 테이블스페이스를 효율적으로 관리하기 위해 다음 표에 나열된 뷰(정적 뷰, 동적 뷰 포함)를 통해 테이블스페이스의 정보를 제공하고 있다.

테이블스페이스 내의 익스텐트의 크기 및 개수, 할당된 서버, 포함된 데이터 파일의 이름 및 크기, 세그먼트의 이름 및 종류, 크기 등의 정보를 제공한다.

뷰	설명
DBA_TABLESPACES	Tibero RDBMS 내의 모든 테이블스페이스의 정보를 조회하는 뷰이다.
USER_TABLESPACES	현재 사용자에 속한 테이블스페이스의 정보를 조회하는 뷰이다.
V$TABLESPACE	Tibero RDBMS내의 모든 테이블스페이스에 대한 간략한 정보를 조회하는 뷰이다.

참고

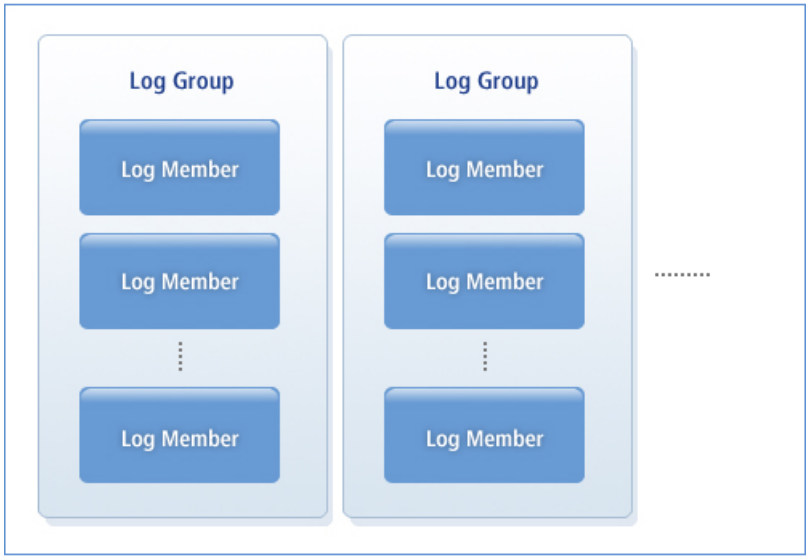
정적 뷰와 동적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

3.4. 로그 파일

로그 파일은 Redo Log를 저장하는 파일이다. **Redo 로그**는 데이터베이스에서 발생하는 모든 변경 내용을 포함하며, 데이터베이스에 치명적인 에러가 발생한 경우, 커밋된 트랜잭션의 갱신된 내용을 복구하는 핵심적인 데이터 구조이다.

다음은 Redo Log의 구조를 나타내는 그림이다.

[그림 3.3] Redo 로그의 구조



Redo 로그

[그림 3.3]와 같이, Redo 로그는 두 개 이상의 로그 그룹(Log Group)으로 구성된다. Tibero RDBMS에서는 이러한 로그 그룹을 순환적(Circular)으로 사용한다.

예를 들어, 세 개의 로그 그룹으로 Redo 로그를 구성하는 경우, 먼저 로그 그룹 1에 로그를 저장한다. 로그 그룹 1에 로그가 가득 차면, 그 다음 로그 그룹 2, 3에 로그를 저장한다. 로그 그룹 3까지 로그가 가득 차면 로그 그룹 1부터 다시 저장한다. 이처럼 하나의 로그 그룹을 모두 사용하고 그 다음 로그 그룹을 사용하는 것을 **로그 전환**(log switch)이라고 한다.

Redo 로그에는 하나 이상의 로그 레코드(log record)가 저장된다. 로그 레코드에는 데이터베이스에서 발생하는 모든 변경 내용이 포함되어 있으며, 이전에 변경된 값과 새로운 변경 값이 함께 저장된다.

Tibero RDBMS는 동시에 하나의 로그 그룹만을 사용하는 데, 현재 사용 중인 로그 그룹을 활성화(active)된 로그 그룹이라고 한다.

하나의 로그 그룹은 하나 이상의 로그 멤버로 구성할 수 있다. 이러한 구성을 다중화(multiplexing)라고 한다. 단, 다중화를 하려면 동일한 그룹에 속해 있는 모든 로그 멤버의 크기는 일정해야 하며, 동일한 데이터를 저장하고 동시에 갱신되어야 한다. 반면에 서로 다른 영역에 있는 로그 그룹은 각각 다른 개수의 로그 멤버를 포함할 수 있으며, 로그 멤버의 크기가 같지 않아도 된다.

하나의 로그 그룹을 여러 로그 멤버로 구성하는 이유는 일부 로그 멤버가 손상되더라도 다른 로그 멤버를 사용하기 위함이다. 디스크가 대단히 신뢰성이 높거나 데이터가 손실되어도 큰 문제가 없다면 다중화를 하지 않아도 된다.

ARCHIVELOG 모드의 설정

Redo 로그에 저장된 내용을 제3의 저장 장치에 반영구적으로 저장할 수 있다. 이러한 과정을 아카이브(Archive)라고 하며, 디스크 상에 로그 파일이 손상될 경우를 대비하는 작업이다. 아카이브에 사용되는 저장 장치로는 대용량 하드 디스크 또는 테이프 등이 있다.

Tibero RDBMS에서는 Redo 로그를 사용하지 않을 때나, 데이터베이스와 함께 사용 중인 경우에도 동시에 아카이브를 수행할 수 있다. Redo 로그를 사용하는 중에 아카이브를 하려면 로그 아카이브 모드를 **ARCHIVELOG**로 설정해야 한다.

ARCHIVELOG 모드는 마운트(MOUNT) 상태에서 다음의 SQL 문장을 실행하여 설정할 수 있다.

```
SQL> ALTER DATABASE ARCHIVELOG;
```

ARCHIVELOG 모드에서는 아카이브가 되지 않은 로그 그룹은 재사용되지 않는다. 예를 들어, 로그 그룹 1를 전부 사용하고 나서 로그 그룹 2를 사용하려고 할 때, 로그 그룹 2 이전에 저장된 로그가 아카이브가 되지 않은 경우에는 로그 그룹 2가 아카이브가 될 때까지 대기해야 한다. 이때 읽기 전용이 아닌 모든 트랜잭션은 실행이 잠시 중지된다. 로그 그룹 2가 아카이브가 되면 바로 활성화되어 로그를 저장한다. 또한, 잠시 중지되었던 트랜잭션도 모두 다시 실행 된다. DBA는 이러한 일이 발생하지 않도록 로그 그룹의 개수를 충분히 설정해야 한다.

NOARCHIVELOG 모드의 설정

Redo 로그를 사용하는 중에 아카이브를 수행하지 않으려면 로그 아카이브 모드를 **NOARCHIVELOG**로 설정해야 한다. NOARCHIVELOG 모드에서는 아카이브가 수행되지 않으며, 로그 그룹을 순환적으로 활성화하기 전에 아카이브 되기를 기다리는 경우가 발생하지 않으므로 데이터베이스의 성능이 향상된다.

하지만, 데이터베이스와 Redo 로그 자체에 문제가 발생하여 동시에 복구할 수 없는 경우라면, 이전에 커밋된 트랜잭션에 의해 갱신된 데이터를 모두 잃어버리게 된다. 따라서, NOARCHIVELOG 모드에서는 복구가 제한적으로 이루어지므로 항상 데이터베이스 전체를 백업할 것을 권장한다.

3.4.1. 로그 파일의 구성

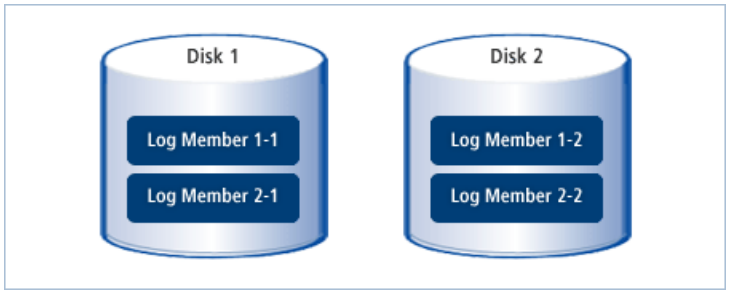
로그 멤버는 기본적으로 하나의 로그 파일이다. Redo 로그를 구성할 때, 각 로그 그룹과 로그 멤버에 서로 다른 로그 파일을 할당해야 한다. 로그 파일은 데이터 파일과 서로 다른 디스크에 저장할 것을 권장한다. 로그 파일과 데이터 파일을 같은 디스크에 저장하는 경우, 디스크에 장애가 발생한다면 데이터베이스를 다시 복구할 수 없게 된다. 각 로그 그룹이 여러 개의 로그 멤버로 구성된다면, 최소한 로그 멤버 하나는 데이터 파일과 다른 디스크에 저장되어야 한다.

로그 멤버의 다중화

로그 그룹 하나에 포함된 로그 멤버는 시스템의 성능을 위해 서로 다른 디스크에 저장하는 것이 좋다. 같은 로그 그룹 내의 모든 멤버는 같은 로그 레코드를 저장해야 한다. 모든 로그 멤버가 서로 다른 디스크에 존재하게 된다면, 로그 레코드를 저장하는 과정을 동시에 수행할 수 있다.

다음은 같은 로그 그룹의 모든 로그 멤버를 서로 다른 디스크에 배치한 그림이다.

[그림 3.4] 로그 멤버의 다중화



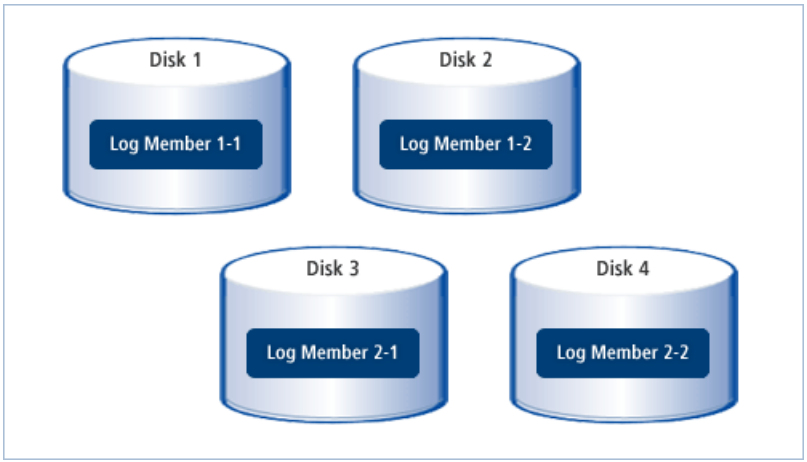
[그림 3.4]에서 Log Member 1-1은 로그 그룹 1의 첫 번째 로그 멤버라는 의미이다. 하나의 디스크에 같은 그룹의 로그 멤버가 존재한다면 동시에 같은 로그 레코드를 저장할 수 없다. 이 때문에 데이터베이스 시스템의 성능이 저하되는 원인이 되기도 한다.

로그 아카이브 모드를 **ARCHIVELOG**로 설정했을 때, Redo 로그 안에 활성화된 로그 그룹의 로그가 저장됨과 동시에 비활성화된 로그 그룹 중 하나에 대해서 아카이브가 수행된다. 활성화된 로그 그룹과 아카이브 중인 로그 그룹이 한 디스크에 존재하게 된다면 이 또한 데이터베이스 시스템의 성능이 저하되는 원인이 된다. 따라서, 서로 다른 로그 그룹의 로그 파일은 각각 다른 디스크에 저장하는 것이 시스템 성능을 높이는 데 도움이 된다.

로그 그룹의 다중화

다음은 두 개의 로그 멤버로 구성된 두 개의 로그 그룹을 서로 다른 디스크에 분리하여 배치한 그림이다.

[그림 3.5] 로그 그룹의 다중화



[그림 3.5]에서 Log Member 1-1은 로그 그룹 1의 첫 번째 로그 멤버라는 의미이다.

로그 그룹의 크기와 개수를 정할 때는 아카이브 작업을 충분히 고려해야 한다. 로그 그룹의 크기는 제3의 저장 장치에 빠르게 전달하고 저장 공간을 효율적으로 사용할 수 있도록 설정해야 한다. 또한, 로그 그룹의 개수는 아카이브 중인 로그 그룹을 대기하는 경우가 발생하지 않도록 해야 한다.

로그 그룹의 크기와 개수는 데이터베이스를 실제로 운영하면서 변경해야 한다. 즉, 데이터베이스에 최적화된 파라미터를 설정한 후 로그 그룹의 크기와 개수를 증가시켜가면서, 데이터베이스 처리 성능에 무리가 가지 않는 범위에서 변경해야 한다.

3.4.2. 로그 파일의 생성, 제거

새로운 로그 그룹 또는 로그 그룹에 포함되어 있는 로그 멤버를 생성하거나 제거하려면 **ALTER DATABASE** 문을 사용해야 한다. 단, ALTER DATABASE 문을 사용하기 위해서는 시스템 특권이 필요하다.

로그 파일의 생성

새로운 로그 그룹을 생성하려면 ALTER DATABASE 문에 **ADD LOGFILE** 절을 삽입해야 한다. 이 절은 로그 파일을 추가할 때 사용한다. 단, 로그 파일을 추가할 때에는 로그 그룹 단위로만 해야 한다.

다음은 두 개의 로그 멤버로 구성된 로그 그룹을 추가하는 예이다. 본 예제에서는 두 로그 멤버의 크기를 512KB로 설정한다. 이때 두 로그 멤버의 크기는 항상 같아야 한다.

```
ALTER DATABASE ADD LOGFILE (
        '/usr/tibero/log/my_log21.log',
        '/usr/tibero/log/my_log22.log') SIZE 512K
```

또한, ADD LOGFILE 절에 로그 그룹의 번호를 지정할 수 있는 GROUP 옵션을 추가할 수 있다. 로그 그룹의 번호를 설정하면, 이후에 특정한 로그 그룹을 지칭하여 로그 멤버를 추가하는 등의 작업을 수행할 수 있다.

다음은 로그 그룹에 번호를 설정하는 예이다.

```
ALTER DATABASE ADD LOGFILE GROUP 5 (
        '/usr/tibero/log/my_log21.log',
        '/usr/tibero/log/my_log22.log') SIZE 512K
```

기존의 로그 그룹에 새로운 로그 멤버를 추가하려면 **ADD LOGFILE MEMBER** 절을 삽입해야 한다. 이때 로그 파일에 할당된 서버 프로세스를 반드시 명시해야 한다.

다음의 SQL 문장은 로그 그룹 5에 새로운 로그 멤버를 추가하는 예이다.

```
ALTER DATABASE ADD LOGFILE MEMBER
        '/usr/tibero/log/my_log25.log' TO GROUP 5
```

새로운 로그 멤버를 추가할 때에는 로그 파일의 크기를 지정하면 안 된다. 로그 파일의 크기는 같은 로그 그룹 내의 로그 멤버의 크기와 동일하게 설정한다.

로그 파일의 제거

로그 그룹을 제거하려면 **DROP LOGFILE** 절을 삽입해야 한다.

다음의 SQL 문장은 로그 그룹 5를 제거하는 예이다.

```
ALTER DATABASE DROP LOGFILE GROUP 5;
```

다음은 로그 그룹을 제거하기 전에 고려해야 할 사항이다.

- 로그 그룹이 둘 이상인가?

  Tibero RDBMS는 로그 그룹을 최소한 두 개 이상 가져야 한다.

- 로그 그룹을 제거한 후 남은 로그 그룹의 개수가 하나인가?

  남은 로그 그룹의 개수가 하나이면 에러를 반환한다.

- 현재 활성화되어 사용 중인 로그 그룹인가?

  로그 그룹은 제거되지 않는다.

- ARCHIVELOG 모드에서 아카이브 되지 않은 로그 그룹인가?

  로그 그룹은 제거되지 않는다.

로그 그룹 내의 하나의 로그 멤버를 제거하기 위해서는 **DROP LOGFILE MEMBER** 절을 삽입해야 한다. 로그 그룹이 할당된 서버를 명시해야 하며, 로그 그룹은 명시하지 않아도 된다.

다음의 SQL 문장은 로그 멤버 하나를 제거하는 예이다.

```
ALTER DATABASE DROP LOGFILE MEMBER '/usr/tibero/log/my_log25.log'
```

로그 멤버도 로그 그룹을 제거할 때처럼 로그 그룹 내에 남겨진 로그 멤버가 하나도 없는 경우, 에러를 반환하게 된다. 뿐만 아니라 현재 활성화되어 사용 중이거나 ARCHIVELOG 모드에서 아카이브 되지 않은 로그 그룹 내의 로그 멤버도 제거되지 않는다.

3.4.3. 로그 파일의 정보 조회

Tibero RDBMS에서는 Redo 로그 관리에 도움을 주기 위해 다음 표에 나열된 동적 뷰를 제공하고 있다. Redo 로그의 그룹별 로그 파일, 다중화 정보, 갱신 날짜 등의 정보를 제공하며, DBA나 일반 사용자 모두가 이 뷰를 사용할 수 있다.

동적 뷰	설명
V$LOG	로그 그룹의 정보를 조회하는 뷰이다.
V$LOGFILE	로그 파일의 정보를 조회하는 뷰이다.

참고

뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

3.5. 컨트롤 파일

컨트롤 파일은 데이터베이스 자체의 메타데이터를 보관하고 있는 바이너리 파일이다. 최초의 컨트롤 파일은 Tibero RDBMS를 설치할 때 함께 생성된다. 최초로 설정된 컨트롤 파일에 대한 정보는 **$TB_SID.tip** 파일에 저장된다.

컨트롤 파일은 Tibero RDBMS에 의해서만 생성과 갱신을 할 수 있다. 단, DBA가 컨트롤 파일의 내용을 조회하거나 갱신할 수는 없다.

컨트롤 파일에는 다음과 같은 정보가 포함되어 있다.

정보	설명
데이터베이스	데이터베이스 이름, $TB_SID.tip 파일의 이름 또는 생성되었거나 변경된 타임스탬프 등이 있다.
테이블스페이스	테이블스페이스를 구성하는 데이터 파일 또는 생성되었거나 변경된 타임스탬프 등이 있다.
데이터 파일	데이터 파일의 이름과 위치 또는 생성되었거나 변경된 타임스탬프 등이 있다.

정보	설명
Redo 로그	로그 그룹의 개수 및 이를 구성하는 로그 멤버(로그 파일)의 이름과 위치 또는 생성되었거나 변경된 타임스탬프 등이 있다.
체크포인트	최근 체크포인트를 수행한 타임스탬프 등이 있다.

Tibero RDBMS에서는 데이터베이스를 다시 기동할 때마다 먼저 컨트롤 파일을 참조한다.

참조하는 절차는 다음과 같다.

1. 테이블스페이스와 데이터 파일의 정보를 얻는다.

2. 데이터베이스에 실제 저장된 데이터 사전과 스키마 객체의 정보를 얻는다.

3. 필요한 데이터를 읽는다.

Tibero RDBMS에서 컨트롤 파일은 같은 크기, 같은 내용의 파일을 두 개 이상 유지하기를 권장한다. 이는 Redo 로그 멤버를 다중화하는 방법과 유사하다.

같은 로그 그룹 내의 로그 멤버를 서로 다른 디스크에 설치하는 것처럼, 컨트롤 파일의 복사본을 서로 다른 디스크에 저장하는 것이 좋다. 이는 데이터베이스의 시스템 성능과 안정성을 유지하는 데 매우 필요하다.

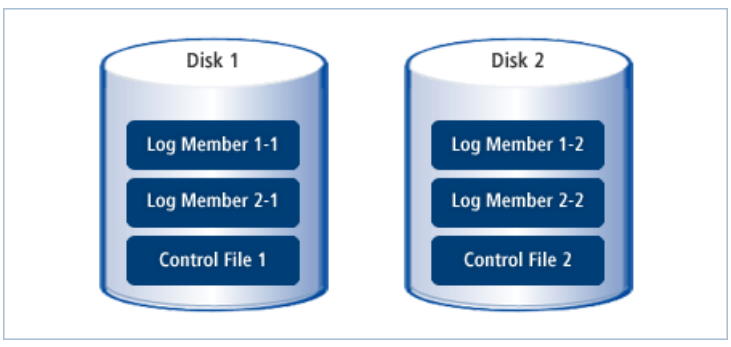
예를 들어, 한 디스크에 컨트롤 파일의 복사본이 존재하는 경우 문제가 발생할 수 있다. 만약 이 디스크를 영구적으로 사용할 수 없게 된다면, 컨트롤 파일은 복구할 수 없는 상태가 된다. 따라서 컨트롤 파일은 Redo 로그와 연관하여 배치하는 것이 좋다.

주의

컨트롤 파일은 데이터베이스를 운영할 때 매우 중요한 파일이므로 컨트롤 파일이 손상되지 않도록 주의해야 한다.

다음은 컨트롤 파일을 다중화한 그림이다.

[그림 3.6] 컨트롤 파일의 다중화



위 그림에서 보듯이, 디스크마다 하나의 로그 그룹에 여러 로그 멤버를 배치한 것처럼 컨트롤 파일의 복사본을 같은 위치에 배치한다.

Tibero RDBMS에서는 컨트롤 파일로부터 정보를 확인할 때 여러 복사본 중에서 하나만 읽는다. 그리고 테이블스페이스의 변경 등의 이유로 컨트롤 파일을 갱신해야 하는 경우에는 모든 복사본을 동시에 갱신한다.

컨트롤 파일의 갱신을 유발하는 SQL 문장은 모두 DDL 문장이다. DDL 문장의 특징은 하나의 문장이 하나의 트랜잭션이 된다는 것이다. 따라서, DDL 문장을 실행하면 바로 커밋되며, 갱신된 내용은 바로 디스크에 반영된다.

3.5.1. 컨트롤 파일의 변경

DBA는 컨트롤 파일의 복사본을 추가하거나 제거할 수 있다. 컨트롤 파일은 DBMS에 대한 메타데이터이므로, 데이터베이스를 운영 중일 때에는 컨트롤 파일의 변경이 불가능하다. 따라서, 컨트롤 파일의 복사본을 추가 또는 제거하기 위한 SQL 문장은 존재하지 않는다. 반드시 데이터베이스를 종료한 후, 컨트롤 파일을 변경해야 한다.

이처럼 컨트롤 파일을 추가 또는 제거하기 위한 SQL 문장이 존재하지 않기 때문에 일반적인 운영체제 명령어를 사용하여 변경 작업을 수행해야 한다. 그 다음 변경된 내용을 $TB_SID.tip 파일에 반영한다.

다음은 UNIX Command에서 컨트롤 파일을 복사하는 예이다.

```
$ cp /usr1/tibero/control01.ctl /usr3/tibero/control03.ctl
```

위의 예에서 usr1과 usr3은 서로 다른 디스크에 존재하는 디렉터리이다.

Tibero RDBMS는 데이터베이스를 다시 기동하면서 $TB_SID.tip 파일을 읽고, 변경된 내용에 따라 컨트롤 파일의 갱신을 수행한다. 이때 유의할 점은 $TB_SID.tip 파일 내에 설정된 컨트롤 파일의 이름은 절대 경로를 포함한 이름이어야 한다.

디스크 에러 등의 원인으로 컨트롤 파일의 복사본 중 하나라도 문제가 발생한다면, Tibero RDBMS는 즉시 데이터베이스 운영을 중지한다. 컨트롤 파일에 문제가 있는 상태에서 Tibero RDBMS를 다시 기동할 수 없으므로, DBA는 문제가 있는 컨트롤 파일의 복사본을 제거하거나 문제가 없는 컨트롤 파일의 복사본을 다시 복사해야 한다. 이때 컨트롤 파일의 변경 내용은 반드시 $TB_SID.tip 파일에 반영시켜야 한다.

컨트롤 파일의 백업은 논리적 백업만을 지원한다. 따라서 컨트롤 파일을 생성하는 SQL 문장을 백업해야 한다.

특히 테이블스페이스, 데이터 파일, Redo 로그를 새로 생성하거나 변경 또는 제거를 수행한 경우에는 바로 컨트롤 파일을 백업하는 것이 관리 측면에서 안전하다. 물론 데이터베이스 전체를 백업할 때에도 컨트롤 파일 자체를 백업해야 한다.

참고

자세한 내용은 “6.4. 백업의 실행”을 참고한다.

다음의 SQL 문장은 컨트롤 파일을 백업하는 예이다.

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE AS
          '/tibero4/backup/ctrlfile1.sql' REUSE NORESETLOGS;
```

위 예에서 보듯이, 백업할 컨트롤 파일의 복사본(ctrlfile1.sql)은 기존의 복사본과 다른 디스크에 저장해야 하므로, 반드시 절대 경로를 포함한 이름을 명시해야 한다.

3.5.2. 컨트롤 파일의 정보 조회

Tibero RDBMS에서는 컨트롤 파일을 관리하는 데 도움을 주기 위해 다음 표에 동적 뷰를 제공하고 있다. 데이터베이스 생성 시간, 체크포인트 정보 등의 정보를 제공하며, DBA나 일반 사용자 모두가 이 뷰를 사용할 수 있다.

동적 뷰	설명
V$DATABASE	ARCHIVELOG 모드 여부와 체크포인트 등의 정보를 조회하는 뷰이다.
V$CONTROLFILE	컨트롤 파일의 이름과 상태 등의 정보를 조회하는 뷰이다.

참고

동적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

제4장 스키마 객체의 관리

본 장에서는 Tibero RDBMS에서 실제로 데이터베이스를 구성하는 데 필요한 **논리적 저장 영역** 즉 **스키마 객체**를 관리하는 방법과 이를 구성하는 최소 단위인 **디스크 블록**을 설명한다.

다음은 데이터베이스의 대표적인 스키마 객체이다.

- 테이블 (Table)
- 인덱스 (Index)
- 뷰 (View)
- 시퀀스 (Sequence)
- 동의어 (Synonym)

스키마 객체는 한 스키마에 의해 생성되며 또한, 그 스키마에 속하게 된다.

여기서 테이블, 인덱스와 같이 실제 물리적 공간을 가지는 객체를 세그먼트라고 한다.

4.1. 테이블

테이블은 관계형 데이터베이스에서 가장 기본적인 스키마 객체이다.

다른 스키마 객체의 형태로 표현하더라도 대부분의 데이터베이스 처리는 테이블에서 이루어진다. 따라서, 관계형 데이터베이스의 설계는 테이블의 설계가 가장 중심이 되며, 테이블을 효율적으로 관리하는 것이 데이터베이스 성능에 큰 영향을 미친다.

테이블은 다음과 같이 두 가지 구성요소로 이루어진다.

구성요소	설명
컬럼 (column)	테이블에 저장될 데이터의 특성을 설정한다.
로우 (row)	하나의 테이블을 구성하며 다른 유형의 데이터가 저장된다.

데이터베이스를 효율적으로 운영하기 위해서는 테이블 설계를 정확하게 해야 하며, 테이블의 배치와 저장 환경설정 등을 고려해야 한다. Tibero RDBMS는 테이블을 생성하고 나서 설계를 변경하는 것을 어느 정도 허용하고 있다. 그러나, 테이블의 설계를 변경하려면 처리 비용이 많이 들기 때문에 될 수 있으면 잦은 변경은 하지 말아야 한다.

테이블을 정확하게 설계하기 위해서는 정규화(normalization) 과정과 각 테이블에 적절한 무결성 제약조건(integrity constraints)을 설정해야 한다. 예를 들어, 조인 연산을 피하기 위해 테이블 간의 중복된 데이터를 허용하는 경우 데이터의 일관성 유지를 위해 어떻게 테이블을 설계할 것인지를 고려해야 한다.

4.1.1. 테이블의 생성, 변경, 제거

본 절에서는 테이블을 생성, 변경, 제거하는 방법을 설명한다.

테이블의 생성

테이블은 다음 같은 경우에 따라 생성 요건이 다르다.

- 현재 사용자가 자신의 스키마에 테이블을 생성하는 경우 **CREATE TABLE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.
- 다른 사용자의 스키마에 테이블을 생성하는 경우 **CREATE ANY TABLE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

테이블을 생성하기 위해서는 **CREATE TABLE** 문을 사용해야 한다.

다음은 테이블을 생성할 때 포함되는 구성요소이다.

구성요소	설명
테이블의 이름	- 테이블의 이름을 설정한다. 이 구성요소는 테이블을 생성할 때 반드시 포함되어야 한다. - 테이블의 이름은 최대 128자로 설정할 수 있다. - 한 사용자의 스키마 내에서 유일해야 하며, 모든 스키마 객체의 이름과 달라야 한다. - 서로 다른 사용자는 같은 이름의 테이블을 소유할 수 있다. - 인덱스, 트리거, 대용량 객체형과도 같은 이름을 사용할 수 있다. - 공용 동의어(public synonym)와 같은 이름도 테이블의 이름으로 사용할 수 있다. 공용 동의어의 이름을 SQL 문장에서 사용하면 공용 동의어라는 의미 대신 현재 사용자가 소유한 테이블이 된다.
테이블의 컬럼 구조	- 테이블에 저장될 데이터의 특성(컬럼 이름, 데이터 타입, 디폴트 값 등)을 설정한다. 이 구성요소는 테이블을 생성할 때 반드시 포함되어야 한다. - 테이블은 하나 이상의 컬럼으로 구성되며, 각 컬럼은 반드시 데이터 타입을 선언해야 한다. - 한 테이블은 최대 1,000개의 컬럼으로 구성할 수 있다.

구성요소	설명
	- 컬럼의 이름은 최대 128자로 설정할 수 있다. - 컬럼의 디폴트 값과 제약조건은 선택적으로 선언할 수 있다.
무결성 제약조건	- 테이블의 컬럼에 사용자가 원하지 않는 데이터가 입력, 변경, 제거되는 것을 방지하기 위해 설정한다. 이 구성요소는 테이블을 생성할 때 선택적으로 사용할 수 있다. 자세한 내용은 "4.2. 제약조건"을 참고한다. - 기본 키 (PRIMARY KEY), 유일 키 (UNIQUE KEY), 참조 무결성(referential integrity), NOT NULL, CHECK 등의 제약조건이 있다. - 제약조건의 이름은 테이블 내에서 유일해야 한다. - 제약조건은 컬럼 또는 테이블 단위에서 설정할 수 있다. - 두 개 이상의 복합 컬럼을 사용하는 경우 제약조건을 따로 설정해야 한다. - ALTER TABLE 문을 사용하여 제약조건을 추가 또는 상태를 변경하거나 제거할 수 있다.
테이블스페이스	- 테이블이 저장될 테이블스페이스를 설정한다. 이 구성요소는 테이블을 생성할 때 선택적으로 사용할 수 있다. - 테이블스페이스를 명시하지 않으면 사용자의 디폴트 테이블스페이스로 설정된다. - 테이블의 적절한 배치가 데이터베이스 성능에 큰 영향을 미치므로, 테이블의 소유자는 테이블이 저장될 테이블스페이스를 별도로 명시하는 것이 좋다.
디스크 블록 파라미터	디스크 블록마다 테이블의 갱신에 대비하여 어느 정도 여유 공간을 남겨둘 것인가를 설정한다. 자세한 내용은 "4.3. 디스크 블록"을 참고한다. - PCTFREE - INITRANS
파티션	- 파티션을 정의한다.

다음은 테이블을 생성하는 예이다.

[예 4.1] 테이블의 생성

```
CREATE TABLE DEPT
  (
      DEPTNO    NUMBER PRIMARY KEY,
      DEPTNAME  VARCHAR(20),
      PDEPTNO   NUMBER
  )
  TABLESPACE my_space
```

```
  PCTFREE 5 INITRANS 3;

CREATE TABLE EMP
  (
      EMPNO     NUMBER PRIMARY KEY,
      ENAME     VARCHAR(16) NOT NULL,
      ADDR      VARCHAR(24),
      SALARY    NUMBER,
      DEPTNO    NUMBER,
      CHECK (SALARY >= 10000),
      FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)
  )
  TABLESPACE my_space
  PCTFREE 5 INITRANS 3;
```

위의 예에서 보듯이, 테이블 EMP는 다섯 개의 컬럼(EMPNO, ENAME, ADDR, SALARY, DEPNO)으로 구성되어 있으며, 네 개의 제약조건으로 선언되어 있다. 또한, 테이블스페이스 my_space에 저장되며, 디스크 블록 파라미터의 세부 항목(PCTFREE, INITRANS)을 모두 설정하였다.

테이블의 변경

테이블은 다음 같은 경우에 따라 변경 요건이 다르다.

- 현재 사용자가 자신의 스키마에 속한 테이블을 변경하는 경우 **ALTER TABLE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

- 다른 사용자의 스키마에 있는 테이블을 변경하는 경우 **ALTER ANY TABLE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

테이블을 변경하기 위해서는 **ALTER TABLE** 문을 사용해야 한다.

다음은 테이블을 변경할 때 포함되는 구성요소이다.

구성요소	설명
테이블의 이름	- 테이블의 이름을 변경한다. - 테이블의 이름은 최대 128자로 변경할 수 있다.
컬럼의 정의 변경	- 컬럼에 정의된 속성(디폴트 값, 제약조건 등)을 변경한다. - 컬럼의 디폴트 값과 제약조건은 MODIFY 절을 이용하여 변경한다. [예 4.2]를 참고한다.
컬럼의 이름	- 컬럼의 이름과 정의된 컬럼의 속성을 변경한다.

구성요소	설명
	- 컬럼 이름은 최대 30자로 변경할 수 있으며, RENAME COLUMN 절을 사용하여 변경한다. [예 4.3]을 참고한다.
디스크 블록의 파라미터	- 파라미터의 이름과 값을 지정한다. [예 4.4]를 참고한다.
제약조건	- 제약조건의 이름을 변경한다. - 제약조건을 추가하거나 제거한다. - 제약조건의 상태를 변경한다.
테이블스페이스	- 테이블에 할당된 테이블스페이스는 변경할 수 없다.
파티션	- 파티션을 추가하거나 제거한다.

다음은 [예 4.1]에서 생성한 EMP 테이블의 속성을 변경하는 예이다.

- 정의된 컬럼의 속성을 변경하는 경우

[예 4.2] 테이블의 변경 - 컬럼 속성

```
ALTER TABLE EMP
       MODIFY (SALARY DEFAULT 5000 NOT NULL);
```

SALARY 컬럼은 MODIFY 절을 사용하여 디폴트 값과 NOT NULL 제약조건으로 재정의한다. 본 예제에서는 컬럼 SALARY의 디폴트 값을 5000으로 하고, 동시에 SALARY 값이 NULL이 되지 못하도록 컬럼의 속성을 변경한다.

동시에 여러 컬럼의 속성을 변경할 수도 있다. 이때 각 컬럼의 내용은 콤마(,)로 분리하여 다시 정의한다.

- 컬럼의 이름을 변경하는 경우

[예 4.3] 테이블의 변경 - 컬럼 이름

```
ALTER TABLE EMP RENAME COLUMN ADDR TO ADDRESS;
```

ADDR 컬럼은 RENAME COLUMN 절을 사용하여 컬럼의 이름을 **ADDRESS**로 변경한다.

컬럼의 이름을 변경하면 이전에 컬럼 이름을 사용하던 제약조건 등은 Tibero RDBMS 시스템에 의해 자동으로 변경된다. 예를 들어, 위 SQL 문장이 실행되면 ADDR 컬럼에 설정된 제약조건이 ADDRESS 컬럼에 자동으로 적용된다. 단, CHECK 제약조건의 경우 제대로 동작하지 않을 수 있으며, 사용자가 ALTER TABLE 문을 이용하여 제약조건을 다시 정의해야 한다.

- 디스크 블록의 파라미터의 값을 변경하는 경우

[예 4.4] 테이블의 변경 - 디스크 블록의 파라미터

```
ALTER TABLE EMP PCTFREE 10;
```

디스크 블록의 파라미터의 값을 변경하려면 파라미터의 이름과 값을 지정하면 된다. 본 예제에서는 테이블 EMP의 PCTFREE 파라미터의 값을 5에서 10으로 변경한다.

- 제약조건을 변경하는 경우

  테이블에 설정된 제약조건과 상태를 변경하는 내용은 “4.2. 제약조건”에서 자세히 설명한다. 또한, 테이블을 생성하거나 변경을 위한 자세한 문법은 "Tibero RDBMS SQL 참조 안내서"를 참고한다.

테이블의 제거

테이블은 다음 같은 경우에 따라 제거 요건이 다르다.

- 현재 사용자가 자신의 스키마에 속한 테이블을 제거하는 경우 **DROP TABLE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

- 다른 사용자의 스키마에 있는 테이블을 제거하는 경우 **DROP ANY TABLE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

테이블을 제거하기 위해서는 **DROP TABLE** 문을 사용해야 한다.

다음은 테이블을 제거하는 예이다.

[예 4.5] 테이블의 제거

```
DROP TABLE EMP;
```

다른 사용자가 소유한 테이블을 제거하려면, 반드시 다른 사용자의 이름을 명시한 후 DROP TABLE 문을 실행해야 한다.

예를 들면 다음과 같다.

```
DROP TABLE John.EMP;
```

제거하려는 테이블의 기본 키가 다른 테이블의 참조 무결성 제약조건으로 정의된 경우, 참조된 테이블은 바로 제거할 수 없다. 이러한 경우에는 참조하는 테이블을 먼저 제거하거나, 참조하는 테이블에 정의된 참조 무결성 제약조건을 제거해야 한다.

참조하는 테이블에 정의된 참조 무결성 제약조건을 제거하기 위해서는 DROP TABLE 문에 **CASCADE CONSTRAINTS** 절을 삽입해야 한다.

예를 들면 다음과 같다.

```
DROP TABLE EMP CASCADE CONSTRAINTS;
```

4.1.2. 테이블의 효율적인 관리

테이블을 효율적으로 관리하기 위해서는 각각의 경우에 맞는 적절한 조치 방법을 수행해야 한다.

예를 들면 다음과 같은 경우이다.

- 동시에 액세스될 가능성이 높은 테이블인 경우 병렬 쿼리 처리가 수행될 가능성을 높이기 위해 서로 다른 디스크에 데이터를 저장한다.

  예를 들어, 조인이 이루어지는 SELECT 문에서 액세스할 두 개의 테이블의 SELECT 연산을 먼저 수행한 후, 조인 연산을 수행하는 경우라면, 이 두 테이블을 서로 다른 디스크에 저장하여 SELECT 연산이 병렬적으로 수행되도록 한다.

- 테이블이 저장될 디스크의 용량을 결정하는 경우 테이블의 최대 크기를 추정한다.

  테이블에 UPDATE 연산이 자주 발생하는 경우라면, 디스크 블록에 갱신을 위한 디스크 공간을 충분히 할당해야 한다. 디스크 공간을 할당하는 방법은 **PCTFREE** 파라미터를 삽입하여 설정하면 된다.

- 테이블에 동시에 액세스할 트랜잭션의 수를 결정하는 경우 동시에 액세스할 트랜잭션의 수를 추정한다.

  테이블을 구성하는 디스크 블록 안에 트랜잭션의 정보를 저장해야 하며, 얼마만큼의 디스크 공간을 할당할 것인지를 정해야 한다. 이러한 공간을 할당하는 방법은 INITRANS 파라미터를 삽입하여 설정한다.

  갱신에 대비하거나 테이블을 액세스할 트랜잭션의 정보를 저장할 디스크 공간이 필요한 경우, 같은 크기의 테이블이라도 좀 더 많은 디스크 공간을 필요로 한다.

  PCTFREE와 INITRANS 파라미터에 대한 자세한 내용은 “4.3. 디스크 블록”을 참고한다.

- 테이블에 INSERT 연산이 발생하는 경우 로그를 저장하는 디스크 공간을 할당한다.

  테이블에 INSERT 연산이 자주 발생하는 경우라면, Redo 로그를 구성하는 로그 그룹의 크기와 개수를 증가시켜야 하므로, 그만큼 디스크의 공간도 커져야 한다. 단, Redo 로그를 저장하는 디스크는 데이터를 저장하는 디스크와는 다른 것을 사용해야 한다.

4.1.3. 테이블의 정보 조회

Tibero RDBMS에서는 테이블의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
DBA_TABLES	Tibero RDBMS 내의 모든 테이블의 정보를 조회하는 뷰이다.
USER_TABLES	현재 사용자에 속한 테이블의 정보를 조회하는 뷰이다.
ALL_TABLES	현재 사용자가 접근 가능한 테이블의 정보를 조회하는 뷰이다.

정적 뷰	설명
DBA_TBL_COLUMNS	Tibero RDBMS 내의 모든 테이블과 뷰에 속한 컬럼의 정보를 조회하는 뷰이다.
USER_TBL_COLUMNS	현재 사용자에 속한 테이블 및 뷰에 속한 컬럼의 정보를 조회하는 뷰이다.
ALL_TBL_COLUMNS	현재 사용자가 접근 가능한 테이블 및 뷰에 속한 컬럼의 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

4.2. 제약조건

제약조건 (Constraints)은 사용자가 원하지 않는 데이터가 테이블의 컬럼에 삽입, 변경, 제거되는 것을 방지하는 방법이다.

4.2.1. 제약조건의 선언, 변경, 제거

본 절에서는 제약조건을 선언하고 변경, 제거하는 방법을 설명한다.

테이블을 생성할 때 제약조건을 선언하는 방법은 다음과 같다.

제약조건	설명
기본 키	무결성 제약조건과 고유 키 무결성 제약조건을 결합한 방법이다. 기본 키로 설정된 컬럼은 NULL 값을 가질 수 없다.
유일 키	테이블의 컬럼은 동일한 값을 가질 수 없다. 대신, NULL 값은 여러 컬럼에 입력할 수 있다.
참조 무결성	다른 테이블이나 현재 사용자가 소유한 테이블의 기본 키나 유일 키를 참조할 때 사용하는 방법이다.
NOT NULL	테이블의 컬럼은 NULL 값을 가질 수 없다. 테이블 레벨의 제약조건은 설정할 수 없다.
CHECK	삽입 또는 변경할 값이 만족해야 할 제약조건을 설정한다. 한 컬럼에 여러 개의 제약조건을 설정할 수 있다.

제약조건의 선언

제약조건은 삭제 가능성, 제약조건에 포함되는 컬럼의 개수 등에 따라 선택적으로 선언할 수 있다. 제약조건을 어떻게 선언하든 테이블 내에서 미치는 영향은 동일하다. 제약조건을 선언한 후, 정의 또는 상태를 변경하기 위해서는 제약조건을 선언할 때 제약조건의 이름을 설정해야 한다. 제약조건의 이름을 찾아 변경한다.

제약조건에 이름을 설정하기 위해서는 제약조건을 선언할 때 예약어 **CONSTRAINT**와 **제약조건의 이름**을 추가로 정의해야 한다.

다음은 [예 4.1]에서 선언한 제약조건에 이름을 설정하는 예이다.

[예 4.6] 제약조건의 이름 설정

```
CREATE TABLE DEPT
  (
      DEPTNO    NUMBER PRIMARY KEY,
      DEPTNAME  VARCHAR(20),
      PDEPTNO   NUMBER
  )
  TABLESPACE my_space
  PCTFREE 5 INITRANS 3;

CREATE TABLE EMP
  (
      EMPNO       NUMBER        PRIMARY KEY,
      ENAME       VARCHAR(16)   NOT NULL,
      ADDR        VARCHAR(24),
      SALARY      NUMBER,
      DEPTNO      NUMBER,
      CONSTRAINT  SALARY_MIN CHECK (SALARY >= 10000),
      CONSTRAINT  DEPTNO_REF FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)
  )
  TABLESPACE my_space
  PCTFREE 5 INITRANS 3;
```

제약조건은 다음과 같은 경우에 따라 사용하는 문법이 다르다.

- 컬럼 단위로 제약조건을 선언하는 경우

  컬럼의 정의와 함께 제약조건을 선언한다.

제약조건	설명
CHECK	문법이 동일하다.
NOT NULL	처음 선언할 때는 반드시 컬럼의 정의와 함께 선언되어야 한다. 선언된 이후에 변경할 경우 **ALTER TABLE MODIFY** 문을 사용한다.

다음은 컬럼 단위로 제약조건을 선언하는 예이다.

[예 4.7] 제약조건의 선언 - 컬럼 단위

```
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
);
```

본 예제에서는 컬럼 PROD_ID, PROD_NAME, PROD_COST, PROD_DATE 각각에 기본 키, NOT NULL 제약조건을 선언한다.

- 테이블 단위로 선언하는 경우

  모든 컬럼을 정의한 후 제약조건을 선언한다.

  다음은 테이블 단위로 제약조건을 선언하는 예이다.

[예 4.8] 제약조건의 선언 - 테이블 단위

```
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)
);
```

두 개 이상의 컬럼에 제약조건을 선언하고자 한다면 반드시 모든 컬럼을 정의한 후 선언해야 한다. 본 예제에서는 컬럼 PROD_ID와 PROD_NAME를 통합하여 기본 키 제약조건을 선언한다.

제약조건의 변경

Tibero RDBMS에서는 이미 선언된 제약조건을 변경할 수 있다. 제약조건은 ALTER TABLE 문에서 변경할 수 있다. 단, 변경할 수 있는 테이블에 한해서만 제약조건을 변경할 수 있다.

- 제약조건의 이름을 변경하는 경우

  ALTER TABLE 문의 **RENAME CONSTRAINT** 절을 삽입한다. 제약조건의 이름을 변경할 때에는 테이블 내에서 유일해야 한다.

다음은 제약조건의 이름을 변경하는 예이다.

[예 4.9] 제약조건의 변경 - 제약조건의 이름

```
ALTER TABLE EMP
        RENAME CONSTRAINT EMP_PRI_KEY TO EMP_KEY;
```

- 제약조건을 새로 추가하는 경우

제약조건을 새로 추가하려면 ALTER TABLE 문의 **ADD 절**을 삽입해야 한다. 사용하는 방법은 CREATE TABLE 문에서 컬럼을 정의한 후, 제약조건을 선언하는 문법과 동일하게 ADD 절 다음에 제약조건을 선언한다. 단, NOT NULL 제약조건의 경우에는 ADD 절이 아닌 MODIFY 절로 추가해야 한다.

다음은 각각 새로운 CHECK 제약조건과 UNIQUE 제약조건을 추가하는 예이다.

[예 4.10] 제약조건의 변경 - 제약조건의 추가

```
ALTER TABLE EMP
        ADD CONSTRAINT SALARY_MAX CHECK (SALARY >= 50000);

ALTER TABLE EMP
        ADD UNIQUE (ENAME, DEPTNO);
```

제약조건을 추가할 때 제약조건의 이름은 CHECK 제약조건의 예에서처럼 이름을 설정할 수도 있지만, 선택적으로 설정하지 않을 수 있다.

컬럼의 이름을 변경하면 기존에 컬럼의 이름을 사용하던 제약조건은 Tibero RDBMS 시스템에 의해 자동으로 변경된다. 예를 들어 [예 4.3]처럼 SQL 문장을 실행하면 ADDR 컬럼에 설정된 제약조건을 ADDRESS 컬럼에 자동으로 적용한다. 단, CHECK 제약조건의 경우 제대로 동작하지 않을 수 있으며, 사용자가 ALTER TABLE 문으로 제약조건을 다시 정의해야 한다.

제약조건의 제거

제약조건을 제거하기 위해서는 ALTER TABLE 문에 **DROP** 절을 삽입해야 한다. 기본 키, 유일 키 제약조건을 제외하고는 반드시 제약조건의 이름이 있어야 한다. 제약조건을 선언할 때 제약조건의 이름을 명시하지 않으면 Tibero RDBMS가 임의의 이름을 자동으로 생성해 준다.

제약조건의 이름을 설정하지 않은 경우 **USER_CONSTRAINTS** 뷰에서 해당 제약조건의 이름을 확인하여 이를 변경 또는 제거할 수 있다.

다음은 제약조건을 제거하는 예이다.

[예 4.11] 제약조건의 제거

```
ALTER TABLE EMP
        DROP PRIMARY KEY;
```

```
  ... 기본 키가 설정된 제약조건을 제거한다. ...

ALTER TABLE EMP
        DROP CONSTRAINT SALARY_MAX;
  ... 제약조건의 이름이 SALARY_MAX인 제약조건을 제거한다. ...
```

4.2.2. 제약조건의 상태

제약조건의 상태는 다음과 같이 두 가지로 나뉜다.

- ENABLE

  테이블에 삽입 또는 갱신되는 모두 로우에 적용된다.

  ENABLE은 아래와 같이 두 가지 옵션을 추가적으로 사용할 수 있다.

옵션	설명
VALIDATE	제약조건이 설정되지 않은 상태에서 많은 수의 로우가 새로 삽입되거나 갱신될 때 로우가 제약조건을 만족하는지를 확인하는 옵션이다. 가능하면 모든 로우를 한꺼번에 확인하는 것이 데이터베이스 성능 향상에 도움이 된다.
NOVALIDATE	기존에 저장된 테이블의 로우가 제약조건에 만족하지 않아도 되거나 또는 모든 로우가 제약조건에 만족하는 경우에만 사용하는 옵션이다. 테이블에 저장된 로우가 제약조건에 만족하는지 확인하지 않아도 되므로 데이터베이스 성능 향상에 도움이 된다. 단, 기본 키 또는 유일 키 제약조건은 내부적으로 사용하는 인덱스의 특성 상 NOVALIDATE 옵션을 사용한다고 해도 무조건 VALIDATE로 동작한다.

- DISABLE

  ENABLE과는 반대의 경우로 선언된 제약조건을 적용하지 않는다. 한꺼번에 많은 수의 로우를 테이블에 삽입하거나 갱신하는 경우, 제약조건을 DISABLE 상태로 하여 작업을 마친 후 제약조건을 다시 ENABLE 상태로 다시 설정하는 것이 데이터베이스 성능 향상에 도움이 된다.

  tbLoader 또는 tbImport 유틸리티나 배치 프로그램을 통해 많은 수의 로우를 삽입하거나 갱신할 수 있다. 테이블에 저장된 로우가 제약조건에 만족하는지를 확인하지 않아도 되므로 데이터베이스 성능 향상에 도움이 된다.

제약조건의 상태 변경

제약조건의 상태를 변경하기 위해서는 ALTER TABLE 문을 사용해야 한다.

다음은 제약조건의 상태를 변경하는 예이다.

- ENABLE 상태로 변경하는 경우

[예 4.12] 제약조건의 상태 변경 - ENABLE

```
ALTER TABLE EMP MODIFY CONSTRAINT EMP_UNIQUE ENABLE;
```

- DISABLE 상태로 변경하는 경우

[예 4.13] 제약조건의 상태 변경 - DISABLE

```
ALTER TABLE EMP MODIFY PRIMARY KEY DISABLE;
```

- NOVALIDATE로 추가한 제약조건을 다시 VALIDATE로 변경하는 경우

[예 4.14] 제약조건의 상태 변경 - VALIDATE

```
ALTER TABLE EMP MODIFY CONSTRAINT SALARY_MIN ENABLE NOVALIDATE;
```

4.2.3. 제약조건의 정보 조회

Tibero RDBMS에서는 제약조건의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
DBA_CONSTRAINTS	Tibero RDBMS 내의 모든 제약조건의 정보를 조회하는 뷰이다.
USER_CONSTRAINTS	현재 사용자에 속한 제약조건의 정보를 조회하는 뷰이다.
ALL_CONSTRAINTS	사용자가 접근 가능한 제약조건의 정보를 조회하는 뷰이다.
DBA_CONS_COLUMNS	Tibero RDBMS 내의 모든 제약조건에 적용된 컬럼 정보를 조회하는 뷰이다.
USER_CONS_COLUMNS	현재 사용자에 속한 제약조건에 적용된 컬럼 정보를 조회하는 뷰이다.
ALL_CONS_COLUMNS	사용자가 접근 가능한 제약조건에 적용된 컬럼 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

4.3. 디스크 블록

디스크 블록은 데이터를 저장하는 물리적인 최소 단위이며, 크기가 일정하다. Tibero RDBMS에서는 디스크 블록을 효율적으로 사용할 수 있도록 스키마 객체별로 파라미터를 제공한다. 스키마 객체의 특성에 따라 파라미터를 설정하면, 데이터베이스 성능의 향상과 저장 공간의 활용도를 높일 수 있다.

4.3.1. PCTFREE 파라미터

PCTFREE는 디스크 블록에 저장된 스키마 객체의 갱신에 대비하여 얼마만큼의 여유 공간을 남길 것인가를 설정하는 파라미터이다.

퍼센트 값으로 표현하며, 1에서 99 사이의 임의의 정수로 설정할 수 있다. 디스크 블록의 여유 공간이 PCTFREE 파라미터에 설정한 값 이하로 떨어질 때까지 계속 새로운 로우를 삽입한다. PCTFREE 파라미터에 설정한 값 이하인 경우에는 더 이상 새로운 로우를 삽입하지 않으며, 남은 공간은 기존 로우의 갱신에 대비하게 된다.

디스크 블록 내의 빈 공간이 PCTFREE 파라미터의 값보다 작아지면, 디스크 블록 내의 객체를 삭제한다. 빈 공간이 PCTFREE 파라미터의 값보다 커지더라도 바로 새로운 객체를 삽입하지는 않는다. 이후에 충분한 공간이 생겼을 때 디스크 블록에 삽입된다.

현재 디스크 블록에 저장된 객체가 갱신되어 크기가 증가할 가능성이 있는 경우, PCTFREE 파라미터의 값을 크게 설정해야 한다. 이때, PCTFREE 값이 작은 경우와 비교하여 하나의 디스크 블록에 저장되는 객체의 수가 상대적으로 적어지므로, 같은 수의 객체를 저장하는 데에 더 많은 디스크 블록을 필요로 하게 된다. 이러한 점은 데이터베이스 성능 저하의 원인이 될 수 있다.

반면에, 하나의 객체를 여러 디스크 블록에 저장해야 하는 가능성이 적어지므로 성능 향상에 도움이 될 수도 있다. 따라서, PCTFREE 파라미터의 값은 데이터베이스를 운용하며 적절하게 설정해야 하며, 객체의 갱신이 많이 발생하는 경우에는 PCTFREE 파라미터의 값을 크게 잡는 것이 좋다. 디폴트로 설정될 PCTFREE 파라미터의 값은 10%이다.

디스크 블록을 액세스하는 트랜잭션 내의 갱신 연산이 객체의 크기를 증가시키지 않거나 트랜잭션이 읽기 연산만으로 이루어져 있다면, PCTFREE 파라미터의 값을 매우 작게 설정할 수 있다. 예를 들어, PCTFREE 값을 5로 설정할 수 있다.

4.3.2. INITRANS 파라미터

하나의 디스크 블록은 동시에 여러 트랜잭션에 접근할 수 있으며, 디스크 블록에 포함되어 있는 각 로우는 자신을 마지막으로 생성, 갱신, 삭제한 트랜잭션의 정보를 가지고 있다. 이러한 트랜잭션의 데이터를 디스크 블록 내의 한 곳에 모아두는 것을 **트랜잭션 엔트리 리스트**(transaction entry list)라고 한다.

트랜잭션 엔트리 리스트는 디스크 블록의 헤더에 포함되며, 디스크 블록이 생성될 때 최초 크기는 **INITRANS 파라미터**에 설정된 값이 된다. 리스트 내의 트랜잭션 엔트리는 트랜잭션이 커밋되면 다른 트랜잭션에 의해 다시 사용할 수 있다.

예를 들어, 더 많은 트랜잭션 엔트리가 필요한 경우에는 전체 리스트의 크기(트랜잭션 엔트리의 개수)를 하나씩 증가시킨다. 이 때 디스크 블록의 크기에 따라 트랜잭션 엔트리의 최대 개수가 정해진다. 즉 디스크 블록의 크기가 커질수록 트랜잭션 엔트리의 개수가 증가하게 된다. 단, 최대 255개를 초과하지 않아야 한다.

트랜잭션 엔트리 개수가 최대 개수에 도달하면 더 이상 리스트의 크기를 증가시키지 않으며, 트랜잭션의 실행을 중지하고 대기한다. 트랜잭션은 다른 트랜잭션이 끝나고, 기존의 트랜잭션 엔트리를 다시 사용할 수 있게 되면 실행을 계속한다.

트랜잭션 엔트리 리스트를 증가시키는 작업은 처리 비용이 필요하므로, 자주 하지 않는 것이 좋다. 따라서, 하나의 디스크 블록을 여러 트랜잭션에서 액세스할 가능성이 높은 경우에는 INITRANS 파라미터의 값을 처음부터 높게 설정해 주는 것이 중요하다.

INITRANS 파라미터를 설정하지 않으면 기본값은 2이다.

4.3.3. 파라미터의 설정

PCTFREE와 INITRANS 파라미터는 스키마 객체 단위로 설정할 수 있다. 스키마 객체는 항상 같은 파라미터의 값을 갖는다. 파라미터는 스키마 객체를 생성하거나 변경할 때 설정할 수 있다.

다음은 테이블 EMP를 생성할 때 파라미터를 설정하는 예이다.

```
CREATE TABLE EMP (
                  ENAME    VARCHAR(16) NOT NULL,
                  ADDR     VARCHAR(24),
                  SALARY   INT,
                  DEPTNO   INT
                )
                PCTFREE  5
                INITRANS 5;
```

다음은 파라미터를 **ALTER TABLE** 문을 이용하여 변경하는 예이다.

```
ALTER TABLE EMP PCTFREE 10;
```

인덱스를 생성할 때에는 INITRANS 파라미터의 값만 설정할 수 있다.

다음의 SQL 문장은 테이블 EMP의 인덱스를 생성할 때 파라미터를 설정하는 예이다.

```
CREATE INDEX EMP_DEPTNO_IDX ON EMP (DEPTNO) INITRANS 5;
```

디스크 블록의 파라미터의 값을 변경하더라도 디스크 블록에는 바로 반영되지는 않는다. 파라미터별로 디스크 블록에 반영되는 경우는 다음과 같다.

파라미터	설명
PCTFREE	새로 객체를 삽입하고 삭제하는 경우 기존의 디스크 블록에 반영된다.
INITRANS	기존의 디스크 블록에는 반영되지 않으며, 새로 할당된 디스크 블록에만 반영된다.

4.4. 인덱스

인덱스는 테이블에서 원하는 데이터를 효율적으로 검색하기 위해 사용하는 데이터 구조이다. 인덱스는 테이블과는 다른 스키마 객체이므로 독립적으로 생성, 변경, 제거, 저장할 수 있다.

다음은 인덱스의 종류이다.

- 단일 컬럼 인덱스(Single Index)

  하나의 컬럼으로 구성된 인덱스이다.

- 복합 컬럼 인덱스(Concatenated Index)

  하나 이상의 컬럼으로 구성된 인덱스이다.

- 유일 인덱스(Unique Index)

  테이블에서 유일한 값을 가진 컬럼으로 구성된 인덱스이다.

- 비유일 인덱스(Non-Unique Index)

  중복되는 값을 인정하는 컬럼으로 구성된 인덱스이다.

4.4.1. 인덱스의 생성, 제거

Tibero RDBMS는 기본 키, 유일 키 그리고 외래 키에 제약조건이 설정된 컬럼을 제외하고는 임의의 컬럼에 인덱스를 생성하거나 제거할 수 있다. 인덱스의 이름은 사용자가 별도로 지정하지 않으면 디폴트로 지정된 제약조건의 이름과 동일하게 생성된다.

인덱스의 생성

인덱스는 다음 같은 경우에 따라 생성, 제거 요건이 다르다.

- 현재 사용자가 자신의 스키마에 속한 테이블에 인덱스를 생성하고 제거하는 경우 **CREATE INDEX 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

- 다른 사용자의 스키마에 속한 테이블에 인덱스를 생성하고 제거하는 경우 **CREATE ANY INDEX 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

인덱스를 생성하기 위해서는 **CREATE INDEX** 문을 사용해야 한다.

다음은 인덱스를 생성할 때 포함되는 구성요소이다.

구성요소	설명
인덱스의 이름	생성할 인덱스의 이름을 설정한다. 이 구성요소는 테이블을 생성할 때 반드시 포함되어야 한다.
테이블의 이름	해당 컬럼이 속한 테이블의 이름을 설정한다. 이 구성요소는 테이블을 생성할 때 반드시 포함되어야 한다.
Unique	Unique 인덱스 여부를 설정한다. 이 구성요소는 인덱스를 생성할 때 선택적으로 사용할 수 있다.
컬럼 정의, 정렬 방향	해당 컬럼을 정의하고 정렬 방향을 설정한다. 이 구성요소는 인덱스를 생성할 때 선택적으로 사용할 수 있다.
테이블스페이스	인덱스가 저장될 테이블스페이스를 설정한다. 이 구성요소는 인덱스를 생성할 때 선택적으로 사용할 수 있다.
디스크 블록의 파라미터	INITRANS 파라미터를 설정할 수 있다. 이 구성요소는 인덱스를 생성할 때 선택적으로 사용할 수 있다.
파티션 인덱스	파티션 인덱스를 설정할 수 있다. 이 구성요소는 인덱스를 생성할 때 선택적으로 사용할 수 있다.

다음은 테이블 EMP의 컬럼 DEPTNO에 인덱스를 생성하는 예이다. 인덱스의 이름은 **EMP_FK**로 설정한다.

[예 4.15] 인덱스의 생성

```
CREATE INDEX EMP_FK ON EMP (DEPTNO);
```

인덱스의 제거

인덱스를 제거하기 위해서는 **DROP INDEX** 문을 사용해야 한다.

인덱스는 테이블처럼 독립적인 스키마 객체이므로, 인덱스를 제거하더라도 테이블의 데이터에는 영향을 미치지 않는다. 단, 인덱스를 제거하게 되면 해당 컬럼의 데이터를 조회할 때 이전과 다르게 조회 속도가 느려질 수도 있다. 인덱스가 더 이상 필요하지 않는 경우에는 제거하는 것이 데이터베이스 성능 향상에 도움이 된다.

다음은[예 4.15]에서 생성한 인덱스 EMP_FK를 제거하는 예이다.

[예 4.16] 인덱스의 제거

```
DROP INDEX EMP_FK;
```

4.4.2. 인덱스의 효율적인 관리

Tibero RDBMS에서는 인덱스의 기본 구조로 B-트리를 제공한다. 이를 이용하여 단일 키 검색(single key search), 범위 검색(range search), 복합 키 검색(composite key search)을 수행할 수 있다.

인덱스의 검색 유형

인덱스를 이용하여 테이블의 로우를 검색하는 방법은 다음과 같다.

- 단일 키 검색

  하나의 키를 갖는 로우를 검색하는 방법이다. 인덱스가 유일 인덱스인 경우에는 하나의 키를 갖는 한 개의 로우만 검색되며, 비유일 인덱스의 경우에는 여러 개의 로우가 검색된다.

- 범위 검색

  검색 범위에 포함되는 키를 갖는 로우를 모두 검색하는 방법이다.

- 복합 키 검색

  서로 다른 두 개 이상의 컬럼을 하나의 키로 조합하여 검색하는 방법이다.

  다음은 복합 키를 이용하여 검색하는 예이다.

  [예 4.17] 복합 키 검색

  ```
  SELECT * FROM EMP
           WHERE DEPTNO = 5 AND ADDR = 'Seoul';
  ```

  위의 예제에서 컬럼 DEPTNO와 컬럼 ADDR가 별도의 인덱스로 생성되어 있다면, 원하는 로우를 검색하기 위해 먼저 **DEPTNO = 5**인 조건을 만족하는 로우와 **ADDR = '서울'**인 조건을 만족하는 로우를 각각 검색하게 된다. 그 다음 두 조건에서 공통으로 포함하는 모든 로우를 출력하게 된다.

  컬럼 DEPTNO와 컬럼 ADDR를 복합 키 인덱스로 생성하는 방법은 DEPTNO+ADDR 또는 ADDR+DEPTNO 형태로 조합하면 된다. 복합 키 인덱스를 생성하면, 한꺼번에 원하는 로우를 검색할 수 있어 효율적이다.

인덱스의 효율적인 관리 지침

인덱스를 효율적으로 관리하기 위한 지침은 다음과 같다.

- 인덱스가 필요한 테이블과 컬럼에만 생성한다.

  인덱스는 데이터의 저장 공간과 데이터베이스 성능에 영향을 미친다. 인덱스를 생성하면 데이터를 저장하기 위한 별도의 공간이 필요하며, 테이블에 로우가 삽입, 변경, 제거될 때마다 인덱스도 함께 갱신된다.

- 기본 키가 적용된 컬럼과 함께 조인 연산의 대상이 되는 컬럼의 경우 인덱스를 생성한다는 것은 컬럼에 정렬을 수행한 결과를 저장한다는 의미이다.

  Tibero RDBMS에서는 정렬된 컬럼 간의 조인 연산을 더욱 효율적으로 수행한다. 이러한 결과는 조인의 대상이 되는 테이블이 컸을 때 효과적이다.

- WHERE 절의 검색 조건에 포함되는 빈도가 높으며, 검색 결과가 전체 테이블의 10% 이하인 컬럼의 경우 단일 키 검색의 경우 인덱스가 없으면 전체 테이블을 액세스한다.

  반면에 인덱스가 있으면 하나의 로우만 액세스하므로 검색 성능을 향상시킬 수 있다.

- 복합 키 인덱스를 생성할 때 컬럼 값의 순서에 유의한다.

  검색 조건에 포함되는 빈도가 높은 컬럼부터 정렬한다. 예를 들어, 컬럼 C1과 컬럼 C2에 복합 키 인덱스를 생성할 때, 인덱스 키를 만드는 방법은 (C1, C2)와 (C2, C1)의 두 가지가 있다.

  컬럼 C1에 대한 검색이 컬럼 C2에 대한 검색보다 더 빈번하게 일어난다면 (C1, C2) 값을 이용하여 복합 키 인덱스를 생성하는 것이 유리하다. 그 이유는 인덱스 내에서 같은 C1 값을 갖는 키들이 모여 있기 때문에 검색을 위해 액세스해야 할 디스크 블록의 개수가 적기 때문이다. (C1, C2) 값으로 인덱스를 생성했다면 컬럼 C2에 대한 검색은 거의 사용할 수 없다.

- 사용 빈도가 낮거나 필요 없는 인덱스는 제거한다.

- 하나의 테이블에 많은 수의 인덱스는 생성하지 않는다.

- 시스템의 성능 향상을 위해 인덱스와 테이블은 서로 다른 디스크에 저장한다.

4.4.3. 인덱스의 정보 조회

Tibero RDBMS에서는 인덱스의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
DBA_INDEXES	Tibero RDBMS 내의 모든 인덱스의 정보를 조회하는 뷰이다.

정적 뷰	설명
USER_INDEXES	현재 사용자에 속한 인덱스의 정보를 조회하는 뷰이다.
ALL_INDEXES	사용자가 접근 가능한 인덱스의 정보를 조회하는 뷰이다.
DBA_IDX_COLUMNS	Tibero RDBMS 내의 모든 인덱스에 적용된 컬럼의 정보를 조회하는 뷰이다.
USER_IDX_COLUMNS	현재 사용자에 속한 인덱스에 적용된 컬럼의 정보를 조회하는 뷰이다.
ALL_IDX_COLUMNS	사용자가 접근 가능한 인덱스에 적용된 컬럼의 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

4.5. 뷰

뷰는 SELECT 문으로 표현되는 질의에 이름을 부여한 가상 테이블이다. SQL 문장 내에서 테이블과 동일하게 사용된다. 단, 실제 데이터를 포함하는 스키마 객체는 아니며, 다른 스키마 객체를 통해 정의된다.

4.5.1. 뷰의 생성, 변경, 제거

본 절에서는 뷰를 생성, 변경, 제거하는 방법을 설명한다.

뷰의 생성

뷰는 다음 같은 경우에 따라 생성 요건이 다르다.

- 현재 사용자가 자신의 스키마에 속한 뷰를 생성하는 경우 **CREATE VIEW 문**을 사용할 수 있는 시스템 특권이 있어야 한다.
- 다른 사용자의 스키마에 있는 뷰를 생성하는 경우 **CREATE ANY VIEW 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

뷰를 생성하기 위해서는 **CREATE VIEW** 문을 사용해야 한다. 모든 기반 객체 (base objects)의 액세스 권한을 갖고 있어야 하며, 기반 테이블의 수에 상관 없이 액세스 권한이 있어야 한다.

예를 들면 다음과 같다.

[예 4.18] 뷰의 생성

```
CREATE VIEW MANAGER AS
       SELECT * FROM EMP
       WHERE DEPTNO = 1;
```

```
CREATE VIEW EMP_DEPT AS
       SELECT E.EMPNO, E.ENAME, E.SALARY, D.DEPTNO, D.LOC
       FROM EMP E, DEPT D
       WHERE E.DEPTNO = D.DEPTNO;
```

뷰를 이용하여 수행할 수 있는 연산은 기반 테이블에 대한 뷰 정의자가 수행할 수 있는 연산의 교집합이다. 예를 들어, 뷰 EMP_DEPT를 정의한 사용자가 테이블 EMP에 대하여 삽입, 제거 연산이 가능하고 테이블 DEPT에 대하여 삽입, 갱신 연산이 가능하다면, 뷰 EMP_DEPT에 대해서는 삽입 연산만 할 수 있다.

뷰는 테이블과 같이 액세스 권한을 다른 사용자에게 부여할 수 있다. 단, 뷰를 정의한 기반 객체에 대한 GRANT OPTION 또는 ADMIN OPTION과 함께 액세스 권한을 부여 받아야 한다. 뷰에 대한 액세스 권한을 부여 받은 사용자는 그 뷰를 정의한 기반 객체에 직접 액세스할 수 있는 권한이 없어도 그 뷰를 통하여 액세스할 수 있다. 단, 수행할 연산에 대한 권한은 뷰의 정의자가 가지고 있어야 한다.

다음은 뷰를 생성하는 예이다.

```
CREATE VIEW V_PROD AS
       SELECT PROD_ID, PROD_NAME,PROD_COST
       FROM PRODUCT
       WHERE PROD_ID= 100001;
```

뷰의 변경

뷰 또는 기반 객체의 변경으로 인해 사용할 수 없게 된 뷰를 다시 사용하기 위해서는 **CREATE OR REPLACE VIEW** 문을 사용해야 한다. 단, 뷰를 생성하고 제거할 수 있는 권한이 있어야 한다.

다음은 뷰를 변경하는 예이다.

[예 4.19] 뷰의 변경

```
CREATE OR REPLACE VIEW MANAGER AS
       SELECT * FROM EMP
       WHERE DEPTNO = 2;
```

위와 같은 SQL 문장을 실행하면 다른 사용자에게 부여한 뷰 MANAGER의 권한이 그대로 남아 있게 된다. 반면에 DROP VIEW와 CREATE VIEW 문을 연속으로 사용하면 뷰 MANAGER의 권한은 없어진다.

뷰의 제거

뷰는 다음 같은 경우에 따라 제거 요건이 다르다.

- 현재 사용자가 자신의 스키마에 속한 뷰를 제거하는 경우 **DROP VIEW 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

- 다른 사용자의 스키마에 속한 뷰를 제거하는 경우 **DROP ANY VIEW 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

뷰를 제거하기 위해서는 **DROP VIEW** 문을 사용해야 한다.

다음은 뷰를 제거하는 예이다.

[예 4.20] 뷰의 제거

```
DROP VIEW EMP_DEPT;
```

4.5.2. 뷰의 갱신 가능성

모든 뷰는 SQL 질의가 가능하나 삽입, 갱신, 제거가 불가능한 뷰가 존재한다. 또한, 대상 컬럼에 따라 삽입, 갱신이 가능하거나 불가능한 뷰도 존재한다.

삽입, 갱신, 제거를 수행할 수 있는 뷰를 **갱신 가능한 뷰** (updatable view)라고 한다.

다음의 뷰는 갱신 가능한 뷰가 아니다.

- 두 개 이상의 테이블에 대한 집합 연산을 수행하여 생성한 뷰
- 그룹화(grouping)와 집단 함수(aggregate function)를 이용하여 생성한 뷰
- 다:다(many-to-many) 관계를 갖는 컬럼 간에 조인을 수행하여 생성한 뷰
- WITH READ ONLY 절을 사용하여 생성한 뷰

조인 뷰

두 개 이상의 테이블을 조인하여 생성한 뷰를 **조인 뷰** 라고 한다.

조인 뷰를 갱신하는 경우, INSERT, UPDATE, DELETE 연산은 뷰를 구성하는 기반 테이블 중 키 보존 테이블(key-preserved table)의 특성을 만족하는 테이블에 한해서만 수행한다.

키 보존 테이블 이란 기본 키를 가지고 있는 테이블이다. 하나의 뷰에서 키 보존 테이블이 여러 개가 존재하는 경우, 하나의 DML 문장은 이 중 하나의 테이블에 대해서만 연산을 수행한다. 다시 말해, 삽입, 갱신, 제거가 일어나는 컬럼이 이 키 보존 테이블의 컬럼이어야 한다는 뜻이다.

DELETE 연산을 하는 경우, 컬럼을 명시적으로 지정하지 않으므로 키 보존 테이블이 여러 개가 존재하는 경우에 제거 연산이 일어나는 테이블은 (FROM 절에서 명시된 순서) 맨 처음이 키 보존 테이블이 된다.

[예 4.18]에서 생성한 뷰 EMP_DEPT는 테이블 EMP의 외래 키 컬럼 DEPTNO와 테이블 DEPT의 기본 키 컬럼 DEPTNO를 조인하여 얻어진 뷰이다. 테이블 EMP 내에 같은 DEPTNO 값을 갖는 로우가 여러 개일 수 있으므로, 뷰 EMP_DEPT의 질의 결과의 로우 중에는 같은 DEPTNO 값이 여러 로우에 걸쳐 나타날 수 있다.

하지만, 컬럼 EMPNO는 테이블 EMP의 기본 키 컬럼이므로, 뷰 EMP_DEPT의 질의 결과의 로우 중 하나의 EMPNO 값이 여러 로우에 중복되어 나타날 수 없다. 따라서, 뷰 EMP_DEPT의 키 컬럼은 EMPNO이며 같은 컬럼을 키 컬럼으로 갖는 테이블 EMP가 키 보존 테이블이다.

일반적으로, 기본 키와 외래 키의 컬럼처럼 일:다 (one-to-many) 관계를 갖는 조인 컬럼에 의해 생성된 조인 뷰에서 다 (many) 측의 테이블, 외래 키 조인 컬럼을 갖는 테이블이 키 보존 테이블이 된다. 셋 이상의 기반 테이블을 정의한 뷰에서도 마찬가지이다.

다음은 뷰 EMP_DEPT에 INSERT, UPDATE, DELETE 연산을 수행하는 예이다. 세 문장 모두 테이블 EMP의 컬럼에 삽입, 갱신, 제거 연산을 수행한다.

```
INSERT INTO EMP_DEPT (EMPNO, ENAME, SALARY)
        VALUES (1562, 'Brown', 35000);

UPDATE EMP_DEPT SET SALARY = SALARY * 1.1
        WHERE DEPTNO = 5;

DELETE FROM EMP_DEPT
        WHERE ENAME = 'Scott';
```

INSERT 문에서 값이 지정되지 않은 컬럼은 NULL 값이 삽입되거나 기본 키의 값으로 채워진다. 단, NOT NULL 제약조건이 설정된 경우 에러를 반환한다.

뷰를 생성할 때 **WITH CHECK OPTION** 제약을 사용할 수도 있는데, 이 때 갱신, 삽입, 제거가 가능한 조건이 약간 변경된다. WITH CHECK OPTION 제약이 명시된 뷰를 갱신할 때는, 갱신되는 로우가 갱신 후에도 뷰에 의해 SELECT 되는 경우에만 그 로우의 갱신이 허용된다. 조인 뷰의 경우는 조인된 컬럼 또는 반복되는 컬럼은 무조건 갱신이 불가능하다는 제약조건이 추가된다.

삽입의 경우도 갱신과 마찬가지로 삽입하는 로우가 뷰에 의해 SELECT 되어야 한다. 단, 기반 테이블이 하나일 경우에만 가능하며 조인 뷰의 경우에는 삽입할 수 없다.

제거의 경우는 제거의 대상이 되는 키 보존 테이블이 반복되지 않으면 가능하다.

4.5.3. 뷰의 정보 조회

Tibero RDBMS에서는 뷰의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
DBA_VIEWS	Tibero RDBMS 내의 모든 뷰의 정보를 조회하는 뷰이다.
USER_VIEWS	현재 사용자에 속한 뷰의 정보를 조회하는 뷰이다.
ALL_VIEWS	사용자가 접근 가능한 뷰의 정보를 조회하는 뷰이다.
DBA_UPDATABLE_COLUMNS	Tibero RDBMS 내의 모든 뷰에 속한 컬럼의 갱신 가능성 정보를 조회하는 뷰이다.

정적 뷰	설명
USER_UPDATABLE_COLUMNS	현재 사용자에 속한 뷰에 속한 컬럼의 정보를 조회하는 뷰이다.
ALL_UPDATABLE_COLUMNS	사용자가 접근 가능한 뷰에 속한 컬럼의 갱신 가능성 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

4.6. 시퀀스

시퀀스는 순차적으로 부여하는 고유번호이다. 주로 새로운 데이터에 유일한 고유번호를 자동으로 부여할 때 사용한다.

4.6.1. 시퀀스의 생성, 변경, 제거

본 절에서는 시퀀스를 생성, 변경, 제거하는 방법을 설명한다.

시퀀스는 여러 개의 트랜잭션이 서로 겹치지 않는 고유번호를 만들어낼 때 사용된다.

시퀀스를 사용하지 않으면, 고유번호를 만들어내기 위해서 마지막으로 사용된 번호를 기억하는 테이블을 만들고, 각 트랜잭션이 해당 값을 읽어서 하나씩 증가시켜야 한다. 이러한 방법을 사용하면 고유번호를 생성하는 모든 트랜잭션 사이에 잠금 (Lock)으로 인한 데이터 충돌이 발생하게 된다. 이로 인해 데이터베이스 성능이 저하되는 원인이 될 수 있다.

시퀀스의 생성

시퀀스는 다음 같은 경우에 따라 생성 요건이 다르다.

- 현재 사용자가 자신의 스키마에 시퀀스를 생성하는 경우 **CREATE SEQUENCE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.
- 다른 사용자의 스키마에 시퀀스를 생성하는 경우 **CREATE ANY SEQUENCE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

시퀀스를 생성하기 위해서는 **CREATE SEQUENCE** 문을 사용해야 한다.

다음은 시퀀스를 생성할 때 포함되는 구성요소이다.

구성요소	설명
시퀀스의 이름	시퀀스의 이름을 설정한다.

구성요소	설명
	이 구성요소는 시퀀스를 생성할 때 반드시 포함되어야 한다.
MINVALUE	시퀀스가 생성할 수 있는 최솟값이다.
MAXVALUE	시퀀스가 생성할 수 있는 최댓값이다.
INCREMENT BY	시퀀스의 값을 사용할 때마다 얼마씩 증가 또는 감소할지를 설정한다.
CACHE	데이터베이스 성능 향상을 위해 내부적으로 메모리에 값을 캐시한다.
START WITH	시퀀스를 가장 처음 사용할 때 생성되는 값을 설정한다. 이 값을 설정하지 않으면 최솟값 (감소할 경우에는 최댓값)으로 정의된다.
NOCYCLE	시퀀스는 기본적으로 NOCYCLE로 정의되어 있다. 최댓값 (값이 감소할 경우에는 최솟값) 에 도달하면 ALTER SEQUENCE 문을 사용하지 않는 한 새로운 값을 생성할 수 없다.
CYCLE	최댓값(최솟값)에 도달하면 자동으로 다음 값은 최솟값(최대값)으로 순환한다.

시퀀스는 데이터베이스 성능 향상을 위해 내부적으로 메모리에 값을 캐시한다. MAX_SEQ_BUFFER 파라미터에 캐시의 크기를 지정하며, 기본값은 20이다.

시스템이 정상적으로 종료될 경우 캐시에 존재하지만 아직 실제로 쓰이지 않은 값은 디스크에 저장되어 다음 번 Tibero RDBMS가 기동할 때 사용할 수 있다. 하지만 시스템이 비정상적으로 종료될 경우 캐시에 존재하던 값은 모두 사용된 것으로 간주되며, 캐시의 최대 크기만큼 시퀀스의 값을 건너뛸 수 있다.

이러한 경우가 발생하지 않으려면 시퀀스를 **NOCACHE**로 선언해야 한다. 하지만, 시퀀스를 사용할 때마다 디스크 액세스가 일어나므로 데이터베이스 성능이 저하된다. 특수한 상황이 아니면 NOCACHE를 사용하지 않기를 권장한다.

다음은 시퀀스를 생성하는 예이다.

[예 4.21] 시퀀스의 생성

```
CREATE SEQUENCE NEW_ID
       MINVALUE 1000
       MAXVALUE 9999
       INCREMENT BY 10
       CACHE 100
       NOCYCLE;
```

다음은 시퀀스의 값을 사용하는 예이다.

```
CREATE TABLE EMP_ID (ID NUMBER, NAME VARCHAR(30));

INSERT INTO EMP_ID VALUES(NEW_ID.NEXTVAL, 'Peter');
```

시퀀스에 일단 사용된 값은 해당 트랜잭션이 롤백되거나 시스템이 비정상적으로 종료되어도 재사용되지 않는다.

시퀀스의 변경

시퀀스는 다음 같은 경우에 따라 변경 요건이 다르다.

- 현재 사용자가 자신의 스키마에 속한 시퀀스를 변경하는 경우 **ALTER SEQUENCE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

- 다른 사용자의 스키마에 속한 시퀀스를 변경하는 경우 **ALTER ANY SEQUENCE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

시퀀스를 변경하기 위해서는 **ALTER SEQUENCE** 문을 사용해야 한다.

다음은 시퀀스를 변경하는 예이다.

[예 4.22] 시퀀스의 변경

```
ALTER SEQUENCE NEW_ID
      MAXVALUE 99999
      INCREMENT BY 1
      CACHE 200;
```

Tibero RDBMS는 시퀀스를 사용할 때, 사용자가 마지막으로 NEXTVAL로 적용한 값이 아니라 다음 사용자가 NEXTVAL을 부를 때 가져갈 값을 저장한다. 따라서 ALTER SEQUENCE 문을 실행했을 때의 결과가 타 DBMS와 다를 수 있다.

다음은 내부적으로 **다음 번에 받아갈 값 = 10**을 기억하고 있으며, 마지막 결과가 10이 되는 시퀀스의 예이다.

```
SQL> CREATE SEQUENCE S1 START WITH 10 INCREMENT BY 1;
created

SQL> ALTER SEQUENCE S1 INCREMENT BY 5;
altered

SQL> SELECT S1.NEXTVAL FROM DUAL;
   NEXTVAL
----------
        10

1 selected
```

시퀀스의 제거

시퀀스는 다음 같은 경우에 따라 제거 요건이 다르다.

- 현재 사용자가 자신의 스키마에 속한 시퀀스를 제거하는 경우 **DROP SEQUENCE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.
- 다른 사용자의 스키마에 속한 시퀀스를 제거하는 경우 **DROP ANY SEQUENCE 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

시퀀스를 제거하기 위해서는 **DROP SEQUENCE** 문을 사용해야 한다.

다음은 시퀀스를 제거하는 예이다.

[예 4.23] 시퀀스의 제거

```
DROP SEQUENCE NEW_ID;
```

4.6.2. 시퀀스의 정보 조회

Tibero RDBMS에서는 시퀀스의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
DBA_SEQUENCES	Tibero RDBMS 내의 모든 시퀀스의 정보를 조회하는 뷰이다.
USER_SEQUENCES	현재 사용자에 속한 시퀀스의 정보를 조회하는 뷰이다.
ALL_SEQUENCES	사용자가 접근 가능한 시퀀스의 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

4.7. 동의어

동의어는 스키마 객체의 별칭(Alias)이다. 단, 실제 데이터를 포함하는 스키마 객체는 아니며, 다른 스키마 객체를 통해 정의된다.

4.7.1. 동의어의 생성, 제거

본 절에서는 동의어를 생성, 제거하는 방법을 설명한다.

동의어의 생성

동의어는 다음 같은 경우에 따라 생성 요건이 다르다.

- 현재 사용자가 자신의 스키마에 동의어를 생성하는 경우 **CREATE SYNONYM 문**을 사용할 수 있는 시스템 특권이 있어야 한다.
- 다른 사용자의 스키마에 동의어를 생성하는 경우 **CREATE ANY SYNONYM 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

동의어를 생성하기 위해서는 **CREATE SYNONYM** 문을 사용해야 한다.

다음은 동의어를 생성할 때 포함되는 구성요소이다.

구성요소	설명
동의어의 이름	동의어의 이름을 설정한다. 이 구성요소는 동의어를 생성할 때 반드시 포함되어야 한다.
테이블의 이름	동의어를 적용할 테이블의 이름을 설정한다. 이 구성요소는 동의어를 생성할 때 반드시 포함되어야 한다.

다음은 동의어를 생성하는 예이다.

[예 4.24] 동의어의 생성

```
CREATE SYNONYM T1 FOR U1.EMP;
```

동의어의 제거

정의된 동의어를 변경하기 위해서는 먼저 동의어를 제거하고 다시 생성해야 한다.

동의어는 다음 같은 경우에 따라 제거 요건이 다르다.

- 현재 사용자가 자신의 스키마에 속한 동의어를 제거하는 경우 **DROP SYNONYM 문**을 사용할 수 있는 시스템 특권이 있어야 한다.
- 다른 사용자의 스키마에 속한 동의어를 제거하는 경우 **DROP ANY SYNONYM 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

동의를 제거하기 위해서는 **DROP SYNONYM** 문을 사용해야 한다.

다음은 동의어를 제거하는 예이다.

[예 4.25] 동의어의 제거

```
DROP SYNONYM T1;
```

4.7.2. 공용 동의어의 생성, 제거

본 절에서는 공용 동의어를 생성, 제거하는 방법을 설명한다.

공용 동의어의 생성

동의어를 모든 사용자가 액세스할 수 있도록 생성할 수 있다. 이를 공용 동의어 (PUBLIC SYNONYM)라고 한다. 공용 동의어는 Tibero RDBMS에서 정의하고 있는 PUBLIC 이라는 특수한 사용자가 소유하는 동의어이다.

공용 동의어를 생성하기 위해서는 **CREATE PUBLIC SYNONYM** 문을 사용해야 한다.

다음은 공용 동의어를 생성할 때 포함되는 구성요소이다.

구성요소	설명
공용 동의어의 이름	공용 동의어의 이름을 설정한다. 이 구성요소는 공용 동의어를 생성할 때 반드시 포함되어야 한다.
테이블의 이름	공용 동의어를 적용할 테이블의 이름을 설정한다. 이 구성요소는 공용 동의어를 생성할 때 반드시 포함되어야 한다.

다음은 공용 동의어를 생성하는 예이다.

[예 4.26] 공용 동의어의 생성

```
CREATE PUBLIC SYNONYM PUB_T1 FOR U1.EMP;
```

동의어는 객체 참조 방법의 편의성과 투명성(transparency)을 제공한다. 예를 들어, 현재 사용자가 아닌 다른 사용자가 U1이 소유한 테이블 EMP를 접근하려고 하면 매번 U1.EMP라고 입력해야 한다. 하지만, 공용 동의어를 정의하면 PUB_T1만 입력하면 된다.

예를 들어, 다음의 두 SQL 문장은 같은 결과를 반환한다.

```
SELECT EMPNO, ENAME, ADDR FROM T1;

SELECT EMPNO, ENAME, ADDR FROM U1.EMP;
```

하나의 테이블을 액세스하는 애플리케이션 프로그램에서 다른 테이블을 액세스하는 동의어를 사용하는 경우, 프로그램 내에서 액세스하는 테이블의 이름을 모두 변경하는 대신 정의된 동의어를 변경하는 것으로도 충분하다.

공용 동의어의 제거

공용 동의어를 제거하려면 **DROP PUBLIC SYNONYM** 문을 사용해야 한다. 단, DROP PUBLIC SYNONYM 시스템 특권이 있어야 한다.

다음은 공용 동의어를 제거하는 예이다.

[예 4.27] 공용 동의어의 제거

```
DROP PUBLIC SYNONYM PUB_T1;
```

4.7.3. 동의어의 정보 조회

Tibero RDBMS에서는 동의어의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
DBA_SYNONYMS	Tibero RDBMS 내의 모든 동의어의 정보를 조회하는 뷰이다.
USER_SYNONYMS	현재 사용자에 속한 동의어의 정보를 조회하는 뷰이다.
ALL_SYNONYMS	사용자가 접근 가능한 동의어의 정보를 조회하는 뷰이다.
PUBLICSYN	모든 공용 동의어의 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

4.8. 트리거

트리거는 테이블의 로우를 삽입, 변경, 삭제할 때 자동으로 수행되도록 미리 지정해 놓은 PSM(Persistent Store Module) 프로시저이다. 제약조건으로 표현하기 어려운 데이터베이스의 논리적 조건을 표현할 때 사용한다. 예를 들어, 사용자 계정별로 테이블에 삽입할 수 있는 값의 범위를 다르게 제한하고 싶을 때 트리거를 사용할 수 있다.

4.8.1. 트리거의 생성, 제거

본 절에서는 트리거의 생성, 제거하는 방법을 설명한다.

트리거의 생성

트리거는 다음 같은 경우에 따라 생성 요건이 다르다.

- 현재 사용자가 자신의 스키마에 속한 트리거를 생성하는 경우 **CREATE TRIGGER 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

- 다른 사용자의 스키마에 속한 트리거를 생성하는 경우 **CREATE ANY TRIGGER 문**을 사용할 수 있는 시스템 특권이 있어야 한다.

트리거를 생성하기 위해서는 **CREATE TRIGGER** 문을 사용해야 한다. 트리거는 삽입, 변경, 삭제 연산을 수행하기 직전이나 직후에 수행할 수 있다.

다음은 EMP 테이블에 새로운 로우를 삽입한 직후에 트리거를 수행하는 예이다.

[예 4.28] 트리거의 생성

```
CREATE TRIGGER TRG1
        AFTER INSERT ON EMP
        FOR EACH ROW
        WHEN (TRUE)
        BEGIN
                --- PSM BLOCK ---
        END;
```

생성된 트리거는 트리거를 생성한 사용자의 권한을 가지고 동작한다.

참고

트리거에 대한 자세한 내용은 "Tibero RDBMS SQL 참조 안내서"를 참고한다.

트리거의 제거

트리거를 제거하기 위해서는 **DROP TRIGGER** 문을 사용해야 한다.

다음은 트리거를 제거하는 예이다.

[예 4.29] 트리거의 제거

```
DROP TRIGGER TRG1;
```

4.9. 파티션

테이블의 크기가 점점 커지고 많은 트랜잭션이 동시에 액세스하는 경우, 운영체제는 빈번한 입출력과 잠금(Lock) 현상이 발생하게 된다. 이러한 현상은 데이터베이스 성능이 저하되는 원인이 된다.

이를 해결하기 위해 하나의 논리적 테이블을 여러 개의 물리적인 공간으로 나누는 파티션을 설정할 수 있다. **파티션**은 대용량 서비스를 하는 데이터베이스에서 효율적으로 관리하고 동작하기 위해 지원하는 옵션이다. 파티션은 서로 다른 테이블스페이스에 생성할 수 있으며, 입출력과 같은 물리적인 제약을 감소시킬 수 있다.

하나의 테이블로만 모든 데이터가 유지된다면, 모든 트랜잭션이 한 곳에 집중하게 된다. 이로 인해 각 트랜잭션이 다른 트랜잭션을 대기하는 일이 많아져서 데이터베이스 성능이 저하된다. 하지만 파티션으로 나눈 경우에는 각 트랜잭션은 자신이 접근해야 할 파티션에만 접근하면 되므로 대기 확률이 줄어든다.

일부 DML 문장의 경우 특정 파티션에 있는 데이터만 접근하면 되므로, 전체 테이블을 모두 읽는 것보다 1/N(파티션 개수)의 정보만을 검색할 수 있다.

파티션은 다음과 같이 세 가지 종류가 있다.

파티션	설명
RANGE	각 파티션에 포함될 RANGE를 지정하여 파티션을 정의한다.
HASH	HASH 함수를 이용하여 파티션을 정의한다.
LIST	각 파티션에 포함될 값을 직접 지정하여 파티션을 정의한다.

4.9.1. 파티션의 생성

파티션을 생성하기 위해서는 **CREATE TABLE** 문을 사용할 때 파티션의 정보를 정의함으로써 파티션된 테이블을 만들 수 있다. 테이블을 만들 수 있는 권한만 있다면 특별한 권한은 필요하지 않다.

다음은 파티션으로 구성한 테이블을 생성하는 예이다.

[예 4.30] 파티션의 생성

```
CREATE TABLE PARTITIONED_TABLE1 (C1 NUMBER, C2 CLOB, C3 NUMBER)
             PARTITION BY RANGE (C1, C3)
             (
                PARTITION PART1 VALUES LESS THAN (30, 40),
                PARTITION PART2 VALUES LESS THAN (50, 60),
                PARTITION PART3 VALUES LESS THAN (60, 70)
             );
```

테이블을 파티션으로 나누는 방법은 범위를 통해 가능하다. 위의 예에서는 각 파티션에 범위를 지정하여 해당 파티션에 데이터를 삽입한다. 이를 통해 데이터가 어느 파티션에 속해 있는지를 알 수 있다. 파티션은 최대 10,000개까지 생성할 수 있다.

파티션의 또 다른 장점은 관리의 편의성이다. 각 연도별로 파티션을 나눈 테이블이 있다고 했을 때 필요치 않은 10년 전의 정보를 저장하고 있는 해당 파티션을 제거함으로써 쉽게 삭제(DROP)할 수 있다. 또한 다음 해에 해당하는 파티션이 필요하다면 새로운 파티션을 추가(ADD)할 수 있다.

파티션의 제거와 추가는 ALTER TABLE 문에서 파티션과 관련된 옵션을 사용함으로써 가능하다.

```
ALTER TABLE PARTITIONED_TABLE1 ADD PARTITION PART3
        VALUES LESS THAN (70, 80);

ALTER TABLE PARTITIONED_TABLE1 DROP PARTITION PART1;
```

파티션을 정의할 때 다음과 같은 **주의 사항**이 있다.

- 각 파티션을 정의한 순서에 따라 범위가 정렬되어야 한다.

  예를 들면, 다음의 문장은 에러가 발생한다.

```
CREATE TABLE PARTITIONED_TABLE2 (C1 NUMBER, C2 CLOB, C3 NUMBER)
             PARTITION BY RANGE (C1, C3)
             (
                PARTITION PART1 VALUES LESS THAN (50, 20),
                PARTITION PART2 VALUES LESS THAN (30, 10),
                PARTITION DEF_PART VALUES LESS THAN (MAXVALUE, MAXVALUE)
             );
```

  PART1이 PART2보다 먼저 선언되었지만 범위가 오히려 PART1이 PART2를 포함하는 것을 볼 수 있다. 범위를 정의할 때 VALUES LESS THAN 절은 **'이전 PARTITION에 들어가지 않고 ~ 보다 작은 값을 갖는 데이터를 포함하는 파티션'** 이라는 의미를 가진다. 그러므로 항상 파티션의 범위는 정렬되어야 한다.

- ALTER TABLE 문에 의해 새로 만들어진 파티션은 기존의 마지막 파티션의 범위보다 높은 범위를 가져야 한다.

  범위 간의 비교는 선행하는 파티션의 키가 더 큰 쪽이 큰 범위이다. 선행하는 파티션의 키가 MAXVALUE로 지정되어 새로운 파티션의 키로 더 큰 값을 지정할 수 없는 경우에는 파티션을 추가하거나 생성할 수 없으므로 주의해야 한다.

  다음은 이와 같은 에러를 발생시키는 예이다.

```
CREATE TABLE PARTITIONED_TABLE3 (C1 NUMBER, C2 CLOB, C3 NUMBER)
             PARTITION BY RANGE (C1, C3)
             (
                PARTITION PART1 VALUES LESS THAN (50, 20),
                PARTITION PART2 VALUES LESS THAN (60, 70)
             );

ALTER TABLE PARTITIONED_TABLE3 ADD PARTITION PART3
            VALUES LESS THAN (80, 40);

ALTER TABLE PARTITIONED_TABLE3 ADD PARTITION PART3
            VALUES LESS THAN (70, 100);
```

4.9.2. 복합 파티션의 생성

복합 파티션은 한 개의 키로 우선 파티션을 한 뒤 각 파티션을 같은 키 혹은 다른 키로 다시 파티션을 하는 테이블 파티션의 한 방식이다. 아래 예는 solid_date 컬럼을 이용해 월별로 RANGE 파티셔닝을 우선 한 뒤 각 파티션들을 다시 product_id로 HASH 파티셔닝한 예이다.

```
CREATE TABLE years_sales
(
    product_id          NUMBER,
    product_name        VARCHAR(20),
    price               NUMBER,
    sold_date           DATE
)
PARTITION BY RANGE (sold_date)
  SUBPARTITION BY HASH (product_id)
  (
    PARTITION jan_sales VALUES LESS THAN
                         (TO_DATE('31-01-2006', 'DD-MM-YYYY')),
    PARTITION feb_sales VALUES LESS THAN
                         (TO_DATE('28-02-2006', 'DD-MM-YYYY')),
    PARTITION mar_sales VALUES LESS THAN
                         (TO_DATE('31-03-2006', 'DD-MM-YYYY')),
    PARTITION apr_sales VALUES LESS THAN
                         (TO_DATE('30-04-2006', 'DD-MM-YYYY')),
    PARTITION may_sales VALUES LESS THAN
                         (TO_DATE('31-05-2006', 'DD-MM-YYYY')),
    PARTITION jun_sales VALUES LESS THAN
                         (TO_DATE('30-06-2006', 'DD-MM-YYYY')),
    PARTITION jul_sales VALUES LESS THAN
                         (TO_DATE('31-07-2006', 'DD-MM-YYYY')),
    PARTITION aug_sales VALUES LESS THAN
                         (TO_DATE('31-08-2006', 'DD-MM-YYYY')),
    PARTITION sep_sales VALUES LESS THAN
                         (TO_DATE('30-09-2006', 'DD-MM-YYYY')),
    PARTITION oct_sales VALUES LESS THAN
                         (TO_DATE('31-10-2006', 'DD-MM-YYYY'))
    SUBPARTITIONS 2,
    PARTITION nov_sales VALUES LESS THAN
                         (TO_DATE('30-11-2006', 'DD-MM-YYYY'))
    SUBPARTITIONS 4,
    PARTITION dec_sales VALUES LESS THAN
                         (TO_DATE('31-12-2006', 'DD-MM-YYYY'))
    (SUBPARTITION dec_1, SUBPARTITION dec_2,
     SUBPARTITION dec_3, SUBPARTITION dec_4)
  );
```

두개의 컬럼(위 예의 경우 solid_date와 product_id)에 대한 조건으로 빈번하게 조회가 일어나는 대용량 테이블에 대해 이런식으로 복합 파티셔닝을 하면 검색할 데이터의 양이 크게 줄기 때문에 성능상의 큰 이득을 볼 수 있다.

참고

각 복합 파티션에 대한 문법은 "Tibero RDBMS SQL 참조 안내서"를 참고한다.

4.9.3. 인덱스 파티션의 생성

테이블뿐만 아니라 인덱스도 파티션을 지정할 수 있다. 인덱스 또한 파티션을 통해 데이터베이스 성능을 향상시킬 수 있다. 인덱스는 다음과 같이 두 가지 방법으로 파티션을 나눌 수 있다.

로컬 파티션

테이블이 파티션되었을 때 테이블 파티션에 들어가는 키로 파티션을 나누는 방법이다. 각 파티션에 아무런 정보를 입력하지 않고 단지 **LOCAL**이라고 선언하면 된다. 이름은 자동으로 생성되며, 그 외 정보는 기본값으로 설정된다.

로컬 파티션으로 설정된 인덱스는 테이블의 한 파티션과 1:1로 대응된다. 로컬 파티션으로 설정된 인덱스의 한 파티션은 테이블의 한 파티션에 있는 로우만을 가리킨다.

다음은 로컬 파티션으로 인덱스를 생성하는 예이다.

[예 4.31] 로컬 파티션 인덱스의 생성

```
CREATE TABLE PARTITIONED_TABLE1 (C1 NUMBER, C2 CLOB, C3 NUMBER)
             PARTITION BY RANGE (C1, C3)
             (
                PARTITION PART1 VALUES LESS THAN (30, 40),
                PARTITION PART2 VALUES LESS THAN (50, 60),
                PARTITION DEF_PART VALUES LESS THAN (MAXVALUE, MAXVALUE)
             );

CREATE INDEX PARTITIONED_INDEX1 ON PARTITIONED_TABLE1 (C1) LOCAL
             (
                PARTITION IPART1 INITRANS 3,
                PARTITION IPART2 PCTFREE 10,
                PARTITION IPART3
             );
```

글로벌 파티션

테이블과는 무관하게 인덱스에 따로 파티션을 설정하는 방법이다. 테이블이 파티션으로 나뉘어져 있든 아니든 글로벌 파티션 인덱스를 만들 수 있다. 글로벌 파티션 인덱스의 한 파티션은 테이블의 어느 파티션에 있는 로우라도 가리킬 수 있다.

다음은 글로벌 파티션으로 인덱스를 생성하는 예이다.

[예 4.32] 글로벌 파티션 인덱스의 생성

```
CREATE TABLE PARTITIONED_TABLE1 (C1 NUMBER, C2 CLOB, C3 NUMBER)
             PARTITION BY RANGE (C1, C3)
             (
                PARTITION PART1 VALUES LESS THAN (30, 40),
                PARTITION PART2 VALUES LESS THAN (50, 60),
                PARTITION DEF_PART VALUES LESS THAN (MAXVALUE, MAXVALUE)
             );

CREATE INDEX PARTITIONED_INDEX1 ON PARTITIONED_TABLE1 (C1)
             GLOBAL PARTITION BY RANGE (C1, C3)
             (
                PARTITION IPART1 VALUES LESS THAN (10, 20) INITRANS 3
                PARTITION IPART2 VALUES LESS THAN (30, 40) PCTFREE 10
             );
```

4.9.4. 파티션의 정보 조회

Tibero RDBMS에서는 파티션된 스키마의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
DBA_PART_TABLES	Tibero RDBMS 내의 파티션된 모든 테이블의 정보를 조회하는 뷰이다.
USER_PART_TABLES	현재 사용자에 속한 파티션된 테이블의 정보를 조회하는 뷰이다.
ALL_PART_TABLES	사용자가 접근 가능한 파티션된 테이블의 정보를 조회하는 뷰이다.
DBA_PART_INDEXES	Tibero RDBMS 내의 파티션된 모든 인덱스의 정보를 조회하는 뷰이다.
USER_PART_INDEXES	현재 사용자에 속한 파티션된 인덱스의 정보를 조회하는 뷰이다.
ALL_PART_INDEXES	사용자가 접근 가능한 파티션된 인덱스의 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

제5장 사용자 관리와 데이터베이스 보안

데이터베이스 보안(Security)은 사용자가 고의나 실수로 데이터베이스에 저장된 데이터를 조작해 일관성을 손상시키거나 전체 데이터베이스를 파손시키는 일을 방지하려는 데 목적이 있다.

본 장에서는 Tibero RDBMS의 데이터베이스 보안을 위한 사용자 계정과 사용자의 특권(Privilege)및 역할(Role) 등을 효율적으로 관리하고 데이터베이스 사용을 감시(Audit)하기 위한 방법을 설명한다.

5.1. 사용자 관리

Tibero RDBMS 내부의 데이터에 접근하기 위해서는 사용자 계정(Account)이 필요하다. 각 계정은 패스워드(Password)를 통해 보안이 유지된다. 패스워드는 사용자 계정을 생성할 때 설정하며, 생성된 이후에 변경할 수 있다. Tibero RDBMS는 패스워드를 데이터 사전(Data Dictionary)에 암호화된 형태로 저장한다.

Tibero RDBMS에서 하나의 사용자 계정은 하나의 스키마를 가지며, 스키마의 이름은 사용자의 이름과 같다.

참고

스키마(Schema)는 테이블, 뷰, 인덱스 등의 스키마 객체(Schema object)의 묶음이다.

특정 스키마에 속한 스키마 객체를 나타내려면 다음과 같이 입력해야 한다.

```
사용자 계정.스키마 객체
```

다음은 tibero라는 사용자 계정으로 접속하여 SYS 사용자의 SYSTEM_PRIVILEGES에 접근하는 예이다.

```
$ tbsql tibero/tmax

tbSQL 4 SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Connected to Tibero.

SQL> SELECT * FROM SYS.SYSTEM_PRIVILEGES;

...
```

다음은 스키마 객체 앞에 특정한 사용자 계정을 명시하지 않았을 경우의 예이다.

```
SELECT * FROM V$DATABASE;
```

사용자 계정이 생략되면, 스키마 객체에 접근할 때 다음과 같은 과정을 거친다.

1. 사용자 자신이 소유한 스키마 객체 중에 V$DATABASE가 있는지 검색한다.

   tibero라는 사용자 계정으로 접속했다고 가정하면, tibero.V$DATABASE를 검색한다.

2. 사용자가 소유한 스키마 객체 중 V$DATABASE 가 존재하지 않으면, PUBLIC 사용자가 소유한 동의어를 검색한다. 즉, PUBLIC.V$DATABASE를 검색한다.

   PUBLIC 사용자는 오직 동의어만을 소유할 수 있다. PUBLIC 사용자가 소유한 동의어를 검색할 때, 스키마 객체 앞에 PUBLIC 사용자 계정을 명시할 수 없다.

   PUBLIC 사용자가 소유한 동의어 V$DATABASE를 찾는다고 가정하면, PUBLIC.V$DATABASE라고 입력할 수 없고, V$DATABASE만 입력해야 한다.

3. PUBLIC.V$DATABASE라는 이름의 스키마 객체가 없거나 있어도 사용자가 PUBLIC.V$DATABASE 객체에 대한 특권이 없다면 에러를 반환한다.

5.1.1. 사용자의 생성, 변경, 제거

사용자를 새로 생성하거나 변경, 제거하기 위해서는 DBA 특권을 가진 사용자로 Tibero RDBMS에 접속해야 한다.

Tibero RDBMS에서는 기본적으로 SYS라는 사용자를 제공한다. SYS 사용자는 Tibero RDBMS를 설치하는 과정에서 생기는 계정으로, DBA 역할이 부여된다.

SYS 사용자로 Tibero RDBMS에 접속하는 방법은 다음과 같다.

```
$ tbsql SYS/tibero

tbSQL 4 SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Connected to Tibero.

SQL>
```

SYS 사용자는 DBA의 역할이 부여된 만큼 될 수 있으면 다른 사용자 계정을 사용할 것을 권장한다.

사용자의 생성

사용자를 생성할 때에는 데이터베이스 보안 정책에 따라 적당한 특권을 가진 사용자 계정을 생성한다.

사용자를 생성하는 방법은 다음과 같다.

```
SQL> CREATE USER Steve                  ... ① ...
            IDENTIFIED BY dsjeoj134     ... ② ...
            DEFAULT TABLESPACE USR;     ... ③ ...
```

① CREATE USER 문을 사용하여 **Steve**라는 사용자를 생성한다.

② 사용자 Steve의 패스워드를 dsjeoj134로 설정한다. CREATE USER 문을 사용할 때에는 반드시 패스워드를 설정해야 한다.

③ 디폴트 테이블스페이스를 USR로 설정한다.

다음은 위와 동일한 방법으로 여러 사용자를 생성하는 예이다.

```
SQL> CREATE USER Peter
             IDENTIFIED BY abcd;
User 'PETER' created.

SQL> CREATE USER John
             IDENTIFIED BY asdf;
User 'JOHN' created.

SQL> CREATE USER Smith
             IDENTIFIED BY aaaa;
User 'SMITH' created.

SQL> CREATE USER Susan
             IDENTIFIED BY bbbb;
User 'SUSAN' created.
```

위와 같이 테이블스페이스를 지정하지 않으면, 자동으로 시스템 테이블스페이스를 사용하게 된다. 새로 생성된 사용자는 아무런 특권을 가지지 않으며, 데이터베이스에 접속할 수 없다. 데이터베이스에 접속하기 위해서는 CREATE SESSION 시스템 특권이나, 이를 포함하는 역할을 부여 받아야 한다.

다음은 생성된 사용자에게 특권을 할당하는 예이다.

```
SQL> GRANT CONNECT TO Peter;
Granted.

SQL> GRANT RESOURCE TO John;
Granted.

SQL> GRANT CONNECT TO Smith;
```

```
Granted.

SQL> GRANT DBA TO Susan;
Granted.
```

참고

자세한 내용은 "5.2. 특권"을 참고한다.

사용자의 변경

사용자에게 설정된 패스워드, 테이블스페이스 등을 변경하는 방법은 다음과 같다.

```
ALTER USER Peter                        ... ① ...
      IDENTIFIED BY abcdef              ... ② ...
      DEFAULT TABLESPACE SYSTEM;        ... ③ ...
```

① ALTER USER 문을 사용하여 Peter라는 사용자의 정보를 변경한다.

② 사용자 Peter의 패스워드를 abcdef로 변경한다.

③ 테이블스페이스를 **SYSTEM** 테이블스페이스로 변경한다.

사용자의 제거

사용자를 제거하는 방법은 다음과 같다.

```
DROP USER user_name CASCADE;
```

항목	설명
DROP USER user_name	DROP USER 문을 통해 user_name에 설정된 사용자를 제거한다.
CASCADE	DROP USER 문의 옵션 중 하나인 CASCADE는 사용자를 제거하기 전에 그 사용자의 모든 스키마 객체를 제거한다. CASCADE 옵션을 사용하지 않으면, 해당 사용자가 아무런 스키마 객체를 가지고 있지 않을 경우에만 사용자를 제거할 수 있다. 제거된 사용자의 스키마 객체를 참조하는 뷰나 동의어, 프로시저, 함수는 모두 INVALID 상태가 된다. 나중에 동일한 이름의 다른 사용자가 만들어지면, 새로운 사용자는 그 이름을 가졌던 이전 사용자로부터 아무것도 상속받지 못한다.

다음은 John 이라는 사용자를 제거하는 예이다.

```
DROP USER John CASCADE;
```

5.1.2. 사용자의 정보 조회

Tibero RDBMS에서는 사용자의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
ALL_USERS	데이터베이스의 모든 사용자의 기본적인 정보를 조회하는 뷰이다.
DBA_USERS	데이터베이스의 모든 사용자의 자세한 정보를 조회하는 뷰이다.
USER_USERS	현재 사용자의 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

5.2. 특권

사용자가 데이터베이스의 특정 스키마 객체에 접근하려면 특권을 부여 받아야 한다. 특권을 부여받으면 허용된 스키마 객체에 SQL 문장 등을 실행할 수 있다.

다음은 사용자에게 특권을 부여하는 예이다.

```
SQL> conn Peter/abcdef          ... ① ...
Connected.

SQL> CREATE TABLE EMPLOYEE
     (ID NUMBER, EMPLOYEE_NAME VARCHAR(20), ADDRESS VARCHAR(50));  ... ② ...
Created.

SQL> GRANT SELECT ON EMPLOYEE TO Smith;   ... ③ ...
Granted.
```

① Peter 라는 사용자 계정으로 데이터베이스에 접속한다.

② CREATE TABLE 문을 사용하여 **EMPLOYEE** 테이블을 생성한다. 총 3개의 컬럼(ID , EMPLOYEE_NAME, ADDRESS)을 생성한다.

③ Smith 라는 사용자가 Peter 사용자가 생성한 **EMPLOYEE** 테이블에 GRANT 명령을 실행하여 SELECT 특권을 부여한다.

이제 사용자 Smith는 Peter의 EMPLOYEE 테이블을 조회할 수 있다.

PUBLIC 사용자에게 특권을 부여할 수 있는데, 존재하는 모든 사용자에게 특권을 부여한 것과 동일한 효과를 갖는다. 특권을 효율적으로 관리 하기 위해, 특권의 집합인 역할을 생성할 수 있다. 동일한 특권을 부여해야 할 사용자가 많은 경우, 역할을 이용함으로써 GRANT 명령의 사용 횟수를 줄일 수 있다. 특권과 마

찬가지로 역할도 PUBLIC 사용자에게 부여하면 모든 사용자에게 역할을 부여한 것과 동일한 효과를 갖는다.

Tibero RDBMS에서 제공하는 특권은 크게 두 가지로 나뉜다.

- 스키마 객체 특권(Schema Object Privilege)

  특정 객체에 대한 질의 및 갱신 특권이다.

- 시스템 특권(System Privilege)

  데이터베이스에서 특정 작업을 수행할 수 있는 특권이다.

5.2.1. 스키마 객체 특권

스키마 객체 특권은 스키마 객체인 테이블, 뷰, 시퀀스, 동의어 등에 접근하는 것을 제어하는 권한이다. 스키마 객체 특권은 GRANT 명령을 사용해 다른 사용자에게 부여할 수 있으며, 그 내용은 데이터 사전에 기록된다.

스키마 객체 특권은 다음과 같다.

스키마 객체 특권	설명
SELECT	테이블을 조회하는 권한이다.
INSERT	테이블에 로우를 삽입하는 권한이다.
UPDATE	테이블에 로우를 갱신하는 권한이다.
DELETE	테이블에 로우를 삭제하는 권한이다.
ALTER	스키마 객체의 특성을 변경하는 권한이다.
INDEX	테이블에 인덱스를 생성하는 권한이다.
REFERENCES	테이블을 참조하는 제약조건을 생성하는 권한이다.

스키마 객체 특권의 부여

다음은 WITH GRANT OPTION을 이용하여 스키마 객체 특권을 부여하는 예이다.

```
SQL> GRANT SELECT, UPDATE (EMPLOYEE_NAME, ADDRESS) ON EMPLOYEE
           TO Smith WITH GRANT OPTION;
```

WITH GRANT OPTION을 이용하여 특권을 부여받았을 경우, 특권을 부여받은 사용자가 부여받은 특권을 다른 사용자에게 부여할 수 있다.

사용자 Peter가 위의 GRANT 명령을 실행하기 위해서는 특권을 갖고 있어야 한다. 그 특권은 스키마 객체 EMPLOYEE에 대한 SELECT, UPDATE 특권과 WITH GRANT OPTION을 함께 사용할 수 있는 특권이다.

위의 명령이 실행되면 사용자 Smith는 EMPLOYEE에 대한 SELECT 및 UPDATE 특권을 갖게 된다. 이때, EMPLOYEE에 대한 UPDATE 특권은 컬럼 EMPLOYEE_NAME과 ADDRESS에 한한다. 사용자 Smith는 또한 EMPLOYEE에 대한 SELECT, UPDATE, WITH GRANT OPTION 특권을 다른 사용자에게 부여할 수 있는 특권도 갖게 된다.

마찬가지로 사용자 Susan과 John에게도 다음과 같은 특권을 할당한다.

```
SQL> GRANT ALL ON EMPLOYEE TO Susan WITH GRANT OPTION;
Granted.

SQL> GRANT SELECT, DELETE ON EMPLOYEE TO John WITH GRANT OPTION;
Granted.
```

스키마 객체 특권의 회수

다른 사용자로부터 스키마 객체 특권을 회수하기 위해서는 REVOKE 명령을 사용해야 한다. 이 명령은 사용자의 스키마 객체 특권의 일부 또는 전체를 회수할 수 있다. DBA는 자신이 직접 부여하지 않는 특권에 대해서도 다른 사용자로부터 회수할 수 있다.

다음은 각각 Peter로부터 테이블 EMPLOYEE에 대한 DELETE 특권을 회수하고 John으로부터 모든 특권을 회수하는 예이다.

```
REVOKE DELETE ON EMPLOYEE FROM Peter;

REVOKE ALL ON EMPLOYEE FROM John;
```

WITH GRANT OPTION과 함께 부여된 특권을 회수할 때에는 연속적으로 특권이 회수된다.

다음은 Peter로부터 부여받은 Peter.EMPLOYEE에 대한 특권을 사용자 Smith가 Susan에게 부여하는 예이다.

```
SQL> conn Smith/abcd
Connected.

SQL> GRANT ALL ON Peter.EMPLOYEE TO Susan;
Granted.
```

사용자 Peter가 다음과 같이 Smith에게 부여한 테이블 EMPLOYEE의 특권을 회수하면, 사용자 Smith가 Susan에게 부여한 같은 특권도 동시에 회수된다.

```
SQL> conn Peter/abcdef
Connected

SQL> REVOKE ALL ON EMPLOYEE FROM Smith;
```

5.2.2. 시스템 특권

데이터베이스를 관리하는 데 필요한 시스템 명령어를 사용하기 위해서는 **시스템 특권**을 부여 받아야 한다. 시스템 특권은 기본적으로 SYS 사용자가 소유하고 있으며, 다른 사용자에게 부여할 수 있다.

시스템 특권은 다음과 같다.

시스템 특권	설명
ALTER SYSTEM	ALTER SYSTEM 문을 실행할 수 있는 권한이다.
CREATE SESSION	데이터베이스에 세션을 생성할 수 있는 권한이다. 즉 로그인이 가능하다는 것을 의미한다.
CREATE USER	사용자를 생성하는 권한이다.
ALTER USER	사용자의 정보를 변경하는 권한이다.
DROP USER	사용자를 제거하는 권한이다.
CREATE TABLESPACE	테이블스페이스를 생성하는 권한이다.
ALTER TABLESPACE	테이블스페이스를 변경하는 권한이다.
DROP TABLESPACE	테이블스페이스를 제거하는 권한이다.
CREATE TABLE	자신의 스키마에 테이블을 생성하는 권한이다.
CREATE ANY TABLE	임의의 스키마에 테이블을 생성하는 권한이다.
ALTER ANY TABLE	임의의 스키마에 속한 테이블을 변경하는 권한이다.
DROP ANY TABLE	임의의 스키마에 속한 테이블을 제거하는 권한이다.
COMMENT ANY TABLE	임의의 스키마에 속한 테이블에 주석을 추가하는 권한이다.
SELECT ANY TABLE	임의의 스키마에 속한 테이블을 조회하는 권한이다.
INSERT ANY TABLE	임의의 스키마에 속한 테이블에 로우를 삽입하는 권한이다.
UPDATE ANY TABLE	임의의 스키마에 속한 테이블에 로우를 갱신하는 권한이다.
DELETE ANY TABLE	임의의 스키마에 속한 테이블에 로우를 제거하는 권한이다.
CREATE ANY INDEX	임의의 스키마에 속한 테이블에 인덱스를 생성하는 권한이다.
ALTER ANY INDEX	임의의 스키마에 속한 테이블에 인덱스를 수정하는 권한이다.
DROP ANY INDEX	임의의 스키마에 속한 테이블에 인덱스를 제거하는 권한이다.
CREATE SYNONYM	자신의 스키마에 동의어를 생성하는 권한이다.
CREATE ANY SYNONYM	임의의 스키마에 동의어를 생성하는 권한이다.
DROP ANY SYNONYM	임의의 스키마에 속한 동의어를 제거하는 권한이다.
SYSDBA	SHUTDOWN, ALTER DATABASE, CREATE DATABASE, ARCHIVELOG, RECOVERY 문을 실행할 수 있는 권한이다.
CREATE PUBLIC SYNONYM	PUBLIC 스키마에 동의어를 생성하는 권한이다.
DROP PUBLIC SYNONYM	PUBLIC 스키마에 속한 동의어를 제거하는 권한이다.

시스템 특권	설명
CREATE VIEW	자신의 스키마에 뷰를 생성하는 권한이다.
CREATE ANY VIEW	임의의 스키마에 뷰를 생성하는 권한이다.
DROP ANY VIEW	임의의 스키마에 속한 뷰를 제거하는 권한이다.
CREATE SEQUENCE	자신의 스키마에 시퀀스를 생성하는 권한이다.
CREATE ANY SEQUENCE	임의의 스키마에 시퀀스를 생성하는 권한이다.
ALTER ANY SEQUENCE	임의의 스키마에 속한 시퀀스를 변경하는 권한이다.
DROP ANY SEQUENCE	임의의 스키마에 속한 시퀀스를 제거하는 권한이다.
SELECT ANY SEQUENCE	임의의 스키마에 속한 시퀀스를 조회하는 권한이다.
CREATE ROLE	역할을 생성하는 권한이다.
DROP ANY ROLE	역할을 제거하는 권한이다.
GRANT ANY ROLE	임의의 역할에 부여하는 권한이다.
ALTER ANY ROLE	역할을 수정하는 권한이다.
ALTER DATABASE	데이터베이스를 변경하는 권한이다.
CREATE PROCEDURE	자신의 스키마에 프로시저를 생성하는 권한이다.
CREATE ANY PROCEDURE	임의의 스키마에 프로시저를 생성하는 권한이다.
ALTER ANY PROCEDURE	임의의 스키마에 속한 프로시저를 변경하는 권한이다.
DROP ANY PROCEDURE	임의의 스키마에 속한 프로시저를 제거하는 권한이다.
EXECUTE ANY PROCEDURE	임의의 스키마에 속한 프로시저를 실행하는 권한이다.
CREATE TRIGGER	자신의 스키마에 속한 트리거를 생성하는 권한이다.
CREATE ANY TRIGGER	임의의 스키마에 속한 트리거를 생성하는 권한이다.
ALTER ANY TRIGGER	임의의 스키마에 속한 트리거를 변경하는 권한이다.
DROP ANY TRIGGER	임의의 스키마에 속한 트리거를 제거하는 권한이다.
GRANT ANY OBJECT PRIVILEGE	모든 스키마 객체에 대한 특권을 가지는 권한이다.
GRANT ANY PRIVILEGE	모든 특권을 전부 부여할 수 있는 권한이다.

시스템 특권의 부여

시스템 특권도 WITH ADMIN OPTION을 이용하여 다른 사용자에게 특권을 부여할 수 있다. 단, 특권을 부여할 때에는 해당 특권을 소유하고 있어야 한다.

다음은 시스템 특권을 가지고 있는 사용자 SYS가 Susan에게 WITH ADMIN OPTION과 함께 GRANT SELECT ANY TABLE 특권을 부여하는 예이다.

```
SQL> conn SYS/tibero
Connected to Tibero.
```

```
SQL> GRANT SELECT ANY TABLE TO Susan WITH ADMIN OPTION;
Granted.
```

사용자 Susan은 SYS 사용자에게 부여 받은 시스템 특권을 다른 사용자에게 부여할 수 있는 권한이 생성된다. 즉, 다른 사용자에게 데이터베이스 내의 모든 테이블을 조회할 수 있는 특권을 부여할 수 있다는 의미이다.

시스템 특권의 회수

WITH ADMIN OPTION과 함께 부여된 특권은 WITH GRANT OPTION과는 달리 연속적으로 회수되지 않는다.

다음은 사용자 Susan이 사용자 SYS로부터 부여받은 특권을 Peter에게 부여하는 예이다.

```
SQL> conn Susan/abcd
Connected to Tibero.

SQL> GRANT SELECT ANY TABLE TO Peter;
Granted.
```

다음과 같이 Susan에게 부여한 시스템 특권을 회수하더라도, 사용자 Susan이 Peter에게 부여한 시스템 특권은 그대로 유지된다.

```
SQL> conn SYS/tibero
Connected to Tibero.

SQL> REVOKE SELECT ANY TABLE FROM Susan;
```

5.2.3. 특권의 정보 조회

Tibero RDBMS에서는 특권의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
DBA_SYS_PRIVS	사용자에게 부여된 모든 특권의 정보를 조회하는 뷰이다.
USER_SYS_PRIVS	현재 사용자에게 부여된 특권의 정보를 조회하는 뷰이다.
DBA_TBL_PRIVS	데이터베이스 내의 모든 스키마 객체 특권의 정보를 조회하는 뷰이다.
USER_TBL_PRIVS	현재 사용자가 소유한 모든 스키마 객체 특권의 정보를 조회하는 뷰이다.
ALL_TBL_PRIVS	현재 사용자가 소유한 모든 스키마 객체 특권과 모든 사용자가 사용할 수 있도록 공개한 모든 스키마 객체 특권의 정보를 조회하는 뷰이다.

정적 뷰	설명
DBA_COL_PRIVS	데이터베이스 내의 모든 스키마 객체 특권 중 스키마 객체의 특정 컬럼에 부여된 특권의 정보를 조회하는 뷰이다.
USER_COL_PRIVS	현재 사용자가 소유한 스키마 객체 특권 중 스키마 객체의 특정 컬럼에 부여된 특권의 정보를 조회하는 뷰이다.
ALL_COL_PRIVS	현재 사용자가 소유한 스키마 객체 특권 혹은 모든 사용자가 사용할 수 있도록 공개한 모든 스키마 객체 특권 중 스키마 객체의 특정 컬럼에 부여된 특권의 정보를 조회하는 뷰이다.
USER_COL_PRIVS_MADE	현재 사용자 소유한 객체 중 특정 컬럼에 부여된 특권의 정보를 조회하는 뷰이다.
ALL_COL_PRIVS_MADE	현재 사용자가 만든 특권 중 스키마 객체의 특정 컬럼에 부여된 특권의 정보를 조회하는 뷰이다.
USER_COL_PRIVS_RECD	현재 사용자에게 부여된 모든 스키마 객체 특권 중 스키마 객체의 특정 컬럼에 부여된 특권의 정보를 조회하는 뷰이다.
ALL_COL_PRIVS_RECD	현재 사용자 또는 PUBLIC 사용자에게 부여된 모든 스키마 객체 특권 중 스키마 객체의 특정 컬럼에 부여된 특권의 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

5.3. 역할

사용자는 데이터베이스와 관련된 애플리케이션 프로그램을 실행하려면 해당 스키마 객체에 대한 적절한 특권을 부여 받아야 한다. 예를 들어, 20개의 테이블에 30명의 사용자가 접근하는 경우라면 DBA는 20개 테이블 * 30명의 사용자로 계산된 총 600번의 특권을 부여하는 작업을 수행해야 한다.

특권은 시간을 필요로 하는 작업이다. 따라서 특권의 효율적인 관리를 위해 DBA가 부여할 특권을 모아놓은 역할을 미리 정의한다면 600번의 특권 부여 작업을 실행할 필요가 없고 특권의 관리가 훨씬 더 간편해진다. **역할** (Role)은 여러 특권을 모아 놓은 것이며, 하나의 단위로서 사용자에게 부여할 수 있다. 역할을 생성하거나 변경, 부여하기 위해서는 이에 맞는 특권이 필요하다.

5.3.1. 역할의 생성, 부여, 회수

다음은 역할을 생성하거나 변경, 특권을 부여하는 예이다.

```
SQL> conn SYS/tibero
Connected to Tibero.

SQL> GRANT CREATE ROLE, ALTER ANY ROLE, GRANT ANY ROLE TO Peter;
Granted.
```

위의 예에서는 사용자 SYS로 접속하여, 사용자 Peter에게 CREATE ROLE, ALTER ANY ROLE, GRANT ANY ROLE 특권을 부여하고 있다.

역할의 생성

다음은 역할을 생성하는 예이다.

```
SQL> CREATE ROLE APP_USER;
Role 'APP_USER' created.

SQL> GRANT CREATE SESSION TO APP_USER;
Granted.

SQL> CREATE ROLE CLERK;
Role 'CLERK' created.

SQL> GRANT SELECT, INSERT ON Peter.EMPLOYEE TO CLERK;
Granted.

SQL> GRANT SELECT, INSERT ON Peter.TIME_CARDS TO CLERK;
Granted.

SQL> GRANT SELECT, INSERT ON Peter.DEPARTMENT TO CLERK;
Granted.
```

다음은 본 예제를 기준으로 생성된 역할을 설명한다.

역할	설명
APP_USER	CREATE ROLE 문을 사용하여 역할 **APP_USER**를 생성한다. 시스템 특권인 CREATE SESSION 문이 역할 APP_USER에 부여된다. APP_USER 역할을 부여받은 사용자는 데이터베이스에 접속할 수 있다.
CLERK	CREATE ROLE 문을 사용하여 **CLERK** 역할을 생성한다. 여러 개의 테이블에 스키마 객체 특권이 부여된다. - Peter 스키마에 속한 EMPLOYEE 테이블에 SELECT, INSERT를 할 수 있다. - Peter 스키마에 속한 TIME_CARDS 테이블에 SELECT, INSERT를 할 수 있다. - Peter 스키마에 속한 DEPARTMENT 테이블에 SELECT, INSERT를 할 수 있다.

역할의 부여

역할을 다른 역할에게 부여할 수 있다.

예를 들면, 다음과 같이 역할 APP_USER를 역할 CLERK에 부여할 수 있다.

```
SQL> GRANT APP_USER TO CLERK;
Granted.
```

위의 SQL 문장이 실행되면 역할 CLERK을 부여 받은 사용자에게 동시에 역할 APP_USER가 부여되고, 역할 CLERK과 APP_USER에 양쪽에 포함된 모든 특권을 사용할 수 있게 된다.

역할을 부여 받은 사용자는 그 역할을 다른 사용자에게 다시 부여할 수 있다. 단, 역할을 부여하는 사용자가 그 역할을 다음과 같이 WITH ADMIN OPTION과 함께 부여 받아야 한다.

```
SQL> GRANT CLERK TO Susan WITH ADMIN OPTION;
Granted.

SQL> GRANT CLERK TO Peter;
Granted.
```

위의 GRANT 명령이 실행되면, 사용자 Susan은 부여된 역할을 다시 다른 사용자에게 부여할 수 있으며, 그 사용자로부터 역할을 회수할 수도 있다. 하지만 사용자 Peter는 CLERK 역할을 사용만 할 뿐, 그 역할을 다시 다른 사용자에게 부여하거나 회수하지는 못 한다.

역할의 회수

다른 사용자로부터 역할을 회수시키기 위해서는 **REVOKE** 명령을 사용해야 한다. DBA는 자신이 직접 부여하지 않은 역할에 대해서도 다른 사용자로부터 회수할 수 있다.

다음은 사용자 Peter와 역할 CLERK으로부터 역할 APP_USER를 회수하는 예이다.

```
REVOKE APP_USER FROM Peter;
REVOKE APP_USER FROM CLERK;
```

위의 예에서는 회수 대상이 사용자든 역할이든 문법은 동일하다.

5.3.2. 미리 정의된 역할

Tibero RDBMS에서는 빈번하게 사용되는 시스템 특권을 모아 다음과 같은 역할로 미리 정의하고 있다.

역할	포함된 특권	설명
CONNECT	CREATE SESSION	단순히 데이터베이스에 접속만 할 수 있는 특권이다. 이 특권은 모든 사용자에게 필요한 특권이다.

역할	포함된 특권	설명
RESOURCE	CREATE PROCEDURE CREATE SEQUENCE CREATE TABLE CREATE TRIGGER	자신의 스키마에 기본적인 스키마 객체를 생성하는 특권이다. 이 특권은 애플리케이션 프로그램 개발자에게 필요한 기본적인 특권이다.
DBA	-	DBA에게 필요한 시스템 특권을 포함하고 있는 특권이다. 이 역할을 부여 받은 DBA는 시스템 특권을 임의로 다른 사용자에게 부여할 수 있다. 모든 시스템 특권이 WITH ADMIN OPTION으로 부여된다.

5.3.3. 기본 역할

사용자에게 부여한 역할은 한 세션 내에서 SET ROLE 명령을 사용함으로써 동적으로 활성화하거나 비활성화할 수 있다. 예를 들어, 사용자가 CLERK, RESOURCE, APP_USER 역할을 가지고 있다면 다음과 같은 명령을 사용하여 이 중 필요한 역할만을 활성화할 수 있다.

```
SET ROLE CLERK, RESOURCE;       /* CLERK, RESOURCE 역할을 활성화한다. */
SET ROLE ALL EXCEPT CLERK;      /* CLERK를 제외한 모든 역할을 활성화한다. */
SET ROLE ALL;                   /* 모든 역할을 활성화한다. */
SET ROLE NONE;                  /* 모든 역할을 비활성화한다. */
```

역할의 활성화와 비활성화는 사용자의 특권을 효율적으로 제어하는 데에 매우 유용하다. 사용자에게 애플리케이션 프로그램을 실행할 때에만 특정한 특권을 부여하길 원한다면, SET ROLE 명령을 사용하여 프로그램의 시작과 동시에 그 특권이 포함된 역할을 사용할 수 있도록 활성화하고, 프로그램 종료 직전에 그 역할의 사용을 중지시키기 위해 비활성화를 하면 된다.

사용자가 처음으로 세션에 접속할 때에 활성화 되는 역할을 그 사용자의 **기본 역할**이라고 한다. 새로 생성한 사용자는 처음에 기본 역할이 ALL로 설정된다. 즉, 사용자에게 부여하는 모든 역할이 기본적으로 활성화된다.

사용자의 기본 역할은 ALTER USER 문을 이용하여 바꿀 수 있으며, 그 형식은 SET ROLE 명령과 비슷하다.

```
ALTER USER Park DEFAULT ROLE CLERK, RESOURCE;
ALTER USER Park DEFAULT ROLE ALL EXCEPT CLERK;
ALTER USER Park DEFAULT ROLE ALL;
ALTER USER Park DEFAULT ROLE NONE;
```

5.3.4. 역할의 정보 조회

Tibero RDBMS에서는 역할의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다. DBA나 일반 사용자 모두 사용할 수 있다.

정적 뷰	설명
DBA_ROLES	Tibero RDBMS 내의 모든 역할의 정보를 조회하는 뷰이다.
DBA_ROLE_PRIVS	사용자나 다른 역할에 부여된 모든 역할의 정보를 조회하는 뷰이다.
USER_ROLE_PRIVS	현재 사용자나 PUBLIC 사용자에 부여된 역할의 정보를 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

5.4. 네트워크 접속 제어

네트워크 접속 제어(NAC: Network Access Control)는 허가되지 않은 사용자나 보안 문제를 가진 사용자의 네트워크 접속을 차단하고 제어하는 네트워크 보안 기술이다. Tibero RDBMS는 이러한 기술을 채택함으로써 기업 내 IT 자산을 보다 효과적으로 보호할 수 있다.

Tibero RDBMS에서 제공하는 네트워크 접속 제어는 네트워크 보안의 범위에 따라 크게 두 가지로 나뉜다.

- 전체 네트워크 접속 제어

  클라이언트 전체의 네트워크 접속을 차단하거나 허용한다.

- IP 주소 기반 네트워크 접속 제어

  클라이언트의 IP 주소에 따라 네트워크 접속을 차단하거나 허용한다.

5.4.1. 전체 네트워크 접속 제어

전체 네트워크 접속 제어는 클라이언트 전체를 더는 TCP/IP 네트워크로 접속하지 못하게 하거나 혹은 접속할 수 있도록 하는 방법이다.

다음은 클라이언트 전체의 네트워크 접속을 허용하는 방법이다. 다음의 명령은 Tibero RDBMS 서버를 처음 기동시킬 때와 같은 상태이다.

```
ALTER SYSTEM LISTENER REMOTE ON;
```

다음은 클라이언트 전체의 네트워크 접속을 차단하는 방법이다.

```
ALTER SYSTEM LISTENER REMOTE OFF;
```

위 명령을 실행하면 외부 클라이언트의 네트워크 접속은 차단되지만, 로컬 호스트에서 접속하는 클라이언트의 네트워크 접속은 여전히 허용된다. 단, 로컬 호스트의 tbdsn.tbr 파일에서 IP가 **'localhost'** 라고 설정된 경우에만 해당된다. 그리고 이미 접속된 클라이언트는 계속 유지된다.

5.4.2. IP 주소 기반 네트워크 접속 제어

IP 주소 기반 네트워크 접속 제어는 초기화 파라미터에 설정된 IP 주소에 따라 클라이언트의 네트워크 접속을 허용하거나 차단하는 방법이다.

- LSNR_INVITED_IP

  이 방법은 특정한 IP 주소를 갖는 클라이언트의 네트워크 접속은 허용되지만, 그 밖의 접속은 차단하는 경우에 사용한다.

  – 하나의 IP 주소는 **'IP 주소/서브넷 마스크의 bit 수'**와 같은 형식으로 표현된다.

  – 여러 개의 IP 주소를 설정하는 경우에는 각 IP 주소를 세미콜론(**;**)으로 구분하여 표기한다.

  – 서브넷 마스크의 bit 수가 32인 경우에는 표기를 생략할 수 있다. 위의 예제에서 192.168.1.1은 192.168.1.1/32와 같은 의미이다.

  ```
  예: 192.168.2.0/24
  ```

  192.168.2.0/24는 서브텟 마스크의 bit 수가 24bit이므로, 255.255.255.0과 같은 서브넷 마스크를 의미한다. 따라서 192.168.2.xxx의 IP 주소를 갖는 모든 클라이언트의 네트워크 접속을 허용한다는 의미이다.

  – 다음은 LSNR_INVITED_IP 초기화 파라미터를 이용하여 클라이언트의 네트워크 접속을 허용하는 방법이다.

  <$TB_SID.tip>

  ```
  LSNR_INVITED_IP=192.168.1.1;192.168.2.0/24;192.1.0.0/16
  ```

- LSNR_DENIED_IP

  이 방법은 특정한 IP 주소를 갖는 클라이언트의 네트워크 접속은 차단되지만, 그 밖의 접속은 허용하는 경우에 사용한다.

  – LSNR_INVITED_IP 초기화 파라미터의 설정 방법과 동일하다.

  – 다음은 LSNR_DENIED_IP 초기화 파라미터를 이용하여 클라이언트의 네트워크 접속을 차단하는 방법이다.

  <$TB_SID.tip>

```
LSNR_DENIED_IP=192.168.1.1;192.168.2.0/24;192.1.0.0/16
```

이 두 파라미터는 다음과 같은 특징이 있다.

- $TB_SID.tip 파일에 LSNR_INVITED_IP와 LSNR_DENIED_IP가 모두 설정되어 있는 경우 LSNR_DENITED_IP의 설정은 무시되며, LSNR_INVITED_IP만 적용된다. 즉 LSNR_INVITED_IP에 설정된 IP 주소의 클라이언트를 제외하고는 모든 접속이 차단된다.

- $TB_SID.tip 파일에 LSNR_INVITED_IP와 LSNR_DENIED_IP가 모두 설정되지 않은 경우 모든 클라이언트의 네트워크 접속이 허용된다.

- 루프백 주소(loopback address, 127.0.0.1)에서 접속하는 경우 LSNR_INVITED_IP 또는 LSNR_DENIED_IP의 설정과는 무관하게 항상 허용된다.

- Tibero RDBMS 서버를 운영하는 중에 서버를 다시 기동하지 않고 LSNR_INVITED_IP 또는 LSNR_DENIED_IP의 설정을 변경하려는 경우

  1. 우선 $TB_SID.tip 파일에 LSNR_INVITED_IP 또는 LSNR_DENIED_IP의 설정을 변경한 후, $TB_SID.tip 파일을 저장한다.

  2. 다음의 명령을 실행한다.

     ```
     alter system listener parameter reload;
     ```

     위의 명령을 실행하면 $TB_SID.tip 파일에서 LSNR_INVITED_IP 또는 LSNR_DENIED_IP의 내용을 다시 읽어 변경된 내용을 실시간으로 적용한다.

참고

해당 초기화 파라미터가 제대로 적용되었는지를 확인하기 위해서는 반드시 리스너의 트레이스 로그 파일의 내용을 확인해 볼 것을 권장한다.

5.5. 감사

감사(Auditing)는 데이터베이스 내에서 지정된 사용자의 동작을 기록하는 보안 기술이다. 관리자는 감사 기능을 통해 특정 동작 또는 특정 사용자에 대해 별도의 로그를 남김으로써 데이터베이스를 보다 효과적으로 보호할 수 있다.

감사 기능은 감사의 대상에 따라 두 종류로 구분된다.

- 스키마 객체에 대한 감사

  지정된 스키마 객체에 수행되는 모든 동작을 기록할 수 있다.

- 시스템 특권에 대한 감사

  지정된 시스템 특권을 사용하는 모든 동작을 기록할 수 있다.

감사 기록(Audit Trail)을 남기고 싶은 사용자 또는 역할을 지정할 수 있으며, 성공한 동작 또는 실패한 동작에 대해서만 감사 기록을 남기도록 지정할 수 있다. 또한 세션 별로 한 번만 감사 기록을 남기거나 동작이 수행될 때마다 감사 기록을 남기도록 지정할 수 있다.

5.5.1. 감사 설정과 해제

감사를 설정하거나 해제하려면 AUDIT 또는 NOAUDIT 명령을 사용한다.

스키마 객체에 대한 감사

다른 사용자가 소유한 스키마의 객체 또는 디렉터리 객체를 감사하기 위해서는 AUDIT ANY 시스템 특권을 부여받아야 한다.

다음은 스키마 객체에 대한 감사를 설정하는 예이다.

```
SQL> AUDIT delete ON t BY SESSION WHENEVER SUCCESSFUL;

Audited.
```

위의 SQL 문장이 성공하면 테이블 t에 수행되는 모든 delete 문에 대해 성공하는 경우에만 감사 기록을 남긴다.

시스템 특권에 대한 감사

시스템 특권을 감사하기 위해서는 AUDIT SYSTEM 시스템 특권을 부여받아야 한다.

다음은 시스템 특권에 대한 감사를 설정하는 예이다.

```
SQL> AUDIT create table BY tibero;

Audited.
```

위의 SQL 문장이 성공하면 tibero라는 사용자가 테이블을 생성하려고 할 때 그 것이 성공하든 실패하든 관계 없이 감사 기록을 남긴다.

참고

시스템 특권에 관한 자세한 내용은 “5.2. 특권”을 참고한다.

감사 해제

시스템 특권의 감사를 해제하기 위해서는 AUDIT SYSTEM 시스템 특권을 부여받아야 한다.

다른 사용자가 소유한 스키마의 객체 또는 디렉터리 객체의 감사를 해제하기 위해서는 AUDIT ANY 시스템 특권을 부여받아야 한다.

다음은 이미 설정된 감사를 해제하는 예이다.

```
SQL> NOAUDIT create table BY tibero;

Noaudited.
```

위의 SQL 문장이 성공하면 tibero라는 사용자가 테이블을 생성할 때 더 이상 감사 기록을 남기지 않는다.

참고

SYS 사용자를 감사의 대상으로 지정할 수 없다. SYS 사용자에 대한 감사는 "5.5.3. SYS 사용자에 대한 감사"을 참고한다.

5.5.2. 감사 기록

감사 기록(Audit Trail)은 명령을 수행한 사용자, 명령이 수행된 스키마 객체, 수행 시각, 세션 ID 등의 기본 정보와 실행된 SQL 문장으로 이루어진다.

감사 기록 저장

감사 기록은 $TB_SID.tip 파일에 설정된 AUDIT_TRAIL 파라미터에 따라 데이터베이스 내부 또는 OS 파일에 저장할 수 있다. OS 파일에 감사 기록을 저장하는 경우 파일의 위치와 최대 크기를 각각 $TB_SID.tip 파일의 AUDIT_FILE_DEST 파라미터와 AUDIT_FILE_SIZE 파라미터로 설정할 수 있다.

다음은 감사 기록의 저장 위치를 지정하는 예이다.

<$TB_SID.tip>

```
AUDIT_TRAIL=DB_EXTENDED
```

위와 같이 설정하면 감사 기록에 포함되는 기본 정보뿐만 아니라 사용자가 실행한 SQL문장까지 데이터베이스에 저장된다.

<$TB_SID.tip>

```
AUDIT_TRAIL=OS
AUDIT_FILE_DEST=/home/tibero/audit/audit_trail.log
AUDIT_FILE_SIZE=10M
```

위와 같이 설정하면 "/home/tibero/audit/audit_trail.log"에 최대 10MB의 크기로 감사 기록이 저장된다.

참고

1. $TB_SID.tip 파일 설정에 관한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

2. SYS 사용자에 대한 감사 기록은 데이터베이스에 저장되지 않는다. SYS 사용자에 대한 감사는 “5.5.3. SYS 사용자에 대한 감사”을 참고한다.

감사 기록 조회

감사 기록은 OS 파일 또는 데이터베이스에 저장된다. OS 파일에 저장하도록 설정한 경우 일반 텍스트 파일의 형태로 저장되므로 쉽게 내용을 열람할 수 있다. 데이터베이스에 저장하도록 설정한 경우에는 다음의 정적 뷰를 통해 감사 기록을 조회할 수 있다.

정적 뷰	설명
DBA_AUDIT_TRAIL	데이터베이스에 저장된 모든 감사 기록을 조회하는 뷰이다.
USER_AUDIT_TRAIL	데이터베이스에 저장된 현재 사용자의 감사 기록을 조회하는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

5.5.3. SYS 사용자에 대한 감사

SYS 사용자의 명령은 보안 상의 이유로 다른 사용자의 명령과는 다른 방식으로 감사된다. SYS 사용자는 기본적으로 감사의 대상에서 제외되기 때문에 AUDIT 또는 NOAUDIT 명령을 사용해 감사를 설정 또는 해제할 수 없다.

SYS 사용자의 명령을 감사하기 위해서는 $TB_SID.tip 파일의 AUDIT_SYS_OPERATIONS 파라미터를 Y로 설정해야 한다. SYS 사용자의 명령을 감사하도록 설정하면, 수행한 모든 동작이 OS 파일에 기록되며 보안 상의 이유로 데이터베이스에는 기록되지 않는다.

다음은 SYS 사용자의 동작을 감사하도록 설정한 예이다.

<$TB_SID.tip>

```
AUDIT_SYS_OPERATIONS=Y
AUDIT_FILE_DEST=/home/tibero/audit/audit_trail.log
AUDIT_FILE_SIZE=10M
```

위와 같이 설정하면 SYS 사용자가 수행한 모든 동작이 "/home/tibero/audit/audit_trail.log"에 최대 10MB의 크기로 저장된다.

제6장 백업과 복구

Tibero RDBMS는 시스템의 예상치 못한 오류 등으로 인해 데이터베이스가 비정상적으로 종료하거나 시스템에 물리적인 손상을 입은 상황을 대처하려고 다양한 백업과 복구 방법을 제공한다.

본 장에서는 이러한 백업과 복구 방법을 설명한다.

6.1. Tibero RDBMS의 구성 파일

Tibero RDBMS의 파일은 컨트롤 파일(Control file), 데이터 파일(Data file), 임시 파일(Temp file), 로그 파일(Log file)로 구성된다.

각 파일의 특징과 역할을 다음과 같다.

- **컨트롤 파일**

  컨트롤 파일은 Tibero RDBMS를 구성하는 모든 파일의 위치와 데이터베이스의 이름 등 데이터베이스의 구조를 저장하는 파일이다. 특히, Tibero RDBMS가 사용하는 데이터 파일, 로그 파일 등의 상태 정보가 기록된다. 컨트롤 파일은 Tibero RDBMS를 기동할 때 복구가 필요한지를 판단하는 데 사용된다.

  컨트롤 파일은 데이터베이스의 복구에 매우 중요한 파일이므로 다른 물리적 파티션에 여러 개 지정하기를 권장한다. 여러 개를 유지하면, 한 파티션에 장애가 발생하여 컨트롤 파일을 사용하지 못하더라도 다른 컨트롤 파일을 이용하여 복구할 수 있다.

  컨트롤 파일은 현재의 데이터베이스를 구성하는 파일에 관한 정보를 담고 있으므로 반드시 최신 것으로 유지해야 한다. 컨트롤 파일의 백업은 현재의 컨트롤 파일 자체를 백업하는 방식으로 이루어지지 않고, 컨트롤 파일을 생성하는 **CREATE CONTROLFILE** 문을 백업해 두었다가, 복구가 필요한 경우 백업해 놓은 컨트롤 파일 생성 문을 사용하여 컨트롤 파일을 다시 생성하는 방식으로 이루어진다.

  최신의 컨트롤 파일을 유지하기 위해서는 데이터베이스에 파일을 추가하거나 변경하는 등 구조상 변화가 있을 때마다 컨트롤 파일의 생성문을 백업해야 한다. 컨트롤 파일 자체를 백업해 두었다가 사용하는 것은 NOARCHIVELOG 모드에서처럼 데이터베이스 전체를 백업하는 경우에만 사용할 수 있다.

- **데이터 파일**

  데이터 파일은 사용자의 데이터를 저장하는 파일로써 로그 파일과 함께 데이터베이스를 구성하는 가장 중요한 파일이다.

  Permanent 테이블스페이스와 Undo 테이블스페이스에서 정의한 파일로 테이블, 인덱스 등의 데이터베이스 객체를 저장한다. 테이블스페이스는 한 개 이상의 데이터 파일로 이루어지며, 한 데이터 파일은 하나의 테이블스페이스에 속한다.

  데이터 파일은 사용자의 데이터가 물리적으로 저장되는 공간이므로 반드시 백업해야 한다.

- **임시 파일**

  임시 파일은 데이터베이스가 메모리에서 처리할 수 없는 방대한 양의 데이터를 다루는 경우 임시로 사용하기 위한 공간이다.

  Tibero RDBMS는 사용자의 질의를 처리하기 위해 정렬 등의 연산을 수행할 때와 임시 테이블(Temporary Table)의 데이터를 저장할 때 데이터 파일을 사용한다. 데이터 파일은 임시 테이블스페이스(Temporary Tablespace)에서만 정의할 수 있고, 임시 테이블스페이스는 하나 이상의 데이터 파일을 가질 수 있다.

  임시 파일은 데이터베이스를 구성하는 데이터가 물리적으로 저장되지는 않고 운영 중에 임시로 사용되는 영역이므로 백업할 필요가 없다.

- **로그 파일**

  로그 파일은 로그를 저장하는 파일이다. 데이터 파일에 기록되는 내용을 시간 순으로 기록하는 파일로써 데이터베이스를 복구할 때 사용한다. 로그 파일은 운영 중에 순환적으로 재사용되는 온라인 로그 파일과 재사용된 온라인 로그 파일을 보관하는 아카이브 로그 파일로 나뉜다.

  ARCHIVELOG 모드로 운영 중일 때만 아카이브 로그 파일이 만들어진다. NOARCHIVELOG 모드에서는 온라인 로그 파일만 사용한다. NOARCHIVELOG 모드에서는 재사용되어 없어진 로그 파일이 있을 수 있으므로 복구할 때 많은 제약이 따른다.

  로그 파일은 데이터베이스 복구할 때 복원된 데이터 파일을 최신 상태로 복구하기 위해, 데이터베이스가 어떻게 변경되어 왔는지에 대한 모든 정보를 기록한다. 따라서, 데이터 파일과 함께 반드시 백업을 해야 한다.

6.2. 백업

백업은 여러 가지 유형의 장애로부터 데이터베이스를 보호하는 것을 뜻한다. 즉 시스템 장애가 발생했을 때 복구를 하거나 시스템 작동을 유지하기 위한 절차 또는 기법이다.

Tibero RDBMS는 데이터베이스를 백업하는 방법을 크게 두 가지로 나누어 수행할 수 있다.

- 논리적 백업

  논리적 백업이란 테이블, 인덱스, 시퀀스와 같은 데이터베이스의 논리적 단위를 백업하는 것을 뜻한다. Tibero RDBMS에서는 이를 위해 tbExport와 tbImport 유틸리티를 제공하고 있다.

참고

tbExport와 tbImport에 대한 자세한 내용은 "Tibero RDBMS 유틸리티 안내서"를 참고한다.

- 물리적 백업

  물리적 백업이란 데이터베이스를 구성하는 물리적인 파일을 백업하는 것이며, 운영체제에서 파일 복사 명령(COPY)으로 백업하는 것을 뜻한다. 물리적 백업이 필요한 파일에는 데이터 파일과 아카이브 로그 파일이 있다.

온라인 로그 파일은 NOARCHIVELOG 모드에서 데이터베이스 전체를 백업하여 복구할 경우에만 의미가 있다.

주의

데이터베이스가 운영 중일 때 운영체제의 파일 복사 명령을 사용하는 것은 안전하지 않으므로 주의해야 한다.

6.3. 백업의 종류

다음은 Tibero RDBMS가 제공하는 백업의 종류이다.

- 모드별 백업

  데이터베이스를 ARCHIVELOG 모드로 운영할 때와 그렇지 않았을 때 사용할 수 있는 백업 방법이 다르다.

모드	설명
ARCHIVELOG 모드	온라인 백업 (Online Backup) 또는 Hot Backup이라 한다. 데이터베이스가 운영 중일 때 백업할 수 있다. 백업이 가능한 파일은 컨트롤 파일의 생성문과 데이터 파일, 아카이브 로그 파일 등이 있다. 복구는 백업 된 아카이브 로그 파일의 시점에 따라 데이터 파일의 백업 시점 전으로 복구할 수 있다.
NOARCHIVELOG 모드	오프라인 백업 (Offline Backup) 또는 Cold Backup이라 한다. 기본적으로 데이터베이스는 NOARCHIVELOG 모드이다. 데이터베이스를 구성하는 전체 파일은 반드시 Tibero RDBMS가 정상적으로 종료된 상태에서 백업한다. 백업 때문에 서비스가 중지되면 안 된다. 복구는 데이터베이스를 백업 받은 시점으로부터 복구할 수 있다.

- Consistent 백업

  Tibero RDBMS를 정상적으로 종료한 상태에서 백업하는 방법이다. 실행 예는 “6.4.2. Consistent 백업”을 참고한다.

- Inconsistent 백업

  Tibero RDBMS의 데이터베이스가 운영 중일 때 백업하거나 정상적으로 종료 되지 않은 상태에서 백업하는 방법이다. NOARCHIVELOG 모드에서는 이 방법을 권장하지 않는다. 실행 예는 “6.4.3. Inconsistent 백업”을 참고한다.

6.4. 백업의 실행

본 절에서는 Tibero RDBMS가 제공하는 논리적 및 물리적인 백업 방법에 따라 백업을 실행하는 예를 설명한다. 컨트롤 파일의 경우에는 항상 논리적인 백업만이 가능하기 때문에, 컨트롤 파일을 생성하는 생성문을 백업하는 방법만을 설명한다.

데이터 파일이나 로그 파일의 경우에는 데이터베이스 상태에 따라서 백업 받는 방법이 다르므로 이를 각각 데이터베이스가 운영 중인 경우(Inconsistent 백업)와 그렇지 않은 경우(Consistent 백업)로 나누어서 설명한다.

6.4.1. 컨트롤 파일의 백업

컨트롤 파일은 물리적인 백업을 지원하지 않는다. Tibero RDBMS에서는 컨트롤 파일의 논리적인 백업만을 지원한다. 데이터베이스의 구조에 변화가 일어난 경우에는 컨트롤 파일의 생성문을 백업하는 것이 좋다.

다음은 컨트롤 파일의 생성문을 tibero4/backup 디렉터리에 있는 ctrlfile1.sql 파일에 백업하는 예이다.

[예 6.1] 컨트롤 파일의 백업

```
SQL> alter database backup controlfile to trace as
       '/tibero4/backup/ctrlfile1.sql' reuse NORESETLOGS;

Altered.
```

생성된 ctrlfile1.sql 파일은 다음과 같은 내용을 포함한다.

[예 6.2] 백업된 컨트롤 파일의 생성문

```
CREATE CONTROLFILE REUSE DATABASE "t4db"
 LOGFILE
 GROUP 0 (
 '/disk1/log001.log',
 '/disk2/log002.log'
 ) SIZE 1M,
 GROUP 1 (
 '/disk1/log003.log',
 '/disk2/log004.log'
 ) SIZE 1M,
 GROUP 2 (
 '/disk1/log005.log',
 '/disk2/log006.log'
 ) SIZE 1M
 NORESETLOGS
 DATAFILE
 '/disk1/system001.tdf',
```

```
'/disk1/undo001.tdf'
NOARCHIVELOG
MAXLOGFILES 255
MAXLOGMEMBERS 8
MAXDATAFILES 100
CHARACTER SET MSWIN949
NATIONAL CHARACTER SET UTF16
;
```

RESETLOGS는 컨트롤 파일의 생성문에 지정한 대로 만들어진다. 이 생성문은 트레이스 파일을 생성한 후 RESETLOGS를 필요로 할 경우에 사용하며, RESETLOGS가 필요하지 않은 경우는 NORESETLOGS로 수정하여 컨트롤 파일을 생성할 때 적용할 수 있다.

참고

컨트롤 파일의 생성문에는 임시 파일을 생성하는 내용이 없다. 컨트롤 파일을 생성한 후 Tibero RDBMS를 기동하면 임시 파일은 존재하지 않는다.

컨트롤 파일을 새로 생성한 경우, 반드시 임시 파일을 추가해야 임시 파일을 이용한 기능을 사용할 수 있다.

생성된 컨트롤 파일은 $TB_SID.tip 파일에 경로를 설정한다.

[예 6.3] 컨트롤 파일의 경로 설정

```
CONTROL_FILES=$TB_HOME/database/$TB_SID/
```

다음과 같이 MOUNT나 OPEN 상태에서 컨트롤 파일의 목록을 조회하려면 동적 뷰 V$CONTROLFILE를 조회한다.

[예 6.4] 컨트롤 파일의 조회

```
SQL> SELECT NAME FROM V$CONTROLFILE;

NAME
------------------------------------------------------------
/disk1/c1.ctl
/disk2/c2.ctl

2 selected.
```

참고

컨트롤 파일을 다시 생성하기 위해서는 $TB_SID.tip 파일에 설정된 컨트롤 파일의 위치를 [예 6.4]의 질의 결과와 동일하게 설정한 후, CREATE CONTROLFILE 문을 실행해야 한다.

6.4.2. Consistent 백업

본 절에서는 Tibero RDBMS가 정상적으로 종료한 후에 백업하는 방법을 설명한다.

데이터 파일, 온라인 로그 파일의 조회

Consistent 백업을 실행 하기에 앞서 백업할 컨트롤 파일, 데이터 파일, 로그 파일을 조회한다.

다음은 MOUNT나 OPEN 상태에서 동적 뷰 **V$DATAFILE**를 통해 데이터 파일을 조회하는 방법이다. 여기서 MOUNT는 Tibero RDBMS의 인스턴스가 시작된 상태이며, OPEN은 컨트롤 파일에 정의한 모든 파일이 오픈된 상태를 의미한다.

[예 6.5] 데이터 파일의 조회

```
SQL> SELECT NAME FROM V$DATAFILE;

NAME
------------------------------------------------------------
/disk1/system001.tdf
/disk2/undo001.tdf
/disk3/user001.tdf

3 selected.
```

다음은 온라인 로그 파일을 MOUNT나 OPEN 상태에서 조회하는 방법이다.

[예 6.6] 온라인 로그 파일의 조회

```
SQL> SELECT MEMBER FROM V$LOGFILE;

MEMBER
------------------------------------------------------------
/disk1/log001.log
/disk2/log002.log
/disk2/log003.log
/disk3/log004.log
/disk3/log005.log
/disk1/log006.log

6 selected.
```

온라인 로그 파일은 ARCHIVELOG 모드가 아닌 경우에는 백업하지 않는 것이 좋다. ARCHIVELOG 모드에서는 온라인 로그 파일이 아카이브되기 때문에 아카이브된 파일을 백업하면 안 된다. 참고로 아카이브된 파일은 LOG_ARCHIVE_DEST 초기화 파라미터에 설정된 위치에 저장된다.

Tibero RDBMS의 종료

데이터베이스는 다음과 같이 NORMAL 모드로 종료하는 것이 좋다.

```
SQL> tbdown NORMAL;
Tibero instance was terminated.
```

데이터베이스가 정상적으로 종료되면, 운영체제별로 제공하는 파일 복사 명령을 사용하여 백업한다.

6.4.3. Inconsistent 백업

본 절에서는 Tibero RDBMS가 운영 중일 때 백업하는 방법을 설명한다.

데이터베이스가 운영 중이면 운영체제의 명령어를 사용해 데이터 파일을 복사하는 것은 안전하지 않다. 이러면 다음과 같은 문장을 실행하여 Tibero RDBMS에 백업의 시작과 종료를 통보해야 한다.

```
alter tablespace {tablespace name} begin backup
...
alter tablespace {tablespace name} end backup
```

begin backup과 end backup 문장 사이에는 해당 테이블스페이스의 변경 사항에 대한 로그가 늘어나기 때문에 데이터베이스에 부담이 가중되게 된다. begin backup을 시작한 이후에는 신속하게 백업을 완료하고 end backup 상태로 복귀시켜야 한다.

Inconsistent 백업의 전체 과정은 다음과 같다.

1. 먼저 백업할 테이블스페이스를 선정한다.

   [예 6.7] Inconsistent 백업 - 테이블스페이스의 선정

   ```
   SQL> select name,type from v$tablespace;

   NAME                           TYPE
   ------------------------------ ----
   SYSTEM                         DATA
   UNDO                           UNDO
   USER                           DATA
   TEMP                           TEMP

   3 selected.
   ```

2. 백업할 테이블스페이스에 속한 데이터 파일을 조회한 후, begin backup, end backup 명령어를 사용하여 백업을 수행한다.

   예를 들어, USER 테이블스페이스를 백업할 경우를 가정하고 수행한다.

[예 6.8] Inconsistent 백업 - begin backup, end backup 명령어의 사용

```
SQL> select f.name
     from v$tablespace t join v$datafile f on t.ts# = f.ts#
     where t.name = 'USER';

NAME
------------------------------------------------------------
/disk3/user001.tdf

1 selected

SQL> alter tablespace SYSTEM begin backup;

Altered.

SQL> !cp /disk3/user001.tdf /backup/
SQL> alter tablespace  SYSTEM end backup;

Altered.
```

6.5. 복구

Tibero RDBMS를 운영하다 보면, 예상치 못한 장애로 인해 정상적인 데이터베이스 운영이 어려운 상황이 발생할 수 있다. **복구**는 장애가 발생하는 경우 복원하는 일련의 과정이다.

복구를 하려면 백업된 데이터베이스가 있어야 한다.

Tibero RDBMS는 데이터베이스에서 일어나는 모든 변화를 로그 파일에 기록한다. 따라서, 백업 이후에 데이터베이스에 일어난 모든 변화에 대해서는 로그를 적용하면 복구할 수 있다. 로그 파일에는 커밋되지 않은 트랜잭션이 수정한 데이터도 포함되어 있다. 복구할 때 아카이브 로그 파일과 로그 파일 모두 사용할 수 있다.

복구 과정은 다음과 같이 두 가지 경우로 수행할 수 있다.

- 데이터 파일에 기록되지 않는 변화를 로그를 사용하여 적용하는 과정

  데이터 파일에 모든 로그의 변화를 기록하는 과정을 통해서 데이터베이스는 안정된 상태가 된다. 즉, 데이터베이스 운영상의 특정 시점까지 모든 작업이 반영되고 그 이후의 변화는 발생하지 않아야 한다.

  데이터베이스에 정상적인 복구가 이루어져 안정된 상태가 되어야만 기동할 수 있다.

- 커밋되지 않는 데이터로 복구하는 과정

  데이터베이스를 종료할 때 커밋하지 않은 트랜잭션이 수정한 내용으로 복구하는 과정이다.

6.5.1. 부트 모드별 복구

Tibero RDBMS는 부트 모드별로 발생되는 작업을 복구 측면에서 보면 다음과 같다.

- NOMOUNT 모드

  NOMOUNT 모드로는 언제나 복구할 수 있다. 이 모드에서는 데이터베이스와 컨트롤 파일을 생성할 수 있다. MOUNT 모드로 동작하기 위해서는 컨트롤 파일이 있어야 한다. 컨트롤 파일이 없거나 컨트롤 파일에 장애가 발생한 경우에는 NOMOUNT 모드로 동작하며, 컨트롤 파일을 생성하면 MOUNT 모드로 동작할 수 있다.

- MOUNT 모드

  MOUNT 모드에서는 데이터 파일, 온라인 로그 파일, 컨트롤 파일 사이의 상태를 검사하여 Tibero RDBMS를 기동할 준비를 한다. 세 파일이 모두 최신 상태이면 OPEN 모드로 동작할 수 있다. 파일에 물리적인 장애가 발생하였거나, 복원된 파일이라면 미디어 복구가 필요하며 MOUNT 모드로 동작한다. MOUNT 모드에서는 제한된 뷰의 조회가 가능하고, 미디어 복구를 수행할 수 있다.

- OPEN 모드

  Tibero RDBMS의 데이터 파일, 온라인 로그 파일, 컨트롤 파일이 일관성을 유지할 때에만 OPEN 모드로 동작할 수 있다. OPEN 모드에서 Tibero RDBMS는 세 파일을 열고 정상으로 동작한다. 일반 사용자는 데이터베이스를 이용할 수 있다.

6.5.2. 파손 복구

파손 복구(Crash Recovery)는 Tibero RDBMS를 운영하는 중에 정전, 시스템 이상, 강제 종료 등으로 데이터베이스가 비정상적으로 종료되었을 때 사용자의 명령 없이 자동으로 복구되는 것을 의미한다. 복구가 완료되면 Tibero RDBMS가 정상적으로 동작한다.

파손 복구는 온라인 로그 파일의 내용 중 아직 데이터 파일에 반영되지 않은 부분을 기록하여 Tibero RDBMS가 비정상적으로 종료되기 직전에 운영 시점의 상태로 복구하는 과정과 이러한 상태로 복구된 시점에서 커밋되지 않은 트랜잭션이 발생시킨 변화를 되돌리는 과정으로 나눌 수 있다.

파손 복구의 모든 과정은 파일의 손상이 없는 한, DBA의 도움 없이 자동으로 이루어진다.

6.5.3. 미디어 복구

Tibero RDBMS를 구성하는 파일이 물리적인 손상이나 정상적으로 동작할 수 없는 경우가 발생할 수 있다. 이러한 경우에 데이터베이스가 정상적으로 동작할 수 있도록 복구하는 과정이 **미디어 복구**(Media Recovery)이다.

미디어 복구 과정은 자동으로 이루어 지지 않는다. DBA가 상황을 파악해서 필요한 과정을 지시하는 일련의 작업이 필요하다. 복구 완료시점을 오류가 발생하기 이전의 가장 최근 시점까지로 할지, 과거의 특정

시점까지로 할지 여부에 따라서, **완전 복구**(Complete Recovery)와 **불완전 복구**(Incomplete Recovery)로 구분된다.

완전 복구

온라인 로그 파일의 가장 최근 로그까지 모두 반영하는 미디어 복구이다.

불완전 복구

온라인 로그 파일의 최근까지가 아닌 그 이전의 특정 시점까지 복구하는 것을 말한다. 불완전 복구 후에는 반드시 RESETLOGS 모드로 Tibero RDBMS를 기동해야 한다.

RESETLOGS는 온라인 로그 파일을 초기화 하는 것이며, 현재 온라인 로그 파일로 데이터베이스를 시작하지 않을 때 사용한다.

RESETLOGS가 필요한 경우는 다음과 같다.

- 불완전 미디어 복구를 한 경우
- RESETLOGS로 컨트롤 파일을 생성한 경우

RESETLOGS로 시작하면 새로운 데이터베이스가 만들어진 것과 같다. RESETLOGS 이전의 데이터 파일, 로그 파일과 RESETLOGS 이후의 파일은 서로 호환되지 않는다. RESETLOGS 이전의 백업 파일이나 로그 파일을 이용하여 RESETLOGS 이후로 복구할 수 없다. 또한 RESETLOGS 이후의 파일을 RESETLOGS 이전 상태로 불완전 복구를 하는 것도 불가능하다. 따라서, RESETLOGS 모드로 기동한 경우에는 반드시 새로 백업을 하기를 권장한다.

RESETLOGS로 데이터베이스를 기동하는 방법은 다음과 같다.

[예 6.9] RESETLOGS를 이용한 데이터베이스의 기동

```
$ tbboot -t RESETLOGS
```

미디어 복구 과정은 MOUNT 모드에서만 이루어진다. 백업된 파일을 사용하여 오류가 발생한 파일을 복원하는 과정과 복원된 파일을 백업한 시점으로부터 최근 혹은 특정 시점까지 반영되지 않은 변화를 로그 파일을 사용하여 복구하는 과정으로 나눌 수 있다. 단순한 복원 과정만으로는 Tibero RDBMS의 정상적인 운영이 불가능하다.

미디어 복구를 위해 장애가 발생한 파일을 찾아 복구해야 한다. 이를 위해 다음과 같은 뷰를 제공한다.

- V$LOGFILE
- V$CONTROLFILE
- V$LOG
- V$RECOVER_FILE

- V$RECOVERY_FILE_STATUS

미디어 복구는 로그 파일을 하나씩 데이터베이스에 순서대로 반영하여 진행한다. 데이터베이스는 현재 복구에 필요한 로그 파일만을 반영할 수 있다. 현재 필요한 로그 파일을 찾기 위해 **시퀀스 번호**가 사용된다.

시퀀스 번호는 데이터베이스가 생성된 이후로 만들어진 로그 파일의 일련 번호이며, 모든 로그 파일은 하나의 유일한 시퀀스 번호를 갖는다. 시퀀스 번호가 큰 로그 파일이 최근 로그 파일이다. 시퀀스 번호는 아카이브 로그 파일의 경우 파일 이름을 통해 알 수 있고, 온라인 로그 파일의 경우 **V$LOG** 뷰를 통해 알 수 있다.

6.6. 백업 및 복구 사례

본 절에서는 NOARCHIVELOG 모드와 ARCHIVELOG 모드로 나누어 백업 및 복구 사례를 설명한다.

6.6.1. NOARCHIVELOG 모드

ARCHIVELOG 모드를 사용하지 않으면 미디어 복구를 할 수 없다. 특정 데이터 파일이 손상된 경우 해당 파일의 백업으로부터 복구하는 과정만이 가능하다.

백업은 Tibero RDBMS를 종료하고 나서 Tibero RDBMS를 구성하는 모든 파일을 그대로 백업 받으면 된다. 단, 임시 파일은 백업 대상에 포함하지 않는다. 백업된 파일을 복구하여 백업받을 당시의 Tibero RDBMS의 상태로만 복구할 수 있으며, 그 이후에 발생한 작업은 모두 다시 수행해야 한다. 이 과정에서 발생할 수 있는 문제는 해당 디스크에 물리적 오류가 발생하여 처음 있던 위치로 복구할 수 없는 경우가 발생할 수 있다.

데이터 파일이나 로그 파일을 해당 위치에 복원할 수 없는 경우

데이터 파일이나 온라인 로그 파일을 해당 위치에 복원할 수 없는 경우에는 다른 위치로 복원을 한 후에 컨트롤 파일에 설정된 데이터 파일의 위치를 수정한다.

[예 6.10] 데이터 파일의 위치 수정

```
$ tbboot -t MOUNT
$ tbsql sys/tibero

SQL> select file#, name from v$datafile;
FILE#      NAME
---------  --------------------------------------------------
0          /disk1/database/system001.tdf
1          /disk1/database/undo001.tdf

SQL> alter database rename file '/disk1/database/system001.tdf'
```

```
       to '/disk2/database/system001.tdf';
Altered.

SQL> tbdown
Tibero instance was terminated.

SQL> quit
$ tbboot
```

컨트롤 파일을 해당 위치에 복구할 수 없는 경우

컨트롤 파일이 복구되어야 하는 위치에 복구될 수 없는 경우에는 백업을 다른 곳으로 복원하면 된다. 이 경우에는 $TB_SID.tip 파일에서 변경된 컨트롤 파일의 위치를 설정하면 된다.

6.6.2. ARCHIVELOG 모드

ARCHIVELOG 모드를 사용하는 경우에는 미디어 복구를 할 수 있다. 데이터 파일과 아카이브 로그 파일을 백업하면 된다.

예를 들어, 로그 파일에 장애가 발생하면 해당 로그 파일이 속한 로그 그룹의 다른 로그 파일을 복사하여 복구한다. 만약 로그 그룹에 속한 모든 로그 파일에 장애가 발생하면 현재 로그 그룹에 장애가 발생한 그룹을 제거한 후 다른 그룹을 추가하면 된다. 하지만, 해당 로그 그룹이 현재 로그 그룹이면 커밋 시점까지의 내용만 복구하는 불완전한 복구를 해야 한다.

시스템 테이블스페이스의 데이터 파일이 손상된 경우

시스템 테이블스페이스에 속한 데이터 파일은 Tibero RDBMS 운영에 필요한 구성요소이므로 반드시 복원되어야 한다. 특정 데이터 파일이 손상된 경우 해당 데이터 파일을 백업으로부터 복원하여 복구를 진행한다.

다음은 시스템 테이블스페이스의 데이터 파일을 복구하는 예이다.

[예 6.11] 시스템 테이블스페이스의 데이터 파일 복구

```
SQL> col tablespace format a10
SQL> col filename format a50
SQL> select a.name tablespace, b.name filename
     from v$tablespace a join v$datafile b on (a.ts# = b.ts#);

TABLESPACE FILENAME
---------- --------------------------------------------------
SYSTEM     /disk1/database/system00.tdf
UNDO       /disk1/database/undo001.tdf
ZZ         /disk2/database/tmp001.tdf
```

```
3 selected.

SQL> tbdown
SQL> quit
$ rm -f /disk1/database/system00.tdf
$ tbboot
listener port = 8629

******************************************************
* Critical Warning : Raise svmode failed. The reason is
* TBR-1024 : Database needs media recovery: open \
failed(/disk1/database/system001.tdf)
* Current server mode is MOUNT.
******************************************************

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started suspended at MOUNT mode.

$ tbsql sys/tibero
tbSQL 4 SP1

Copyright (c) 2001-2009 Tibero Corporation. All rights reserved.

Connected to Tibero.

SQL> select * from v$recover_file;

FILE# ERROR                      CHANGE# TIME
---------- ---------------------- ---------- ----------
0 FILE MISSING            4294965795 2006/11/14

1 selected.

SQL> !cp /backup/system00.tdf /disk1/database/system00.tdf
SQL> select * from v$recover_file;

FILE# ERROR                      CHANGE# TIME
---------- ---------------------- ---------- ----------
0 WRONG FILE CREATE       4294965795 2006/11/14

1 selected.

SQL> alter database recover automatic database;
```

```
Altered.

SQL> select * from v$recover_file;

FILE# ERROR                       CHANGE# TIME
----------- ----------------------- ---------- ----------

0 selected.

SQL> tbdown
Tibero instance was terminated.
SQL> quit
$ tbboot
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NORMAL mode).
```

비시스템 테이블스페이스의 데이터 파일이 손상된 경우

비시스템 테이블스페이스의 데이터 파일이 손상된 경우, 해당 데이터 파일을 복원하여 복구하는 방법과 해당 테이블스페이스를 제거하는 방법이 있다. 시스템 테이블스페이스나 Undo 테이블스페이스가 아닌 경우에는 장애가 발생한 테이블스페이스만 없어질 뿐 Tibero RDMBS 운영 자체에는 문제가 발생하지 않는다. 데이터 파일을 복구하는 방법은 [예 6.11]과 동일한 방법으로 수행하면 된다.

다음은 테이블스페이스의 내용이 별로 중요하지 않거나, 없어져도 문제가 없는 경우에 제거하는 예이다.

[예 6.12] 장애가 발생한 데이터 파일이 속한 테이블스페이스의 제거

```
$ tbboot
******************************************************
* Critical Warning : Raise svmode failed. The reason is
* TBR-1024 : Database needs media recovery: open \
failed(/disk1/test1.tdf).
* Current server mode is MOUNT.
******************************************************

$ tbsql sys/tibero
SQL> select * from v$recover_file;

FILE# ERROR                       CHANGE# TIME
```

```
---------- ---------------------- ---------- ----------
2 FILE MISSING            4294965795 2006/11/14

1 selected.

SQL> alter database datafile 2 offline for drop;

Altered.

SQL> tbdown
Tibero instance was terminated.
SQL> tblistener: connection with tbsvr is closed.
$ tbboot -t resetlogs
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NORMAL mode).

$ tbsql sys/tibero

SQL> drop tablespace test_ts including contents and datafiles;

Dropped.

SQL> select file#, name from v$datafile;

FILE#      NAME
---------- --------------------
0          /disk1/system001.tdf
1          /disk1/undo001.tdf

2 selected.
```

백업이 안된 데이터 파일에 장애가 발생한 경우

ALTER DATABASE CREATE DATAFILE 문을 사용하여 데이터 파일을 새로 생성한 후 복구한다.

다음은 데이터 파일을 생성하여 복구하는 예이다.

[예 6.13] 데이터 파일의 생성을 통한 복구

```
SQL> create tablespace z datafile '/disk1/z.tdf' size 10m;

Created.
```

```
SQL> create table z (i number) tablespace z;

Created.

SQL> insert into z values(1);

1 inserted.

SQL> commit;

Commit succeeded.

SQL> tbdown
Tibero instance was terminated.
SQL> quit
$ rm -f /disk1/z.tdf
$ tbboot
******************************************************
* Critical Warning : Raise svmode failed. The reason is
* TBR-1024 : Database needs media recovery: open \
failed(/disk1/z.tdf).
* Current server mode is MOUNT.
******************************************************

$ tbsql sys/tibero

SQL> select * from v$recover_file;

FILE# ERROR                      CHANGE# TIME
---------- ---------------------- ---------- ----------
2 FILE MISSING            4294970441 2006/11/14

1 selected.

SQL> alter database create datafile 2;

Altered.

SQL> select * from v$recover_file;

FILE# ERROR                      CHANGE# TIME
---------- ---------------------- ---------- ----------
2 WRONG FILE CREATE       4294970441 2006/11/14

1 selected.
```

```
SQL> alter database recover automatic database;

Altered.

SQL> select * from v$recover_file;

FILE# ERROR                       CHANGE# TIME
---------- ---------------------- ---------- ----------

0 selected.

SQL> tbdown

$ tbboot
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NORMAL mode).

$ tbsql sys/tibero

SQL> select * from z;

I
----------
1

1 selected.
```

컨트롤 파일에 장애가 발생한 경우

컨트롤 파일의 경우 장애에 대비하여 여러 개의 복사본을 유지하는 미러링(mirroring) 방식을 사용하는 것이 좋다. 미러링된 파일 중 장애가 발생하지 않은 파일이 있으면 이를 사용하여 장애가 발생한 파일에 복사한 후 Tibero RDBMS를 다시 기동하면 된다.

모든 컨트롤 파일이 장애가 발생한 경우에는 **CREATE CONTROLFILE** 문을 사용하여 컨트롤 파일을 새로 생성해야 한다. 다음은 미러링된 파일 중에 장애가 발생하지 않은 파일을 복사하여 컨트롤 파일의 장애를 해결하는 예이다.

[예 6.14] 미러링을 사용한 컨트롤 파일의 복구

```
$ grep "CONTROL_FILES" $TB_HOME/config/$TB_SID.tip
CONTROL_FILES=/disk1/ctl1.ctl, /disk2/ctl2.ctl
```

```
$ tbboot
*********************************************************
* Critical Warning : Raise svmode failed. The reason is
* TBR-1003 : Cannot open file: /disk2/ctl2.ctl.
* Current server mode is NOMOUNT.
*********************************************************

SQL> tbdown
Tibero instance was terminated.

$ ls /disk2/*.ctl
$ cp /dist1/ctl1.ctl /disk2/ctl2.ctl
$ ls /disk2/*.ctl
ctl2.ctl

$ tbboot
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NORMAL mode).
```

위의 예제에서는 $TB_SID.tip 파일에 설정되어 있는 컨트롤 파일 두 개 중 ctl2.ctl 파일에 장애가 발생하여 Tibero RDBMS를 시작하지 못하는 경우이다. 컨트롤 파일이 여러 파일로 미러링된 경우에는 정상적인 파일을 복사하면 Tibero RDBMS를 기동할 수 있다.

다음은 컨트롤 파일을 생성한 후, RESETLOGS로 데이터베이스를 시작하는 예이다. 단, 컨트롤 파일을 생성할 때 임시 파일이 존재하지 않으므로 반드시 추가해야 한다.

[예 6.15] 컨트롤 파일의 생성을 통한 복구

```
$ tbboot -t NOMOUNT
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NOMOUNT mode).

$ tbsql sys/tibero

SQL> CREATE CONTROLFILE REUSE DATABASE "t3db"
```

```
        LOGFILE
          GROUP 0 (
                    '/disk1/log001.log',
                    '/disk2/log002.log'
                  ) SIZE 1M,
          GROUP 1 (
                    '/disk1/log003.log',
                    '/disk2/log004.log'
                  ) SIZE 1M,
          GROUP 2 (
                    '/disk1/log005.log',
                    '/disk2/log006.log'
                  ) SIZE 1M
        RESETLOGS
        DATAFILE
                '/disk1/system001.tdf',
                '/disk1/undo001.tdf'
        NOARCHIVELOG
        MAXLOGFILES 255
        MAXLOGMEMBERS 8
        MAXDATAFILES 100
    ;
Created.

SQL> tbdown
Tibero instance was terminated.

SQL> quit
$ tbboot -t RESETLOGS
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started suspended at NORMAL mode.

$ tbsql sys/tibero

SQL> select file_id, tablespace_name from dba_temp_files;

   FILE_ID TABLESPACE_NAME
---------- ------------------------------

0 selected.

SQL> alter tablespace temp add tempfile '/disk1/temp01.tdf'
```

```
       size 10m reuse;

Altered.

SQL> select file_id, tablespace_name from dba_temp_files;

   FILE_ID TABLESPACE_NAME
---------- ------------------------------
         0 TEMP

1 selected.
```

로그 파일에 장애가 발생한 경우

로그 파일에 장애가 발생한 경우에는 해당 로그 파일이 속한 그룹의 다른 로그 파일을 복사하여 복원을 시도한다. 로그 그룹에 속한 모든 로그 파일이 장애가 발생한 경우에는 현재 로그 그룹이 장애가 발생한 그룹을 제거한 후 다른 그룹을 추가하면 된다. 하지만 해당 로그 그룹이 현재의 로그 그룹이라면 불완전 복구를 통해 복구해야 한다.

사용자의 실수로 테이블의 데이터가 잘못된 경우

사용자가 실수로 테이블의 데이터를 잘못 조작한 경우에는 불완전 복구를 통해 복원한다.

다음은 사용자의 잘못된 조작으로 테이블 emp의 모든 salary의 데이터가 갱신되었다고 가정하고, 백업된 파일을 이용하여 실수 이전의 시점으로 불완전 복구를 수행하는 예이다.

[예 6.16] 불완전 복구를 통한 테이블의 데이터 복원

```
$ tbsql sys/tibero

SQL> select * from emp;

ENO ENAME          SALARY
---------- ---------- ----------
1 Peter            7200
2 Smith            6480
3 Susan            3240

3 selected.

SQL> !date
Wed Nov 15 13:33:28 KST 2006

SQL> update emp set salary = salary * 0.9 where eno <> 0;
```

```
3 updated.

SQL> commit;

Commit succeeded.

SQL> select * from emp;

ENO ENAME          SALARY
---------- ---------- ----------
1 Peter             6480
2 Smith             5832
3 Susan             2916

3 selected.

SQL> tbdown
Tibero instance was terminated.

SQL>
$ cp -f back/*.tdf .
$ tbboot -t mount
port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started suspended at MOUNT mode.

$ tbsql sys/tibero

SQL> alter database recover automatic database
       until time '2006-11-15 13:32:00';

Altered.

SQL> tbdown.

$ tbboot -t RESETLOGS
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started suspended at NORMAL mode.
```

```
$ tbdown
Tibero instance was terminated.

$ tbboot
listener port = 8629

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NORMAL mode).

$ tbsql sys/tibero

SQL> select * from emp;

ENO ENAME          SALARY
---------- ---------- ----------
1 Peter            7200
2 Smith            6480
3 Susan            3240

3 selected.

SQL>
Disconnected.
```

6.7. 복구 관리자

Tibero RDBMS는 다양한 백업 및 복구 시나리오를 제공한다. 숙련된 데이터베이스 관리자라면 상황에 맞는 적절한 방법을 선택하고 활용할 수 있을 것이다. 허나 너무 다양한 기능을 제공함으로써 오히려 사용자에게 혼란을 줄 수도 있다. 이러한 면을 보완하기 위하여 Tibero RDBMS는 복구 관리자(이하 RMGR)를 제공한다.)

복구 관리자 기능

RMGR는 다양한 백업/복구 시나리오를 가능하게 하도록 구성되어있다. Tibero에서 제공되는 RMGR 기능은 다음과 같다.

- Online Full Backup

  Tibero 데이터베이스에 속한 전체 데이터 파일을 온라인 백업한다. 온라인 백업을 위해서는 데이터베이스가 ARCHIVELOG 모드이여야 한다. RMGR은 자동으로 데이터베이스의 Begin Backup 기능을 사용하여 모든 테이블스페이스를 Hot Backup 상태로 만들고 백업을 진행한다.

백업을 완료하면 데이터베이스의 End Backup 기능을 사용하여 모든 테이블스페이스를 Hot Backup 상태로부터 해제한다. 백업할 데이터파일 역시 V$DATAFILE을 조회하여 자동으로 결정해 준다.

- Incremental Full Backup

  RMGR를 통해 온라인 백업을 받았으면 이를 이용하여 Incremental Backup을 할 수 있다. Incremental Backup이란 백업을 받을 때 전체 파일을 받는 것이 아니라 이전 백업과의 차이만을 기록하는 방식으로 백업에 소모되는 디스크 공간을 획기적으로 줄일 수 있다.

  Incremental Full Backup을 하려면 이전에 RMGR를 통해 Online Full Backup을 받았어야 한다. 현재 데이터베이스와 백업본과의 차이를 구하여 백업 파일을 만든다. 이러한 기능은 RMGR를 통해서만 사용할 수 있다.

- Automatic Recovery

  RMGR로 만들어진 백업본을 이용하여 자동 Recovery를 진행한다. 이를 위해서는 백업을 만들 당시에 만들어진 RMGR info file이 필요하다. Online Full Backup/Incremental Full Backup 정보를 분석하여 자동으로 Merge 후 Recovery해준다.

복구 관리자 정보 파일

RMGR로 백업을 진행하면 RMGR Info라는 텍스트 파일이 생기게 된다. RMGR로 받은 백업에 대한 정보가 담겨져 있으며, Incremental Full Backup시에 추가로 계속 남게 된다. 혹시라도 이 파일을 분실하게 되면 Incremental Full Backup으로 남은 백업본은 복구할 수 없으니 주의해야 한다.

복구 관리자 옵션

RMGR는 Shell Command로 실행되며, 다양한 옵션을 지정하여 원하는 기능을 사용할 수 있다.

옵션	**설명**
backup	RMGR를 통해 백업을 진행한다.
recovery	RMGR로 받아놓은 백업본을 이용하여 복구를 진행한다.
-userid	데이터베이스에 접속할 사용자명과 패스워드 및 SID를 지정한다. -userid USERID[[/PASSWD][@SID]] 형식으로 지정한다.
-v, --verbose	RMGR의 진행상황을 자세하게 출력해준다.
-h, --help	RMGR의 옵션 사용법을 출력해준다.
-i, --incremental	Incremental Full Backup을 한다.

옵션	설명
-b	RMGR info 파일의 위치를 지정한다. RMGR info 파일은 이전에 RMGR를 통해 백업받은 디렉터리에 생성된다.
-o	백업받을 디렉터리를 지정한다.

복구 관리자를 이용한 백업/복구 예제

복구 관리자를 이용한 백업/복구 과정을 살펴보겠다. 복구 관리자는 대부분의 과정을 자동으로 진행해 주게 되므로 처음에 실행시키고 나면 특별히 관리해야할 과정이 없다. 실행할 때 어떤일을 할지 지정하여 실행한 후 실행되는 과정을 볼 수 있고, RMGR info 파일을 통해 어떤 백업을 받았는지를 알 수 있다.

참고

데이터 파일이 raw device 파일인 경우에도 복구 관리자를 이용한 백업/복구가 가능하다.

[예 6.17] Online Full Backup 시나리오

```
$ tbrmgr backup -o /home/winds/backup  -v
============================================================
= Recovery Manager(RMGR) starts                            =
=                                                          =
= Tibero, Co. Copyright(C) 2010-2019, All rights reserved. =
============================================================
==========================================
  RMGR - ONLINE backup
==========================================
DB connected
DBNAME: tibero
archive log check succeeded
Database begin backup succeeded

RMGR info file created: /home/winds/backup/rmgr.inf

 RMGR - ONLINE backup target datafile list
 total datafile count: 3
 /home/winds/tibero4_trunk/database/tibero/system001.dtf
 /home/winds/tibero4_trunk/database/tibero/undo001.dtf
 /home/winds/tibero4_trunk/database/tibero/usr001.dtf


    ORG DF: /home/winds/tibero4_trunk/database/tibero/system001.dtf
    FULL BACKUP: /home/winds/backup/system001.dtf.000
    CUR BACKUP: /home/winds/backup/system001.dtf.000
```

```
    BACKUP CNT: 1

RUN: /home/winds/tibero4_trunk/bin/tbrg backup -v \
-o /home/winds/backup/system001.dtf.000 \
/home/winds/tibero4_trunk/database/tibero/system001.dtf
Backup file
 type: full backup
 source file : /home/winds/tibero4_trunk/database/tibero/system001.dtf
 output file : /home/winds/backup/system001.dtf.000
Analyze source file
 blksize   : 8192 bytes
 file size : 5376 blocks
 100.00% |=======================================>|   5376/5376   blks 0.12s
Synchronizing...
5376 blocks copied.

    ORG DF: /home/winds/tibero4_trunk/database/tibero/undo001.dtf
    FULL BACKUP: /home/winds/backup/undo001.dtf.000
    CUR BACKUP: /home/winds/backup/undo001.dtf.000
    BACKUP CNT: 1

RUN: /home/winds/tibero4_trunk/bin/tbrg backup -v \
-o /home/winds/backup/undo001.dtf.000 \
/home/winds/tibero4_trunk/database/tibero/undo001.dtf
Backup file
 type: full backup
 source file : /home/winds/tibero4_trunk/database/tibero/undo001.dtf
 output file : /home/winds/backup/undo001.dtf.000
Analyze source file
 blksize   : 8192 bytes
 file size : 1280 blocks
 100.00% |=======================================>|   1280/1280   blks 0.02s
Synchronizing...
1280 blocks copied.

    ORG DF: /home/winds/tibero4_trunk/database/tibero/usr001.dtf
    FULL BACKUP: /home/winds/backup/usr001.dtf.000
    CUR BACKUP: /home/winds/backup/usr001.dtf.000
    BACKUP CNT: 1

RUN: /home/winds/tibero4_trunk/bin/tbrg backup -v \
-o /home/winds/backup/usr001.dtf.000 \
/home/winds/tibero4_trunk/database/tibero/usr001.dtf
Backup file
 type: full backup
 source file : /home/winds/tibero4_trunk/database/tibero/usr001.dtf
 output file : /home/winds/backup/usr001.dtf.000
```

```
Analyze source file
 blksize   : 8192 bytes
 file size : 256 blocks
 100.00% |=======================================>|    256/256    blks 0.00s
Synchronizing...
256 blocks copied.
 full backup succeeded
Database end backup succeeded.
```

[예 6.18] Incremental Full Backup 시나리오

```
$ tbrmgr backup -i -o /home/winds/backup2 -b /home/winds/backup/rmgr.inf -v
============================================================
= Recovery Manager(RMGR) starts                            =
=                                                          =
= Tibero, Co. Copyright(C) 2010-2019, All rights reserved. =
============================================================
==========================================
  RMGR - INCREMENTAL backup
==========================================
DB connected
DBNAME: tibero
archive log check succeeded
Database begin backup succeeded
rmgr info file: /home/winds/backup/rmgr.inf

RMGR INFO DATAFILE

    ORG DF: /home/winds/tibero4_trunk/database/tibero/system001.dtf
    FULL BACKUP: /home/winds/backup/system001.dtf.000
    CUR BACKUP: /home/winds/backup/system001.dtf.000
    BACKUP CNT: 1

    ORG DF: /home/winds/tibero4_trunk/database/tibero/undo001.dtf
    FULL BACKUP: /home/winds/backup/undo001.dtf.000
    CUR BACKUP: /home/winds/backup/undo001.dtf.000
    BACKUP CNT: 1

    ORG DF: /home/winds/tibero4_trunk/database/tibero/usr001.dtf
    FULL BACKUP: /home/winds/backup/usr001.dtf.000
    CUR BACKUP: /home/winds/backup/usr001.dtf.000
    BACKUP CNT: 1

RMGR info file load finished: /home/winds/backup/rmgr.inf

 RMGR - INCREMENTAL backup target datafile list
```

```
 total datafile count: 3
 /home/winds/tibero4_trunk/database/tibero/system001.dtf
 /home/winds/tibero4_trunk/database/tibero/undo001.dtf
 /home/winds/tibero4_trunk/database/tibero/usr001.dtf


    ORG DF: /home/winds/tibero4_trunk/database/tibero/system001.dtf
    FULL BACKUP: /home/winds/backup/system001.dtf.000
    CUR BACKUP: /home/winds/backup2/system001.dtf.002
    BACKUP CNT: 2

RUN: /home/winds/tibero4_trunk/bin/tbrg backup  -i -b \
/home/winds/backup/system001.dtf.000 -o /home/winds/backup2/system001.dtf.002 \
/home/winds/tibero4_trunk/database/tibero/system001.dtf
 100.00% |=======================================>|   5376/5376   blks 0.02s
0 blocks copied.

    ORG DF: /home/winds/tibero4_trunk/database/tibero/undo001.dtf
    FULL BACKUP: /home/winds/backup/undo001.dtf.000
    CUR BACKUP: /home/winds/backup2/undo001.dtf.002
    BACKUP CNT: 2

RUN: /home/winds/tibero4_trunk/bin/tbrg backup  -i -b \
/home/winds/backup/undo001.dtf.000 -o /home/winds/backup2/undo001.dtf.002 \
/home/winds/tibero4_trunk/database/tibero/undo001.dtf
 100.00% |=======================================>|   1280/1280   blks 0.01s
Synchronizing...
10 blocks copied.

    ORG DF: /home/winds/tibero4_trunk/database/tibero/usr001.dtf
    FULL BACKUP: /home/winds/backup/usr001.dtf.000
    CUR BACKUP: /home/winds/backup2/usr001.dtf.002
    BACKUP CNT: 2

RUN: /home/winds/tibero4_trunk/bin/tbrg backup  -i -b \
/home/winds/backup/usr001.dtf.000 -o /home/winds/backup2/usr001.dtf.002 \
/home/winds/tibero4_trunk/database/tibero/usr001.dtf
 100.00% |=======================================>|    256/256    blks 0.00s
0 blocks copied.
 full backup succeeded
Database end backup succeeded.
```

[예 6.19] Recovery with Online Full Backup 시나리오

```
$ tbrmgr recovery -b /home/winds/backup/rmgr.inf
============================================================
```

```
= Recovery Manager(RMGR) starts                              =
=                                                            =
= Tibero, Co. Copyright(C) 2010-2019, All rights reserved. =
============================================================
rmgr info file: /home/winds/backup/rmgr.inf
===========================================
  RMGR - recovery
===========================================

Tibero instance terminated (NORMAL mode).

listener port = 15048

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (MOUNT mode).

DB connected
Database automatic recovery succeeded
sess: 18 user: SYS

There are active session(s).
 1. [W]ait until sessions are closed.
 2. Shutdown [I]mmediately.
 3. [Q]uit without shutting down.
Select action. (Default: 1): 2

Tibero instance terminated (IMMEDIATE mode).

listener port = 15048

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NORMAL mode).
```

[예 6.20] Recovery with Incremental Full Backup 시나리오

```
$ tbrmgr recovery -b /home/winds/backup/rmgr.inf
============================================================
= Recovery Manager(RMGR) starts                              =
=                                                            =
= Tibero, Co. Copyright(C) 2010-2019, All rights reserved. =
```

```
============================================================
rmgr info file: /home/winds/backup/rmgr.inf
============================================
  RMGR - recovery
============================================

Tibero instance terminated (NORMAL mode).

base file : /home/winds/backup/system001.dtf.000 (uid=37839.8628)
output file : /home/winds/tibero4_trunk/database/tibero/system001.dtf
copy base file
 100.00% |=======================================>|   5376/5376   blks 0.12s
merging /home/winds/backup2/system001.dtf.002 100.00%
1 files Merged.base file : /home/winds/backup/undo001.dtf.000 (uid=37839.8628)
output file : /home/winds/tibero4_trunk/database/tibero/undo001.dtf
copy base file
 100.00% |=======================================>|   1280/1280   blks 0.02s
merging /home/winds/backup2/undo001.dtf.002 100.00%
1 files Merged.base file : /home/winds/backup/usr001.dtf.000 (uid=37839.8628)
output file : /home/winds/tibero4_trunk/database/tibero/usr001.dtf
copy base file
 100.00% |=======================================>|    256/256    blks 0.01s
merging /home/winds/backup2/usr001.dtf.002 100.00%
1 files Merged.listener port = 15048

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (MOUNT mode).

DB connected
Database automatic recovery succeeded
sess: 18 user: SYS

There are active session(s).
 1. [W]ait until sessions are closed.
 2. Shutdown [I]mmediately.
 3. [Q]uit without shutting down.
Select action. (Default: 1): 2

Tibero instance terminated (IMMEDIATE mode).

listener port = 15048

Tibero RDBMS 4SP1
```

```
Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NORMAL mode).
```

제7장 분산 트랜잭션

하나의 데이터베이스 인스턴스 내에서 한 트랜잭션으로 묶인 SQL 문장이 모두 커밋되거나 롤백되듯이 네트워크로 연결된 여러 개의 데이터베이스 인스턴스가 참여하는 트랜잭션에서도 각각 다른 데이터베이스 인스턴스에서 수행한 SQL 문장이 모두 동시에 커밋되거나 롤백될 수 있는 방법이 필요하다.

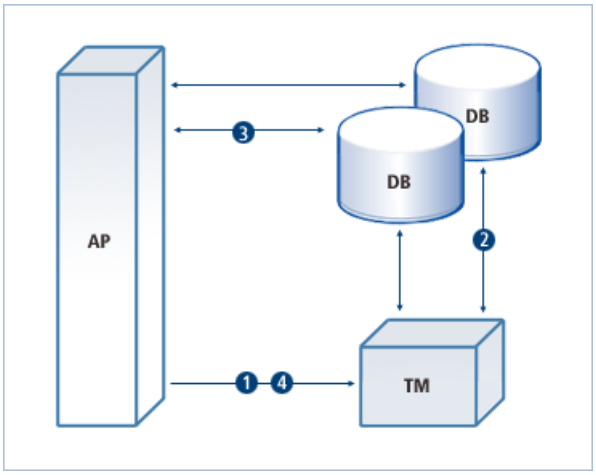
이렇게 여러 개의 노드 또는 다른 종류의 데이터베이스가 참여하는 하나의 트랜잭션을 **분산 트랜잭션** (distributed transaction)이라고 한다. Tibero RDBMS에서는 분산 트랜잭션을 처리하기 위해 **XA**와 **데이터베이스 링크(DBLink)**를 통해 지원한다.

7.1. XA

Tibero RDBMS는 X/Open DTP(Distributed Transaction Processing) 규약의 XA를 지원한다. XA는 2PC(Two-phase commit)를 이용하여 분산 트랜잭션을 처리한다.

다음은 XA가 어떻게 동작하는지를 나타내는 그림이다.

[그림 7.1] XA의 동작 (AP, TM, DB의 상호 작용)



1. 일반적으로 XA는 트랜잭션 매니저 (TM: Transaction Manager, 이하 TM)에 의해 코디네이트된다. 가장 먼저 애플리케이션 프로그램 (AP: Application Program, 이하 AP)은 TM에 분산 트랜잭션의 시작을 알린다.

2. TM은 AP의 요청을 받고 어떤 데이터베이스의 노드가 해당 분산 트랜잭션에 참여하는지 확인한다. 그 다음 각 데이터베이스 노드에 분산 트랜잭션의 시작을 알린다.

   각 데이터베이스 노드에 분산 트랜잭션의 시작을 알릴 때, TM은 내부에 고유한 트랜잭션 ID(이하 XID)를 만들어서 함께 전달한다. 그러면 각 데이터베이스 노드는 이 XID와 관련된 분산 트랜잭션을 시작한다. 앞으로 AP로부터 들어오는 요청은 해당 분산 트랜잭션에 대한 작업이라고 인식한다.

3. AP는 각 데이터베이스에 SQL 문장을 전달함으로써 필요한 작업을 진행한다.

   이때 각 데이터베이스는 전달 받은 요청을 해당 XID와 관련된 작업이라고 인지하고 SQL 문장을 실행한다.

4. 모든 작업이 완료되면 AP는 TM에 분산 트랜잭션의 종료를 알린다.

   TM은 해당 XID로 분산 트랜잭션에 참여했던 각 데이터베이스 노드에 커밋과 롤백을 동시에 하도록 지시한다. 일부 데이터베이스는 커밋을 하고, 일부 데이터베이스는 롤백을 하는 상황이 벌어지지 않도록 TM은 **Two-phase commit mechanism**을 통해 수행한다.

7.2. Two-phase commit mechanism

Two-phase commit mechanism은 분산 컴퓨팅 환경에서 트랜잭션에 참여하는 모든 데이터베이스가 정상적으로 수정되었음을 보장하는 두 단계 커밋 프로토콜이다. 분산 트랜잭션에 참여한 모든 데이터베이스가 모두 함께 커밋되거나 롤백되는 것을 보장한다.

Two-phase commit mechanism은 다음과 같이 두 단계로 작업이 이루어진다.

- First Phase (또는 prepare phase)

  First Phase는 각 데이터베이스 노드에 커밋을 하기 위한 준비 요청 단계이다.

  다음은 First Phase가 실행되는 과정이다.

  1. TM은 각 데이터베이스 노드에 커밋을 준비하라는 **prepare** 메시지를 보낸다.

  2. 요청을 받은 각 데이터베이스는 커밋을 준비한다.

     커밋을 하기 위한 준비 작업에는 필요한 리소스에 잠금(Lock)을 설정하거나 로그 파일을 저장하는 작업 등이 있다.

  3. 각 데이터베이스는 커밋 준비 여부에 따라 TM에 성공 또는 실패 여부를 알린다.

     커밋 준비가 모두 끝나면 prepare가 성공한 것이고, 커밋 준비를 실패하면 prepare가 실패한 것이다.

- Second Phase (또는 commit phase)

  TM은 참여한 모든 데이터베이스 노드로부터 **prepare**의 완료 메시지를 받을 때까지 대기한다.

  이 단계에서는 전달 받은 **prepare**의 메시지에 따라 해당 결과가 다르다.

구분	설명
롤백	한 데이터베이스 노드라도 **prepare ok** 메시지를 받지 않으면, 이 트랜잭션은 커밋할 수 없다고 판단하고 모든 데이터베이스 노드에 **rollback** 메시지를 보내고 해당 작업을 롤백한다.

구분	설명
커밋	모든 데이터베이스 노드로부터 **prepare ok** 메시지를 받으면 다시 모든 데이터베이스 노드에 **commit** 메시지를 보내고 모든 작업을 커밋한다.

7.3. XA의 In-doubt 트랜잭션 처리

Two-phase commit mechanism에 의해 첫 번째 prepare 메시지를 받으면 데이터베이스는 분산 트랜잭션에 해당하는 리소스를 잠금 처리하거나 로그를 남김으로써 커밋할 준비를 한다. 그런데 **prepare**까지 마친 상태에서 네트워크의 이상으로 다음 메시지(커밋 혹은 롤백)를 받지 못하는 경우가 발생할 수 있다.

데이터베이스는 해당 트랜잭션을 커밋해야 할지 롤백해야 할지 판단할 수 없다. 따라서 다음 메시지가 올 때까지 **prepare**된 리소스에 잠금 처리를 한 채로 기다리게 되는 데, 이러한 경우를 **In-doubt 트랜잭션**이라고 한다.

일반적으로 네트워크 또는 TM 측의 문제가 해결된다면 복구되는 즉시 TM은 In-doubt 트랜잭션에 커밋 또는 롤백 메시지를 다시 보낸다. 하지만 In-doubt 트랜잭션이 잡고 있는 리소스가 급하게 반환 되어야 하는 상황이 발생한다면 DBA는 임의로 In-doubt 트랜잭션을 커밋 또는 롤백 시킴으로써 해당 리소스를 반환할 수 있다. 이러한 경우는 DBA의 판단에 의해 결정되므로 이후에 TM으로부터 전달되는 커밋 또는 롤백 메시지가 DBA가 결정한 판단과 다르다면 전체 분산 트랜잭션이 일부 커밋되거나 롤백되는 현상이 발생할 수 있다. 따라서 전체 분산 트랜잭션의 일관성을 위해 TM의 다음 요청을 기다려야 한다.

이러한 문제를 감수하더라도 In-doubt 트랜잭션을 처리해야 하는 경우가 발생한다면 DBA_2PC_PENDING 뷰를 이용하여 이를 해결한다.

7.3.1. DBA_2PC_PENDING

DBA_2PC_PENDING는 현재 정체되고 있는 XA 트랜잭션 브랜치 (XA transaction branch)의 정보를 보여주는 뷰이다.

다음은 XA 트랜잭션 브랜치 (XA transaction branch)의 정보를 조회하는 예이다. 본 예제에서는 XID와 FAIL_TIME 정보를 이용하여 커밋과 롤백을 수행할 브랜치를 선택한다.

[예 7.1] DBA_2PC_PENDING 뷰의 조회

```
SQL> SELECT LOCAL_TRAN_ID, XID, STATUS, FAIL_TIME, FORCE_TIME
    FROM DBA_2PC_PENDING;

LOCAL_TRAN_ID
------------------------------------
XID
------------------------------------
STATUS          FAIL_TIME  FORCE_TIME
--------------- ---------- ----------
2.16.18
```

```
1.1000.1000
PREPARED        2006/12/11

1 selected.
```

DBA는 다음과 같이 원하는 XA 트랜잭션 브랜치에 커밋 명령을 실행할 수 있다. 그러면 해당 XA 트랜잭션 브랜치에서 잡고 있던 리소스는 반환되고, 해당 트랜잭션은 커밋된다.

```
SQL> commit force '2.16.18';

Commit succeeded.
```

DBA는 강제 커밋 (commit force)을 통해 롤백할 수 있다.

```
SQL> rollback force '2.16.18';

Rollback succeeded.
```

TM에 의한 정식 커밋이 아니고 DBA의 임의의 결정으로 커밋을 실행하면 해당 XA 트랜잭션 브랜치의 정보는 그대로 남아 있는다.

다음과 같이 **FORCE_TIME**에 DBA가 강제로 커밋한 시간이 남아 있음을 알 수 있다.

```
SQL> SELECT LOCAL_TRAN_ID, XID, STATUS, FAIL_TIME, FORCE_TIME
    FROM DBA_2PC_PENDING;

LOCAL_TRAN_ID
------------------------------------
XID
------------------------------------
STATUS          FAIL_TIME  FORCE_TIME
--------------- ---------- ----------

1.1000.1000
FORCED_COMMIT   2006/12/11 2006/12/12


1 selected
```

해당 XA 트랜잭션 브랜치의 정보는 TM이 **xa_forget**을 이용하여 더 이상 XA 트랜잭션 브랜치 정보가 필요 없다고 판단하면 해당 정보를 제거한다.

RM에서는 TM의 요청이 있기 전 까지는 XA 트랜잭션 브랜치의 정보를 제거하지 않는다.

7.4. 데이터베이스 링크

데이터베이스 링크는 원격 데이터베이스의 데이터를 마치 로컬 데이터베이스의 데이터처럼 접근할 수 있는 방법을 제공한다. 데이터베이스 링크를 사용하면 원격 데이터베이스의 데이터에 대한 접근, 수정이 용이하며 손쉽게 분산 트랜잭션을 처리할 수 있다. 분산 트랜잭션은 트랜잭션의 원자성을 보장하기 위해 XA와 마찬가지로 Two-phase commit mechanism을 사용한다.

7.4.1. 데이터베이스 링크의 생성, 제거

데이터베이스 링크는 다음과 같은 접근 권한에 따라 생성 및 제거 방법이 다르다.

Public DBLink

데이터베이스 링크를 생성한 사용자와 다른 사용자들도 데이터베이스 링크를 이용할 수 있다.

Public DBLink를 생성하기 위해서는 **create public database link** 권한이 있어야 한다.

다음은 Public DBLink를 생성하는 예이다.

```
create public database link public_tibero using 'remote_2';
```

위의 예에서 using 절 이후의 remote_2는 연결할 데이터베이스를 가리키는 이름으로 tbdsn.tbr 파일에 해당 데이터베이스의 연결 정보가 저장되어 있어야 한다.

다음은 Public DBLink를 제거하는 예이다. Public DBLink는 **drop public database link** 권한을 가진 사용자만 제거할 수 있다.

```
drop public database link public_tibero;
```

Private DBLink

데이터베이스 링크를 생성한 사용자만 데이터베이스 링크를 사용할 수 있다. Private DBLink를 생성하기 위해서는 **create database link** 권한이 있어야 한다.

다음은 Private DBLink를 생성하는 예이다.

```
create database link remote_tibero using 'remote_1';
```

위의 예에서는 remote_1은 데이터베이스에 연결하는 remote_tibero라는 이름의 데이터베이스 링크를 생성한다. 이 데이터베이스 링크는 Private DBLink이므로, 생성한 사용자 외에는 사용할 수 없다.

다음은 Private DBLink를 제거하는 예이다. Private DBLink는 생성한 사용자만 제거할 수 있다.

```
drop database link remote_tibero;
```

7.4.2. 원격 데이터베이스의 연결

원격 데이터베이스와의 연결에 사용하는 계정을 설정하는 방법은 다음과 같이 두 가지가 있다.

설정 방법	설명
지정한 계정	지정한 ID와 패스워드를 사용해 원격 데이터베이스에 접속한다. 단, 지정한 ID와 패스워드를 가진 계정이 원격 데이터베이스에 존재해야 한다. 어떤 사용자가 사용하더라도 데이터베이스 링크를 생성할 때에는 지정한 ID와 패스워드를 사용해야 한다.
현재 연결된 계정	현재 질의를 수행한 사용자의 ID와 패스워드를 사용해 원격 데이터베이스에 접속한다. 데이터베이스 링크를 사용하는 사용자의 ID와 패스워드가 원격 데이터베이스에 동일하게 존재해야 한다. 계정을 지정하지 않으면 기본으로 현재 연결된 계정으로 접속하도록 설정된다. 따라서, 데이터베이스 링크를 사용하는 사용자별로 다른 ID와 패스워드를 사용한다.

원격 데이터베이스에 연결하기 위한 계정은 CREATE SESSION 등의 권한을 가져야 하며, 데이터베이스 링크를 통해 원격 데이터베이스의 연결에 사용된 계정의 권한을 로컬 사용자가 획득하게 되므로 권한 관리에 유의해야 한다. 특히 Public DBLink의 경우에는 모든 로컬 사용자가 원격 데이터베이스에 대한 권한을 갖기 때문에 주의하여 사용해야 한다.

다음은 지정한 계정을 이용하는 데이터베이스 링크의 생성 예이다.

```
create database link remote_tibero connect to user1
identified by 'password' using 'remote_1';
```

다음은 현재 연결된 계정을 이용하는 데이터베이스 링크의 생성 예이다.

```
create database link remote_tibero using 'remote_1';
```

7.4.3. 게이트웨이

데이터베이스 링크를 통해 질의를 수행할 때, 데이터베이스 링크의 대상이 Tibero RDBMS가 아닌 타 DBMS라면 각각의 DBMS를 위한 게이트웨이를 통해 데이터베이스 링크를 수행할 수 있다.

Tibero RDBMS 서버는 타 DBMS에 필요한 질의를 해당 게이트웨이에 전달하고, 게이트웨이는 원격 DBMS에 접속하여 Tibero RDBMS 서버로부터 전달 받은 질의를 수행하고 그 결과를 다시 Tibero RDBMS 서버로 전송한다. 타 DBMS로의 데이터베이스 링크 기능을 사용하고자 하는 경우에는 해당 DBMS에 대한 게이트웨이 바이너리와 설정 파일이 필요하다.

본 절에서는 DBMS 벤더별로 게이트웨이의 종류를 설명하고, 게이트웨이와 Tibero RDBMS 서버가 같은 머신에 존재하는 경우와 다른 머신에 존재하는 경우를 알아본다. 또한, 게이트웨이에서 제공하는 옵션 및 로깅도 설명한다.

DBMS 벤더별 게이트웨이

다음은 Tibero RDBMS에서 데이터베이스 링크 기능을 지원하고 있는 타 DBMS의 종류와 게이트웨이 바이너리명이다.

DBMS 벤더명	게이트웨이 바이너리명	프로그래밍 언어	DBMS 버전
Oracle	gw4orcl	C	Oracle 9i, 10g, 11g
DB2	gw4db2	C	DB2 V8, DB2 9, DB2 9.5
MS-SQL SERVER	tbgateway.jar	Java	MS-SQL SERVER 2000, 2005, 2008
Adaptive Server Enterprise(Sybase)	tbgateway.jar	Java	Sybase SQL Server 10.0.2 or later

각각의 게이트웨이 바이너리는 DBMS의 버전에 따라 다를 수 있기 때문에 버전에 맞는 게이트웨이 바이너리를 사용할 것을 권장한다.

게이트웨이 프로세스 생성 방식

게이트웨이는 연결할 DBMS가 제공하는 라이브러리가 필요하다. 라이브러리를 Tibero RDBMS 서버가 설치된 곳에서 사용할 수 있다면, Tibero RDBMS 서버와 같은 머신 내에서 게이트웨이 프로세스를 생성하여 데이터베이스 링크 기능을 수행할 수 있다. 생성된 게이트웨이 프로세스는 해당 데이터베이스 링크를 사용하는 세션이 닫힐 때 종료된다.

다음은 Oracle 서버와 연결하는 데이터베이스 링크를 사용하기 위해 tbdsn.tbr 파일을 설정하는 예이다.

<tbdsn.tbr>

```
ora_dblink=(
            (GATEWAY=(PROGRAM=gw4orcl)
                     (TARGET=orcl)
                     (TX_MODE=GLOBAL))
)
```

다음은 DB2 서버와 연결하는 데이터베이스 링크를 사용하기 위해 tbdsn.tbr 파일을 설정하는 예이다.

<tbdsn.tbr>

```
db2_dblink=(
            (GATEWAY=(PROGRAM=gw4db2)
                     (TARGET=sample)
```

```
                    (TX_MODE=GLOBAL))
)
```

항목	설명
PROGRAM	게이트웨이 바이너리의 절대 주소이다. 게이트웨이 바이너리가 $TB_HOME/client/bin 디렉터리에 있는 경우, 바이너리명만 명시할 수 있다.
TARGET	DBMS별로 다음과 같이 의미하는 것이 다르다. - Oracle 서버인 경우: 네트워크 서비스명이다. - DB2 서버인 경우: 데이터베이스명이다.
TX_MODE	글로벌 트랜잭션(Global Transaction) 혹은 로컬 트랜잭션(Local Transaction)으로 처리할지의 여부를 설정한다. 글로벌 트랜잭션은 커밋을 요청할 때 Two-phase commit으로 동작하고, 로컬 트랜잭션은 Two-phase commit으로 동작하지 않는다. TX_MODE의 값은 처리 여부에 따라 다음과 같이 설정할 수 있다. - 글로벌 트랜잭션인 경우: GLOBAL - 로컬 트랜잭션인 경우: LOCAL

멀티 스레드 서버 방식

사용자는 Tibero RDBMS 서버와 같은 머신 또는 원격에 있는 머신에서 게이트웨이를 멀티 스레드 서버 방식으로 시작할 수 있다. Tibero RDBMS 서버의 세션은 tbdsn.tbr 파일에 명시된 접속 정보를 통해 게이트웨이와 TCP/IP 통신을 한다. 멀티 스레드 서버 방식의 게이트웨이는 Tibero RDBMS 서버의 세션으로부터 요청이 오면 미리 생성된 워킹 스레드 중 하나가 해당 요청을 처리한다. 특히 Java 프로그래밍 언어를 사용하는 게이트웨이는 멀티 스레드 서버 방식만을 지원한다.

다음은 ORACLE 서버와 연결하는 데이터베이스 링크를 사용하기 위해 tbdsn.tbr 파일을 설정하는 예이다.

<tbdsn.tbr>

```
ora_link_remote=(
                 (GATEWAY=(LISTENER=(HOST=12.34.56.78)
                                    (PORT=9999))
                          (TARGET=orcl)
                          (TX_MODE=GLOBAL))
)
```

다음은 DB2 서버와 연결하는 데이터베이스 링크를 사용하기 위해 tbdsn.tbr 파일을 설정하는 예이다.

<tbdsn.tbr>

```
db2_link_remote=(
                 (GATEWAY=(LISTENER=(HOST=12.34.56.78)
                                    (PORT=9999))
                          (TARGET=sample)
                          (TX_MODE=GLOBAL))
)
```

다음은 MS-SQL SERVER와 연결하는 데이터베이스 링크를 사용하기 위해 tbdsn.tbr 파일을 설정하는 예이다.

<tbdsn.tbr>

```
mssql_link_remote=(
                   (GATEWAY=(LISTENER=(HOST=12.34.56.78)
                                      (PORT=9093))
                            (TARGET=192.168.16.25:1433:master)
                            (TX_MODE=LOCAL))
)
```

다음은 Sybase ASE 서버와 연결하는 데이터베이스 링크를 사용하기 위해 tbdsn.tbr 파일을 설정하는 예이다.

<tbdsn.tbr>

```
ase_link_remote=(
                 (GATEWAY=(LISTENER=(HOST=12.34.56.78)
                                    (PORT=9093))
                          (TARGET=192.168.16.25:5000:master)
                          (TX_MODE=LOCAL))
)
```

항목	설명
LISTENER	- HOST: 원격에 있는 머신에서 게이트웨이가 존재하는 호스트의 IP 주소를 설정한다. - PORT: 원격에 있는 머신에서 게이트웨이가 존재하는 호스트의 포트 번호를 설정한다.
TARGET	DBMS별로 다음과 같이 의미하는 것이 다르다. - Oracle 서버인 경우: 네트워크 서비스명이다. - DB2 서버인 경우: 데이터베이스명이다. - MS-SQL SERVER인 경우: 서버의 연결 정보(IP:PORT:DATABASE NAME)이다.

항목	설명
	- Sybase ASE 서버인 경우: 서버의 연결 정보(IP:PORT:DATABASE NAME)이다.
TX_MODE	글로벌 트랜잭션(Global Transaction) 혹은 로컬 트랜잭션(Local Transaction)으로 처리할지의 여부를 설정한다. TX_MODE의 값은 처리 여부에 따라 다음과 같이 설정할 수 있다. - 글로벌 트랜잭션인 경우: GLOBAL - 로컬 트랜잭션인 경우: LOCAL

원격에 있는 게이트웨이를 사용하기 위해서는 먼저 원격에 있는 게이트웨이를 실행시켜야 한다.

다음은 원격에서 Oracle 서버의 게이트웨이를 실행시키는 예이다.

```
$ gw4orcl
```

ORACLE, DB2

게이트웨이는 기본적으로 TBGW_HOME 환경변수를 통해 설정 파일을 읽고 로그 파일을 기록한다.

TBGW_HOME 환경변수가 설정되어 있지 않은 경우에는 기본값은 ${TB_HOME}/client/gateway 이다. Windows 환경에서는 기본값이 %TB_HOME%\client\gateway로 설정된다.

게이트웨이가 사용하는 설정 파일과 로그 파일이 존재하는 디렉터리 구조는 다음과 같다.

```
$TBGW_HOME
   |--- DBMS 벤더명
          |--- config
          |     |--- tbgw.cfg
          |--- log
                | --- 게이트웨이의 로그 파일
```

위의 디렉터리 구조에서 $TBGW_HOME 이라고 보이는 부분은 시스템 환경에 맞게 바꿔서 읽어야 한다.

DBMS 벤더명/config

tbgw.cfg라는 게이트웨이 설정 파일이 있다. 사용자가 게이트웨이와 관련된 설정 값을 변경하고 싶을 때 생성하며, 위의 디렉터리 구조에 맞게 위치시키면 된다.

DBMS 벤더명/log

게이트웨이와 관련된 로그 파일이 있다. 로그 파일은 DBMS 벤더명에 맞춰 생성된다.

다음은 DBMS 벤더별 로그 파일이다.

DBMS 벤더명	로그 파일명	리스너의 로그명
Oracle	gw4orcl.log	gw4orcl_lsnr.log

DBMS 벤더명	로그 파일명	리스너의 로그명
DB2	gw4db2.log	gw4db2_lsnr.log

tbgw.cfg 파일에 초기화 파라미터의 설정 값을 명시함으로써 게이트웨이와 관련된 설정을 변경할 수 있다.

다음은 게이트웨이를 설정하는 예이다.

<tbgw.cfg>

```
LOG_DIR=${TBGW_HOME}/{DBMS 벤더명}/log
LOG_LVL=2
LISTENER_PORT=9999
MAX_LOG_SIZE=20k
MAX_LOG_CNT=5
FETCH_SIZE=32k
```

초기화 파라미터	기본값	설명
LOG_DIR	${TBGW_HOME}/{DBMS 벤더명}/log	게이트웨이의 로그 파일을 저장할 경로를 설정한다.
LOG_LVL	2	로그 파일에 남길 로그 레벨을 설정한다.
LISTENER_PORT	9999	게이트웨이를 리스너 모드로 실행하는 경우 리스너의 포트 번호를 설정한다.
MAX_LOG_SIZE	0	로그 파일의 최대 크기(Byte 단위)이다. - 값이 0인 경우: 로그 파일의 최대 크기를 설정하는 데 제한이 없다. - 값을 명시한 경우: 로그 파일이 설정된 최대 크기를 초과하면 로그 파일을 백업한다.
MAX_LOG_FILE_CNT	0	로그 파일을 백업하는 기능을 사용하는 경우, 백업 로그 파일(Backup log file)의 최대 개수이다. - 값이 1 이하인 경우: 백업 로그 파일을 생성하지 않는다. - 값을 명시한 경우: 로그 파일을 백업한 후, 설정된 최대 개수를 초과하면 가장 오래된 백업 로그 파일을 삭제한다.
FETCH_SIZE	32KB	DBMS에 질의 처리를 할 때 한꺼번에 가져오는 데이터의 크기를 설정한다. (최댓값: 64KB)
SKIP_CHAR_CONV	N	gw4orcl에만 해당하는 옵션이다. - 값이 Y인 경우 Oracle 데이터베이스에 있는 데이터를 캐릭터셋 변환 없이 가져온다.

초기화 파라미터	기본값	설명
		- Oracle에서 한글을 지원하지 않는 캐릭터셋과 한글 데이터를 동시에 사용하고 있는 특수한 경우에 사용된다.

MS-SQL SERVER, Sybase ASE

사용자는 ${TB_HOME}/client/bin에 있는 tbgateway.zip 파일을 설치할 디렉터리에 복사한 후, 압축을 해제한다. 기본적으로 타깃 데이터베이스에 대한 JDBC 드라이버 파일(jconn3.jar, sqljdbc.jar)은 제공하지 않는다. 따라서 사용자는 해당 DBMS의 웹사이트를 접속하여 JDBC 드라이버 파일을 다운로드 한 후, LIB 디렉터리에 복사한다.

게이트웨이가 사용하는 설정 파일과 로그 파일이 존재하는 디렉터리 구조는 다음과 같다.

```
설치 디렉터리
|--- tbJavaGW
       |--- tbgw
       |--- tbgw.cfg
       |--- tbgwlog.properties
       |--- lib
       |     |--- tbgateway.jar
       |     |--- commons-collections.jar
       |     |--- commons-pool.jar
       |     |--- log4j-1.2.15.jar
       |     |--- jconn3.jar
       |     |--- sqljdbc.jar
       |--- log
             | --- 게이트웨이의 로그 파일
```

tbJavaGW/tbgw

Java 게이트웨이를 실행시키는 스크립트 파일이다.

tbJavaGW/tbgw.cfg

게이트웨이 설정 파일이다. 사용자가 게이트웨이와 관련된 설정 값을 변경하고 싶을 때 생성하며, 위의 디렉터리 구조에 맞게 위치시키면 된다.

tbJavaGW/tbgwlog.properties

로그에 대한 설정 파일이다. 로그 파일의 크기와 로그 레벨 등을 설정할 수 있다. 자세한 형식은 LOG4J를 참고하기 바란다.

tbJavaGW/lib

Java 게이트웨이에서 사용하는 JAR 파일이 있는 디렉터리이다. 타깃 데이터베이스의 JDBC 드라이버도 해당 디렉터리에 있다.

tbJavaGW/log

게이트웨이와 관련된 로그 파일이 생성된다.

tbgw.cfg 파일에 초기화 파라미터의 설정 값을 명시함으로써 게이트웨이와 관련된 설정을 변경할 수 있다.

다음은 게이트웨이를 설정하는 예이다.

<tbgw.cfg>

```
DATABASE=SQL_SERVER
LISTENER_PORT=9093
INIT_POOL_SIZE=10
MAX_POOL_SIZE=1000
ENCODING=MSWIN949
MAX_LONGVARCHAR=4K
MAX_LONGRAW=4K
```

초기화 파라미터	기본값	설명
DATABASE	SQL_SERVER	타깃 데이터베이스를 설정한다. - MS-SQL SERVER: SQL_SERVER - Sybase ASE: ASE
LISTENER_PORT	9093	리스너의 포트 번호를 설정한다.
INIT_POOL_SIZE	10	게이트웨이가 시작할 때 미리 생성할 워킹 스레드의 개수를 설정한다.
MAX_POOL_SIZE	100	게이트웨이가 최대로 생성할 수 있는 워킹 스레드의 개수를 설정한다.
MAX_CURSOR_CACHE_SIZE	100	워킹 스레드 당 최대로 캐시 가능한 커서의 개수를 설정한다.
ENCODING	MSWIN949	Tibero RDBMS 서버의 세션에 문자열을 전달할 때 사용할 인코딩을 설정한다. 단, Tibero RDBMS 서버의 문자 집합과 일치시켜야 한다. 설정할 수 있는 문자 집합은 다음과 같다. - ASCII - EUC-KR - MSWIN949 - UTF-8 - UTF-16 - SHIFT-JIS

초기화 파라미터	기본값	설명
MAX_LONGVARCHAR	4KB	게이트웨이는 LONG, CLOB 타입의 데이터를 일정 간격을 정하여 가져오는 방식(Deferred 형태)을 지원하지 않는다. CHAR나 VARCHAR 타입처럼 한 번에 읽어 오게 되는데, 그때 읽어올 수 있는 최대 크기를 설정한다. (최댓값 : 32KB)
MAX_LONGRAW	4KB	게이트웨이는 LONG RAW, BLOB 타입의 데이터를 일정 간격을 정하여 가져오는 방식을 지원하지 않는다. RAW처럼 한 번에 읽어 오게 되는데, 그때 읽어올 수 있는 최대 크기를 설정한다. (최댓값: 32KB)

게이트웨이 바이너리의 버전

게이트웨이 바이너리의 버전은 다음과 같은 명령을 실행하여 확인할 수 있다.

```
$ gw4orcl -v

tbGateway for oracle : Release 4 Trunk (Build 31190)

Linux Tibero_Linux 2.6.22-16-generic #1 SMP Mon Nov 24 17:50:35
   GMT 2008 x86_64 GNU/Linux version (little-endian)
```

7.4.4. 데이터베이스 링크의 사용

데이터베이스 링크를 통해 접근할 수 있는 데이터베이스 객체의 종류는 다음과 같다.

- 테이블(LOB와 LONG 타입의 컬럼에는 접근할 수 없다.)
- 뷰
- 시퀀스

데이터베이스 링크를 사용할 수 있는 SQL 문장은 SELECT, INSERT, UPDATE, DELETE 이다. 단, SELECT 문에서 FOR UPDATE 절은 사용할 수 없다.

7.4.5. Global Consistency

Homogeneous DBLink로 구성된 분산 트랜잭션은 **Global Consistency**를 제공한다. 이를 위해 분산 트랜잭션에 참여한 데이터베이스 간에 TSN을 동기화하는 작업을 한다.

TSN을 동기화하는 작업은 데이터베이스 사이에 메시지 교환을 할 때 이루어지며, 커밋을 할 때에는 모든 노드의 Commit TSN을 동일하게 동기화 해야 한다.

7.4.6. 데이터베이스 링크의 In-doubt 트랜잭션 처리

정체되고 있는 TX에 대한 처리는 XA의 경우와 동일하다. 본 절에서는 Two-phase commit mechanism에서 XA와의 차이점을 설명한다.

commit point site

데이터베이스 링크를 사용한 분산 트랜잭션에서는 트랜잭션에 참여한 노드 중에서 commit point site를 선정한다.

commit point site는 Two-phase commit을 시작할 때 설정하며, 세션 트리를 따라가며 가장 큰 commit point strength를 가진 노드를 선택한다. 각 데이터베이스는 commit point strength를 가지는데, 이 값은 초기화 파라미터 `COMMIT_POINT_STRENGTH`로 설정할 수 있다.

commit point site는 Two-phase commit의 prepare phase에서 prepare를 하지 않는다. 대신 모든 노드가 prepare를 한 이후에 commit point site를 바로 커밋한다. 그 이후에 다른 노드들은 commit phase를 실행한다.

데이터베이스 링크에서 이와 같이 수정된 Two-phase commit을 사용하는 이유는 Global consistency를 보장하기 위해 생기는 오버헤드를 줄이기 위해서이다. 데이터베이스 링크에서 Global consistency를 보장하기 위해서는 모든 노드의 Commit TSN을 동일하게 해야 한다. Commit TSN은 prepare phase까지 완료한 노드 중 가장 큰 TSN으로 결정하고 commit phase에서 결정된 TSN을 사용해 커밋을 한다.

commit phase 전에는 Commit TSN을 모르기 때문에 다른 트랜잭션에서 해당 트랜잭션의 수정 정보의 조회 여부를 결정할 수 없다. 다른 트랜잭션에서 prepare된 TX가 수정한 내용에 접근하는 경우 다음과 같은 에러가 발생한다.

```
TBR-21019: lock held by in-doubt distributed transaction.
```

데이터베이스 링크를 사용할 때 prepare 상태에서 정체가 발생하는 경우, 해당 데이터에 접근할 수 없기 때문에, In-doubt 트랜잭션으로 인한 문제가 발생된다.

이와 같은 문제를 경감하기 위해 트랜잭션에 참여한 노드 중 한 노드는 prepare phase를 거치지 않고, 트랜잭션이 in-doubt 상태가 되더라도 commit point site는 위와 같이 데이터를 접근하지 못하는 상황을 방지할 수 있도록 하였다. 따라서 commit point strength는 데이터 접근성이 많이 필요한 데이터베이스일수록 큰 값을 설정해야 한다.

7.4.7. 데이터베이스 링크의 정보 조회

Tibero RDBMS에서는 생성한 데이터베이스 링크의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다.

정적 뷰	설명
DBA_DB_LINKS	Tibero RDBMS 내의 모든 데이터베이스 링크의 정보를 보여주는 뷰이다. DBA만 사용할 수 있는 뷰이다.
ALL_DB_LINKS	현재 사용자가 이용할 수 있는 모든 데이터베이스 링크의 정보를 보여주는 뷰이다.
USER_DB_LINKS	현재 사용자가 생성한 데이터베이스 링크의 정보를 보여주는 뷰이다.

참고

정적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

V$DBLINK

동적 뷰 V$DBLINK는 해당 세션에서 원격 데이터베이스에 연결된 데이터베이스 링크의 정보를 보여주는 뷰이다.

사용자가 데이터베이스 링크를 통해 질의를 수행하면 원격 데이터베이스에 연결을 생성하고 해당 질의 또는 트랜잭션이 종료되어도 연결을 해제하지 않고 계속 유지된다. 즉, 같은 데이터베이스 링크를 사용했을 때 발생하는 연결의 비용을 줄이기 위함이다. 이렇게 생성된 연결은 세션이 종료될 때까지 혹은 명시적으로 연결을 종료할 때까지 계속 유지된다.

다음은 remote_tibero를 통해 SELECT 문을 수행한 후, V$DBLINK를 통해 연결 정보를 조회하는 예이다.

```
SQL> select * from employee@remote_tibero;

 ID     NAME
--- ---------
  1      KIM
  2      LEE
  3     HONG

3 rows selected.

SQL> select * from V$DBLINK;

DB_LINK         OWNER_ID   OPEN_CURSORS   IN_TRANSACTION  HETEROGENEOUS
-------------  ----------  ------------   --------------  -------------
REMOTE_TIBERO         15             0    YES             NO

COMMIT_POINT_STRENGTH
---------------------
                   1

1 row selected.
```

현재 remote_tibero의 소유자의 ID는 15번이고, 현재 열려있는 커서는 없으며, 트랜잭션을 수행하고 있다. 동일한 Tibero RDBMS 서버에 대한 데이터베이스 링크이며 원격 데이터베이스의 commit point strength는 1이다.

다음은 데이터베이스 링크 기능을 종료하는 예이다.

```
SQL> alter session close database link remote_tibero;
TBR-12056: database link is in use.                          ... ① ...

SQL> commit;                                                 ... ② ...
Commit succeeded.

SQL> alter session close database link remote_tibero;        ... ③ ...
Session altered.

SQL> select * from V$DBLINK;                                 ... ④ ...

DB_LINK          OWNER_ID   OPEN_CURSORS   IN_TRANSACTION  HETEROGENEOUS
--------------  ----------  ------------   --------------  -------------

COMMIT_POINT_STRENGTH
---------------------

0 row selected.
```

① 데이터베이스 링크 기능을 종료하려는 시도가 실패한 이유는 해당 데이터베이스 링크를 사용한 트랜잭션이 아직 수행 중이기 때문이다.

② commit 문을 통해 해당 트랜잭션을 종료한다.

③ 다시 데이터베이스 링크 기능의 종료를 시도한다.

④ 정상적으로 데이터베이스 링크가 종료되며, 이를 확인하는 방법은 V$DBLINK를 통해 확인할 수 있다.

제8장 Tibero Standby Cluster

본 장에서는 Tibero Standby Cluster의 구성요소와 동작 및 운영 방법을 설명한다.

8.1. 개요

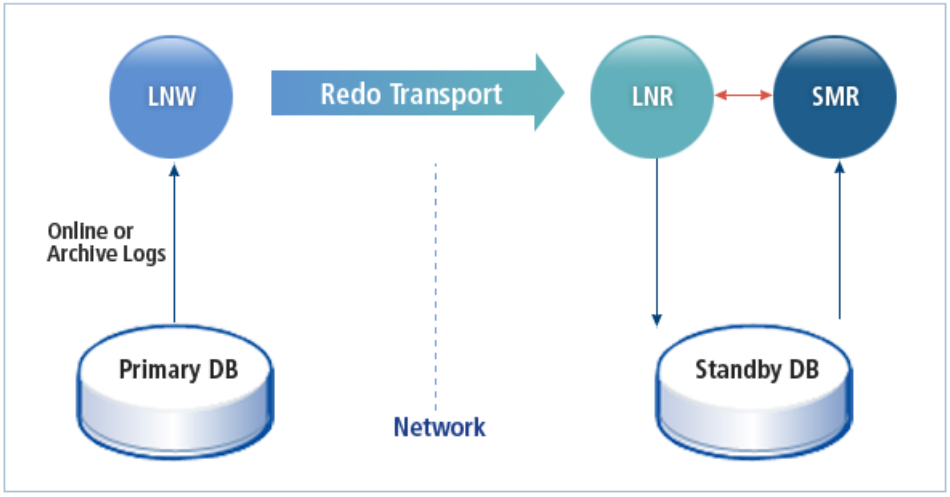
Tibero Standby Cluster는 데이터베이스의 고가용성, 데이터의 보호, 재난 복구 등을 목적으로 제공하는 Tibero RDBMS의 핵심 기능이다.

Tibero Standby 서버는 물리적으로 독립된 장소에 원본 데이터베이스의 복사본을 트랜잭션 단위로 보관한다. 복사할 대상이 되는 원본 데이터베이스를 **Primary DB**(이하 Primary)라고 부르고, 복사된 데이터가 저장되는 데이터베이스를 **Standby DB**(이하 Standby)라고 부른다.

Tibero Standby Cluster의 원리는 Primary에서 생성된 Redo 로그를 배경 프로세스가 Standby로 전송하고, Standby는 Redo 로그를 이용해 Primary의 모든 변화를 똑같이 반영하는 것이다.

다음은 Tibero Standby Cluster가 어떻게 동작하는지를 나타내는 그림이다.

[그림 8.1] Tibero Standby Cluster의 동작 구조



데이터의 복사를 통해 Primary는 서비스가 요청한 데이터 처리에 실패했을 경우, Standby의 데이터를 활용해 해당 서비스를 신속히 재개할 수 있다. 또한 Primary의 서버만으로는 손상된 데이터를 복구를 할 수 없는 경우에도 쉽게 대처할 수 있다. 예를 들면, Primary의 서버의 디스크가 손상된 경우 Standby를 통해 손상된 데이터를 보호할 수 있다.

8.1.1. 프로세스

Tibero Standby Cluster의 프로세스는 다음과 같다.

- LNW(Log Network Writer)

  Primary의 Redo 로그를 Standby로 전송하는 프로세스이다. 로그 전송 방식과 무관하게 로그를 보내는 것은 항상 LNW에서 이루어진다.

  Standby는 9개까지 설정이 가능하며, 각 Standby 마다 LNW가 하나씩 실행된다.

- LNR(Log Network Reader)

  Standby에서 Primary로부터 받은 Redo 로그를 온라인 Redo 로그 파일에 기록하는 프로세스이다.

  Standby는 MOUNT나 NORMAL이 아닌 RECOVERY 부트 모드로 동작하며, 이때 log writer는 사용되지 않고, LNR이 LGWR의 역할을 대신한다.

- SMR(Standby Managed Recovery)

  온라인 Redo 로그를 읽어 Standby에 적용하는 복구 과정을 수행하는 프로세스이다.

  LNR, SMR은 첫 번째 워킹 프로세스의 워킹 스레드 중 하나로 동작한다. 따라서, Standby를 구성하기 위해서는 $TB_SID.tip 파일의 _WTHR_PER_PROC 초기화 파라미터의 값을 2보다 크게 설정해야 한다.

8.1.2. 로그 전송 방식

Primary에서 Standby로 로그가 전송되는 방식은 다음과 같다.

로그 전송 방식	설명
LGWR SYNC	LGWR가 Redo 로그를 디스크에 기록할 때 LNW를 통해 Standby에 Redo 로그를 전송한다.
LGWR ASYNC	LNW가 직접 온라인 Redo 로그 파일을 읽어서 Standby로 보낸다.
ARCH ASYNC	ARCH가 로그 순환을 할 때 아카이브 로그를 만든 후 LNW에게 알려주고, LNW는 아카이브 로그 파일을 읽어 Standby로 보낸다.

8.1.3. Primary의 동작 모드

Primary DB의 동작 모드에는 다음과 같다.

구성요소	설명
PROTECTION 모드	적어도 하나 이상의 LGWR SYNC를 포함해야 한다.

구성요소	설명
	LGWR는 디스크에 기록하는 것 외에도 LGWR SYNC인 Standby 중 적어도 하나의 Standby로부터 성공했다는 응답을 받아야 진행할 수 있다. 모든 LGWR SYNC인 Standby가 실패한 경우, Primary도 같이 실패하고, 더 이상 진행하지 않는다.
AVAILABILITY 모드	적어도 하나 이상의 LGWR SYNC를 포함해야 한다. 모든 LGWR SYNC인 Standby가 실패하면 Standby와의 동기화를 포기한다. 하지만 Primary는 계속 진행한다.
PERFORMANCE 모드	로그 전송 방식에 제한이 없고, Standby의 실패와 무관하게 Primary는 계속 진행한다.

8.2. Primary의 설정 및 운용

다음은 Standby로 연결할 데이터베이스의 정보와 각 Standby의 종류, 그리고 동작 모드를 설정하는 방법이다.

<$TB_SID.tip>

```
LOG_REPLICATION_MODE   = {PROTECTION|AVAILABILITY|PERFORMANCE}
LOG_REPLICATION_DEST_1 = "hostname_1:port_1 {LGWR SYNC|LGWR ASYNC|ARCH ASYNC}"
LOG_REPLICATION_DEST_2 = "hostname_2:port_2 {LGWR SYNC|LGWR ASYNC|ARCH ASYNC}"
...
LOG_REPLICATION_DEST_N = "hostname_N:port_N {LGWR SYNC|LGWR ASYNC|ARCH ASYNC}"
```

다음은 위 파일에서 설정한 각 초기화 파라미터에 대한 설명이다.

- LOG_REPLICATION_MODE
  - 데이터를 보호하는 수준에 중점을 둘지 혹은 성능을 최대화할지에 대한 전체적인 동작 모드를 설정한다. 한번만 설정하면 된다.
  - LOG_REPLICATION_MODE 초기화 파라미터에 설정할 수 있는 항목은 다음과 같다.

항목	설명
PROTECTION	LGWR SYNC로 설정된 Standby가 하나도 없는 경우를 허용할 수 없는 항목이다. 이 항목을 설정하면 서버를 기동할 때 초기화 에러가 발생한다. 이 에러를 해결하기 위해서는 모드에 따라 Standby에 맞게 설정한 후, 다시 서버를 기동해야 한다.
AVAILABILITY	PROTECTION 항목과 마찬가지로 LGWR SYNC로 설정된 Standby가 하나도 없는 경우를 허용할 수 없는 항목이다.

항목	설명
	이 항목을 설정하면 서버를 기동할 때 초기화 에러가 발생한다. 이 에러를 해결하기 위해서는 모드에 따라 Standby에 맞게 설정한 후, 다시 서버를 기동해야 한다.
PERFORMANCE	Standby로 로그가 전송되는 방식에 제한이 없고, 동기화를 보장하지 않으므로 시스템 성능을 높이는 데 가장 유리한 항목이다.

- LOG_REPLICATION_DEST_N
  - 각 Standby의 데이터베이스의 연결 정보(hostname:port)와 로그 전송 방식을 설정한다. 설정할 수 있는 최대 Standby의 개수(N)은 9이고, 필요한 만큼만 LOG_REPLICATION_DEST_1부터 설정한다.
  - LOG_REPLICATION_DEST_N 초기화 파라미터에 설정할 수 있는 항목은 다음과 같다.

항목	설명
LGWR SYNC	LGWR SYNC로 설정된 Standby는 Primary의 Redo 버퍼의 내용을 전송받아 동작하므로, 가장 빈번하게 Redo 로그를 전송한다. 따라서 데이터가 보호될 확률도 높다. 반면에 Primary의 성능 저하가 심하므로 Standby를 Primary와 비슷한 수준으로 구축할 것을 권장한다. PERFORMANCE 모드와 같이 사용할 수 있는데, 이러한 경우 데이터를 보호하지 못하더라도 Primary는 계속 진행할 수 있다. 따라서 Standby가 느리면 온라인 Redo 로그 파일이나 아카이브 로그 파일에서 Redo 로그를 읽어 전송할 수 있으므로 Primary를 ARCHIVELOG 모드로 운영할 것을 권장한다.
LGWR ASYNC	LGWR ASYNC로 설정된 Standby는 LGWR SYNC와 ARCH ASYNC의 중간 수준의 빈도로 Redo 로그를 전송한다. 기본적으로 온라인 Redo 로그 파일을 읽어서 전송하지만, Standby가 따라오지 못하는 경우 아카이브 로그 파일에서 읽을 수도 있으므로 Primary를 ARCHIVELOG 모드로 운영할 것을 권장한다.
ARCH ASYNC	ARCH ASYNC로 설정된 Standby가 하나 이상 존재하면 Primary는 반드시 ARCHIVELOG 모드로 동작해야 한다. 그렇지 않으면 서버의 기동은 정상적으로 되지만, 해당 Standby는 아무런 동작도 하지 못하게 된다. 이 점을 주의해야 한다. 비록 ARCH ASYNC인 Standby를 사용하지 않더라도 Standby 기능을 사용할 때에는 Primary를 ARCHIVELOG 모드로 운영할 것을 권장한다.

Primary를 NORMAL 모드로 기동하면 $TB_SID.tip 파일에 설정된 각 Standby와 연결이 이루어지고, 배경 프로세스에 의해 자동으로 **데이터 이중화**(Replication)가 이루어진다. 예를 들어, PROTECTION이나 AVAILABILITY 동작 모드로 기동하고 LGWR SYNC인 Standby가 모두 연결이 불가능한 상태라면 Primary도 운영이 불가능하므로 반드시 Standby를 먼저 기동해야 한다. 그 외의 경우에는 Standby를 나중에 기동하더라도 자동으로 연결되므로 운영이 가능하다.

참고

Primary에서 데이터베이스를 생성하는 동안에는 $TB_SID.tip 파일에 있는 Standby 설정이 무시된다. 따라서 Primary에서 생성된 DB 파일을 DBA가 수동으로 Standby로 복사해야 한다.

8.3. Standby의 설정 및 운용

Standby를 운영하기 위해 필요한 설정 과정은 다음과 같다.

1. Primary에서 백업한 DB 파일을 복사해서 Standby를 구성한다.

   컨트롤 파일, 온라인 로그 파일, 패스워드 파일을 포함한 모든 데이터 파일을 복사한다. 패스워드 파일을 함께 복사하는 이유는 Primary가 Standby에 SYS 권한을 가지고, 접속하므로 SYS의 패스워드가 서로 일치해야 하기 때문이다.

2. Standby의 $TB_SID.tip 파일을 열어 DB_NAME이 Primary와 같도록 수정한다.

3. $TB_SID.tip 파일에서 컨트롤 파일이 위치하는 디렉터리 경로가 1번에서 Primary 파일을 복사한 경로와 일치하는지 확인한다. 또한 DB_BLOCK_SIZE도 Primary에 설정된 것과 같아야 한다. 설정이 같으면 복사한 데이터 파일을 열 수 있다.

   Primary의 백업을 가져다 놓은 Standby의 디렉터리의 경로가 원래와 달라진 경우에는 Standby의 $TB_SID.tip 파일에 다음과 같이 경로 변환을 위한 정보를 추가해야 한다.

   [예 8.1] Standby의 $TB_SID.tip 파일의 경로 변환

   ```
   STANDBY_FILE_NAME_CONVERT="Primary의 절대경로, Standby의 절대경로"
   ```

   Primary의 절대 경로와 Standby의 절대 경로는 각각 Primary와 Standby의 instance 디렉터리의 절대 경로이다.

4. Standby를 MOUNT 모드로 기동한 후, 다음의 DDL 문장을 수행하면 변환된 경로가 컨트롤 파일에 적용된다.

   [예 8.2] 수정된 Standby의 경로 적용

   ```
   SQL> ALTER DATABASE Standby controlfile;
   ```

   위의 과정은 같은 경로로 DB 파일을 가져다 놓았다면 필요하지 않으며, 경로가 다른데도 위의 과정을 수행하지 않고 Standby를 기동하는 경우 컨트롤 파일에 적힌 경로에 데이터 파일을 열 수 없다는 에러가 발생한다.

5. Standby를 운영하기 위한 설정을 완료하면, 다음의 명령을 통해 DB가 Standby로 동작하도록 기동시킨다.

[예 8.3] Standby의 기동

```
$ tbboot -t RECOVERY
```

주의할 점은 NORMAL 모드로 Standby를 한 번이라도 기동하게 되면 더 이상 Standby로서의 기능은 할 수 없고, 앞서 설명한 과정을 다시 반복하여 Standby를 설정해야 한다.

Standby가 기동하기 전에 DB 파일이 이전의 버전이라 하더라도 일관성만 유지한다면 동작에는 문제가 없다. 내부적으로 Primary가 접속하여 Primary와 Standby사이의 로그 갭(log gap)을 자동으로 맞춰 동작한다. 하지만, 이 과정이 모두 Redo 로그에 의존하므로 두 DB 사이의 차이가 크고, Primary가 ARCHIVELOG 모드가 아니라면 필요한 Redo 로그가 존재하지 않아 동작이 불가능할 수 있다. 따라서 Primary는 ARCHIVELOG 모드로 동작시키거나 최근에 백업한 Primary를 Standby로 복사하여 두 DB의 DB 파일을 맞춘 후에 Primary를 동작시키는 것이 바람직하다.

8.3.1. Standby의 read only 모드

Standby는 내부적으로 Primary로부터 받은 Redo 로그를 디스크에 기록하고, 이를 복구하여 데이터 파일에도 반영하는 일을 수행하는 **RECOVERY** 모드로 동작하기 때문에 DB에 사용자의 접근이 제한된다. 하지만, DB의 데이터에 대해 읽기 작업만을 원하는 경우도 있다. 이를 위해 Standby는 **read only** 모드를 지원한다.

read only 모드로 실행하기 위해서는 관리자 권한으로 다음의 DDL 문장을 수행한다.

[예 8.4] Standby의 read only

```
SQL> ALTER DATABASE open read only;
```

Standby에 반영된 여러 객체의 조회가 허용된다. 하지만, read only 모드로 동작 중일 때에는 내부적으로 Primary로부터 받은 Redo 로그를 복구하는 과정이 중지되므로 Primary로부터 로그를 더 이상 받지 못하는 상태가 될 수 있다.

주의

특히 Primary를 PROTECTION이나 AVAILABILITY 모드로 운영하는 경우에는 DBA는 오랫동안 read only 상태로 Standby를 운영하지 않도록 주의해야 한다.

Standby를 다시 RECOVERY 모드로 전환시킬 때는 다음의 DDL 문장을 수행한다.

[예 8.5] RECOVERY 모드의 전환

```
SQL> ALTER DATABASE Standby;
```

이 문장을 실행할 때, 반드시 read only 모드에서 접근한 세션은 모두 해제된 상태이어야 한다.

Standby를 read only 클러스터 용으로 사용하는 경우와 같이 Standby에서 Redo 로그를 반영하는 과정을 중단하지 않고 읽기 작업을 원하는 경우가 있다. 그 때에는 다음의 DDL 문장을 실행하면 Standby는 복구 과정을 멈추지 않고, read only 세션을 허용한다.

[예 8.6] Standby의 read only continue recovery

```
SQL> alter database open read only continue recovery;
```

Standby가 LGWR SYNC 모드로 설정되어 Primary와 동기화되어 있는 경우라면, Primary에 접속한 경우와 같이 최근에 커밋된 내용을 Standby에서도 볼 수 있다. 이 상태에서는 비록 DB 서버가 read only 모드임에도 불구하고, 데이터의 내용이 변경될 수 있다는 사실에 주의해야 한다. 만약 read only 모드로 변경한 시점을 기준으로 항상 일관적인 데이터를 보고 싶은 경우에는 [예 8.4]의 문장을 수행해야 한다.

read only 모드에서는 Redo 로그를 이용해 복구하는 과정을 중단했든 아니든 간에 Primary와의 접속이 불가능하다. 따라서 Primary와 연결되었는지의 여부를 동적 뷰 V$STANDBY를 이용해 확인한 후 read only 모드로 변경하는 것이 바람직하다.

read only 모드로 동작시킨 상태에서 Primary와의 접속을 허용하기 위해서는 [예 8.5]에서 설명한 문장을 수행하여 RECOVERY 모드로 전환하면 된다.

8.4. 데이터베이스의 역할 전환

본 절에서는 Standby를 Primary로 전환하는 과정을 두 가지 시나리오로 나누어 설명한다.

8.4.1. Switchover

Primiary를 정상적으로 셧다운하고, Standby 중 하나를 Primary로 전환하고 싶은 경우 다음의 절차를 수행하면 된다.

1. Primary에서 다음의 명령어를 입력한다.

   [예 8.7] Switchover 명령어의 실행

   ```
   $ tbdown -t SWITCHOVER
   ```

   위의 명령어를 수행하면, 모든 Standby를 Primary와 동기화시킨다. 그 이후에 Primary가 종료된다.

2. Standby 중 하나를 종료한 후, NORMAL 모드로 기동하거나 다음의 DDL 문장을 수행하면 그 DB가 새로운 Primary가 된다.

   [예 8.8] ALTER DATABASE open 문장의 실행

   ```
   SQL> ALTER DATABASE open;
   ```

   read only 모드인 상태에서 open 모드로 바로 전환하는 것은 지원하지 않는다.

이전의 Primary를 새로운 Standby로 사용하고 싶다면, 새로 Primary가 될 DB의 $TB_SID.tip 파일에 Standby(LOG_REPLICATION_MODE, LOG_REPLICATION_DEST_n 초기화 파라미터)를 설정하고, 이전 Primary를 RECOVERY 모드로 기동한 후에 새로 Primary가 될 DB를 NORMAL 모드로 기동하면 서로 역할을 전환하여 수행할 수 있다.

이때, 두 DB는 이미 동기화된 상태이므로 Standby를 구성할 때 필요한 새로운 Primary의 DB 파일을 복사하고, **ALTER DATABASE Standby controlfile**을 수행하는 과정은 더 이상 필요하지 않는다. 또한, 새로운 Standby의 $TB_SID.tip 파일에 기존의 Standby와 관련된 설정이 남아 있더라도 DB가 NORMAL 모드로 운용될 때만 적용된다.

8.4.2. Failover

현재 Primary가 갑자기 비정상적으로 종료되었거나 접근이 불가능해진 경우, Standby 중 하나를 Primary로 사용할 수 있다. Standby를 Primary로 사용하기 위해서는 종료 후 다시 NORMAL 모드로 기동을 하거나 **ALTER DATABASE open** 문을 수행해야 한다.

참고

기존에 사용하던 Primary는 새로운 Primary가 운영된 후에는 더 이상 어떤 용도(Primary나 Standby)로도 Tibero Standby Cluster에 포함될 수 없다.

8.5. 클라이언트의 설정

Primary와 Standby에 모두 접속하기 위해서는 tbdsn.tbr 파일에 각 DB의 접속 정보를 추가해야 한다.

예를 들면 다음과 같다.

<tbdsn.tbr>

```
PrimaryDB_SID=(
               (INSTANCE=(HOST=primaryDB_hostname)
                         (PORT=primaryDB_port)
                         (DB_NAME=cluster_DB_NAME)
               )
)

StandbyDB_SID=(
               (INSTANCE=(HOST=StandbyDB_hostname)
                         (PORT=StandbyDB_port)
                         (DB_NAME=cluster_DB_NAME)
               )
)
```

Primary와 Standby의 $TB_SID.tip 파일을 설정할 때 공통의 DB_NAME을 각각 SID별로 설정해야 한다.

참고

Primary에서 장애가 발생하면 Standby에 자동으로 접속하는 방법이 있다. 이 방법에 대한 자세한 내용은 “Appendix A. tbdsn.tbr”를 참고한다.

8.6. 제약 사항

Tibero Standby Cluster의 기능은 Primary에서 생성된 Redo 로그를 그대로 Standby에 전송하므로 서로 간의 컴퓨팅 환경(CPU bus size, endianness, OS 등)이 동일해야 하고, $TB_SID.tip 파일의 데이터베이스 블록의 크기도 동일해야 한다.

DB 파일의 변화를 일으키는 테이블스페이스, 데이터 파일의 추가 및 삭제와 같은 DDL 문장은 허용하지 않으며, Standby 없이 수행해야 한다. 즉, 데이터베이스를 백업하여 Standby에 적용해야 하는 제약이 있다.

8.7. Tibero Standby Cluster의 정보 조회

Tibero RDBMS에서는 Tibero Standby Cluster의 상태 정보를 제공하기 위해 다음 표에 나열된 동적 뷰를 제공하고 있다.

동적 뷰	설명
V$STANDBY_DEST	Primary에 설정된 각 Standby의 연결 정보 및 Redo 로그의 전송 상태를 조회하는 뷰이다. Primary에서만 이 뷰를 사용할 수 있다.
V$STANDBY	Standby에서 Primary의 연결 정보와 전달 받은 Redo 로그 그리고 이미 반영된 Redo 로그의 상태를 조회하는 뷰이다. Standby에서만 이 뷰를 사용할 수 있다.

참고

동적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

제9장 Tibero Cluster Manager

본 장에서는 Tibero Cluster Manager의 기본 개념과 동작 모드, 멤버십 관리, 환경설정 및 실행 방법을 설명한다.

9.1. 개요

Tibero Cluster Manager(이하 TBCM)는 클러스터(Cluster)의 가용성을 높이고 관리의 편의를 지원하는 Tibero RDBMS의 부가 기능이다.

TBCM은 클러스터에 속한 전체 노드의 Tibero RDBMS 서버의 동작 상태를 지속적으로 파악한다. 구체적으로 클러스터를 구성하는 TBCM의 노드들은 네트워크 또는 공유디스크를 통해 주기적으로 자신이 살아있음을 알리는 heartbeat 메시지를 서로 주고 받으면서 주변 노드들의 상태를 파악한다. 클러스터의 특정 노드에서 동작하고 있는 Tibero RDBMS 서버가 비정상적인 경우에는, TBCM이 감지하여 클러스터의 멤버십 구성을 자동으로 변경한다. 즉 전체 클러스터의 서비스가 중단되지 않도록 하기 위해서이다.

9.2. TBCM의 동작 모드

TBCM의 동작 모드는 다음과 같이 두 가지로 구성된다.

동작 모드	설명
ACTIVE_SHARED	SHARED 접미사를 가지는 동작 모드는 공유 디스크에 **TBCM 클러스터 파일**을 저장하여 클러스터를 관리한다. **ACTIVE_SHARED 동작 모드**는 TAC (Tibero Active Cluster)를 위해 사용하는 모드로, 공유 디스크를 기반으로 구성한 클러스터에 속하는 모든 노드의 Tibero RDBMS가 클러스터 서비스에 함께 참여한다. 서비스 중에 특정 노드가 동작을 멈춰도 나머지 노드의 서비스가 중지되지 않도록 클러스터의 구성을 자동으로 변경시키는 기능을 수행한다.
ACTIVE_REPLICATION	REPLICATION 접미사를 가지는 동작 모드는 공유 디스크 대신 **TBCM 코디네이터 데몬**을 이용하여 클러스터를 관리한다. **ACTIVE_REPLICATION 동작 모드**는 Tibero MMDBMS를 위해 사용하는 모드로, 이중화 기능으로 연결된 클러스터에서 서비스 하는 특정 노드가 동작을 멈춰도 나머지 노드의 서비스가 중지되지 않도록 클러스터의 구성을 자동으로 변경시키는 기능을 수행한다. 자세한 내용은 "Tibero MMDBMS 관리자 안내서"를 참고한다.

9.3. TBCM의 멤버십 관리

TBCM에서 멤버십(Membership)을 관리하는 방법은 다음과 같다.

- 자동 멤버십 관리

  자동 멤버십 관리 방법으로 TBCM 클러스터 파일을 생성한 경우에는 별도의 멤버십 변경 작업을 하지 않아도 새로운 노드에 TBCM 데몬을 실행한 경우 동작 중인 클러스터 멤버십에 자동으로 참여할 수 있다.

  TBCM 클러스터 파일을 생성할 때, TBCM_NODES 초기화 파라미터를 설정하지 않으면 자동 멤버십 관리 방법으로 TBCM 클러스터 파일이 생성된다.

- 수동 멤버십 관리

  수동 멤버십 관리 방법으로 TBCM 클러스터 파일을 생성한 경우에는 TBCM의 클러스터에 새로운 TBCM 노드를 추가할 때 반드시 수동으로 멤버십 변경 작업을 먼저 수행해야 새로운 노드가 멤버십에 참여할 수 있다.

  새로운 TBCM에 노드를 추가 또는 삭제하기 위해서는 **tbcm -n** 명령어를 이용하여 해당 노드를 기존의 TBCM 클러스터에 수동으로 등록하거나 삭제해야 한다. 자세한 내용은 "9.6. TBCM의 실행"을 참고한다.

9.4. TBCM의 환경설정

TBCM은 Tibero RDBMS와 **환경변수** 및 **환경설정 파일**을 공유한다. 따라서 **TB_HOME, TB_SID** 환경변수를 설정하고 TBCM을 사용하기 위한 초기화 파라미터를 $TB_SID.tip 파일에 설정해야 한다.

9.4.1. 환경변수의 설정

환경변수를 설정하는 방법은 다음과 같다.

- $TB_HOME

  티베로가 설치된 경로를 지정한다.

  ```
  TB_HOME=/home/tibero/tibero4
  ```

- $TB_SID

  노드를 식별할 수 있는 ID를 지정한다. 각 노드는 서로 다른 값을 가져야 한다.

  ```
  TB_SID=tac
  ```

9.4.2. 환경설정 파일의 설정

TBCM을 사용하기 위해서는 $TB_SID.tip 환경설정 파일에 초기화 파라미터를 설정해야 한다.

다음은 TBCM의 동작 모드별로 TBCM의 사용 환경을 설정하는 예이다.

- ACTIVE_SHARED 모드

  <$TB_SID.tip>

```
# 필수 입력 항목
TBCM_CLUSTER_MODE=ACTIVE_SHARED
TBCM_PORT=6000
LOCAL_CLUSTER_ADDR=192.168.1.1
LOCAL_CLUSTER_PORT=8000
TBCM_FILE_NAME=/path/to/tbcm/file

# 선택 입력 항목
TBCM_NAME=node1
TBCM_NODE_ID=1
TBCM_NODES=node1@192.168.1.1:8000;node2@192.168.1.2:8000
TBCM_TIME_UNIT=10
TBCM_HEARTBEAT_EXPIRE=300
TBCM_WATCHDOG_EXPIRE=200
TBCM_DOWN_CMD=/path/to/tbcm_down_cmd.sh
LOCAL_CLUSTER_VIP=192.168.100.1
LOCAL_CLUSTER_NIC=eth0
LOCAL_CLUSTER_VIP_NETMASK=255.255.255.0
LOCAL_CLUSTER_VIP_BROADCAST=192.168.100.255
LOG_LVL_TBCM=2
TBCM_LOG_DEST=/path/to/log/tbcm/
TBCM_LOG_FILE_SIZE=10485760
TBCM_LOG_TOTAL_SIZE_LIMIT=4294967296
```

- ACTIVE_REPLICATION 모드

  다음은 코디네이터 노드의 환경설정한 예이다.

  <$TB_SID.tip>

```
# 필수 입력 항목
TBCM_CLUSTER_MODE=ACTIVE_REPLICATION
TBCM_PORT=6000
LOCAL_CLUSTER_ADDR=192.168.1.1
TBCM_FILE_NAME=/path/to/tbcm/file
TBCM_COORDINATOR_MODE=Y

# 선택 입력 항목
```

```
TBCM_NAME=coordinator
TBCM_NODE_ID=1
TBCM_TIME_UNIT=10
TBCM_HEARTBEAT_EXPIRE=300
LOG_LVL_TBCM=2
TBCM_LOG_DEST=/path/to/log/tbcm/
TBCM_LOG_FILE_SIZE=10485760
TBCM_LOG_TOTAL_SIZE_LIMIT=4294967296
```

- ACTIVE_REPLICATION 모드

  다음은 Worker 노드의 환경설정에 대한 예이다.

  <$TB_SID.tip>

```
# 필수 입력 항목
TBCM_CLUSTER_MODE=ACTIVE_REPLICATION
TBCM_PORT=7000
LOCAL_CLUSTER_ADDR=192.168.1.2
TBCM_FILE_NAME=/path/to/tbcm/file
TBCM_COORDINATOR_NAME=coordinator@192.168.1.1:6000

# 선택 입력 항목
TBCM_NAME=worker1
TBCM_NODE_ID=10
TBCM_TIME_UNIT=10
TBCM_HEARTBEAT_EXPIRE=300
TBCM_DOWN_CMD=/path/to/tbcm_down_cmd.sh
LOCAL_CLUSTER_VIP=192.168.100.1
LOCAL_CLUSTER_NIC=lan1
LOCAL_CLUSTER_VIP_NETMASK=255.255.255.0
LOCAL_CLUSTER_VIP_BROADCAST=192.168.100.255
LOG_LVL_TBCM=2
TBCM_LOG_DEST=/path/to/log/tbcm/
TBCM_LOG_FILE_SIZE=10485760
TBCM_LOG_TOTAL_SIZE_LIMIT=4294967296
```

다음은 TBCM를 사용하기 위해 $TB_SID.tip 환경설정 파일에 설정할 수 있는 초기화 파라미터에 대한 설명이다.

초기화 파라미터	설명
TBCM_CLUSTER_MODE	필수 입력 항목으로, TBCM의 동작 모드를 설정한다. 다음의 값 중에서 하나를 선택하여 설정할 수 있다. - ACTIVE_SHARED

초기화 파라미터	설명
	- ACTIVE_REPLICATION
TBCM_PORT	필수 입력 항목으로, TBCM 데몬에서 접속을 받는 용도로 사용하는 포트 번호이다. (기본값: 8631)
TBCM_FILE_NAME	필수 입력 항목으로, 현재 동작 중인 클러스터의 상태를 저장하는 **TBCM 클러스터 파일**의 경로명을 설정한다. TBCM 클러스터 파일은 **tbcm -c** 명령어로 생성된다. 자세한 내용은 "9.6. TBCM의 실행"을 참고한다. TBCM의 동작 모드에 따라 TBCM 클러스터 파일의 경로명을 설정하는 방법은 다음과 같다. - ACTIVE_SHARED: TBCM 클러스터 파일로 할당된 공유 디스크 파티션의 경로명을 설정한다. 만약 클러스터 파일 시스템을 사용하는 환경이라면, 클러스터 파일 시스템상의 파일 경로명을 설정한다. - ACTIVE_REPLICATION: 각 로컬 디스크에서 클러스터의 구성 정보를 임시로 저장할 파일의 경로명을 설정한다.
LOCAL_CLUSTER_ADDR	필수 입력 항목으로, 각각의 노드에서 TBCM 데몬 간의 통신을 위해 사용하는 개인 IP(Private IP)를 설정한다. TBCM 데몬 간의 통신뿐만 아니라, 클러스터를 구성하는 Tibero RDBMS 서버 간의 통신 (또는 클러스터를 구성하는 Tibero MMDBMS 서버 간의 통신)을 위해서도 사용된다.
LOCAL_CLUSTER_PORT	클러스터를 구성하는 Tibero RDBMS 서버 간의 통신 (또는 클러스터를 구성하는 Tibero MMDBMS 서버 간의 통신)을 위해 사용되는 포트 번호를 설정한다.
TBCM_NAME	클러스터에서 각각의 노드를 구분하기 위해 **노드 식별 이름**에서 사용될 접두사를 설정한다. 생략하면, 기본값은 **tbcm**으로 설정된다. 노드 식별 이름은 다음과 같은 형태로 설정한다. **[노드 이름]@[IP 주소]:[포트 번호]** - [노드 이름]: TBCM_NAME 초기화 파라미터로 설정된 문자열로, 설정하지 않을 경우 tbcm이라는 문자열이 설정된다. - [IP 주소]: LOCAL_CLUSTER_ADDR 초기화 파라미터에 설정된 IP 주소이다. - [포트 번호]: TBCM_PORT 초기화 파라미터에 설정된 포트 번호이다.

초기화 파라미터	설명
	예를 들어, 다음과 같이 설정이 된 경우 해당 노드의 식별 이름은 **node1@192.168.1.1:8000** 이 된다. - TBCM_NAME=node1 - LOCAL_CLUSTER_ADDR=192.168.1.1 - TBCM_PORT=8000
TBCM_NODE_ID	Tibero MMDBMS에서 각각의 노드를 구분하기 위해 숫자로 설정한 ID이다. Tibero MMDBMS의 경우 각 노드의 TBCM 데몬은 **노드 식별 이름** 이외에도 노드 ID라는 숫자를 통해 각 노드를 구분한다. 본 초기화 파라미터를 설정하지 않은 노드의 경우는, TBCM 데몬을 부팅할 때 클러스터 내에서 유일한 노드 ID를 자동으로 부여받게 된다. 반면에, TBCM 데몬의 노드 ID를 사용자가 설정하기 위해서는 TBCM_NODE_ID 초기화 파라미터에 설정된 해당 노드 ID를 설정하면 된다. 전체 클러스터에 참여한 TBCM 데몬의 모든 노드 ID는 유일해야 하므로, 동작 중인 클러스터의 다른 노드에서 해당 TBCM_NODE_ID 초기화 파라미터에 설정된 노드 ID를 사용 중이라면 TBCM을 부팅할 때 에러가 발생된다.
TBCM_NODES	**수동 멤버십 관리 방법**으로 TBCM 클러스터 파일을 생성할 때 설정한다. 자세한 내용은 “9.3. TBCM의 멤버십 관리”를 참고한다. 세미콜론(;)으로 각 **노드 식별 이름**을 구분하여, 다음과 같이 표현한다. **TBCM_NODES="node1@192.168.1.1:8000;node2@192.168.1.2:8000;node3@192.168.1.2:8001"**
TBCM_TIME_UNIT	TBCM에서 사용하는 디폴트 시간 단위(1 tick)를 설정한다. TBCM은 단위 시간마다 자신의 노드에 설치된 Tibero RDBMS 서버와 다른 노드의 현재 상태를 확인한다. 시간은 0.1초 단위로 설정한다. (기본값: 10(= 1초))
TBCM_HEARTBEAT_EXPIRE	heartbeat의 제한 시간을 설정한다. 본 초기화 파라미터로 설정된 시간 동안 heartbeat이 전달 되지 않는 노드가 있는 경우, 다른 노드의 TBCM 데몬은 해당 노드를 장애 상태로 판단한다. (기본값: 300 ticks) TBCM의 동작 모드에 따라 heartbeat의 제한 시간을 설정하는 방법은 다음과 같다.

초기화 파라미터	설명
	- ACTIVE_SHARED: TBCM 데몬 간에는 네트워크 및 공유 디스크를 통해 heartbeat을 전달하여 각 노드의 Tibero RDBMS의 동작 상태를 파악한다. - ACTIVE_REPLICATION: 공유 디스크 대신 코디네이터 노드(Coordinator Node)를 이용하여 heartbeat을 전달한다.
TBCM_WATCHDOG_EXPIRE	ACTIVE_SHARED 모드의 경우 디스크 I/O 장애 등으로 인해 TBCM 데몬의 동작이 멈출 수 있다. 정상적인 다른 노드에서는 비정상적인 노드의 TBCM 데몬이 heartbeat을 보내지 못하기 때문에 비정상적인 노드를 시스템 중단 상태라고 판단하고, 해당 노드의 Tibero RDBMS 서버를 제외하고 새로운 멤버십 구성을 하게 된다. 만약 비정상적인 노드의 Tibero RDBMS 서버가 여전히 공유 디스크를 계속 이용하여 작업을 수행 중이라면, 다른 노드의 멤버십 상태와 불일치가 발생하게 된다. 이로 인해 데이터베이스의 일관성이 문제가 된다. 이러한 현상을 해결하기 위해서 TBCM은 **Watchdog**이라는 기능을 제공한다. 본 초기화 파라미터에 설정된 시간 동안 TBCM 데몬이 반응이 없을 경우 해당 노드의 Tibero RDBMS 서버는 동작을 자동으로 멈추고 셧다운(shutdown)을 한다. 예를 들어, TBCM_WATCHDOG_EXPIRE 초기화 파라미터에 설정된 값을 TBCM_HEARTBEAT_EXPIRE 초기화 파라미터보다 작게 설정했을 경우, 정상적인 다른 노드에서 비정상적인 노드의 heartbeat에 대한 타임아웃이 발생하게 되면, 해당 노드에서 Watchdog 기능에 의해 Tibero RDBMS 서버가 반드시 셧다운하였음을 보장받을 수 있다. (기본값: 290 ticks)
TBCM_DOWN_CMD	클러스터를 구성하는 노드 간의 통신에 사용되는 interconnect 네트워크의 연결이 끊어진 경우에는 Tibero RDBMS의 서비스가 중단되는 현상이 발생할 수 있는데, 이를 **split brain 현상**이라고 한다. TBCM은 split brain 현상이 발생했는지를 감시한다. split brain 현상이 발생했을 때, 문제가 되는 노드의 Tibero RDBMS 서버를 강제로 종료시킨다. 이를 통해 나머지 노드는 서비스를 지속적으로 제공할 수 있다. 이 때, 본 초기화 파라미터에 설정된 스크립트 파일을 이용하여 문제가 되는 노드의 Tibero RDBMS를 강제적으로 종료 시킨다. Tibero RDBMS를 관리하는 시스템 관리자는 반드시 본 초기화 파라미터에 설정된 스크립트 파일의 명령이 정상적으로 동작하는지를 사전에 반드시 확인해야 한다. (기본값: "$TB_HOME/scripts/tbcm_down_cmd.sh")

초기화 파라미터	설명
TBCM_COORDINATOR_MODE	ACTIVE_REPLICATION 모드에서만 사용되는 초기화 파라미터로, 공유디스크가 존재하지 않는 경우 별도의 **코디네이터 노드(Coordinator Node)**로 동작하는 TBCM 데몬이 필요하다. 각 노드별 TBCM 데몬은 코디네이터 노드를 통해서 클러스터의 구성 정보를 공유한다. 코디네이터 노드로 동작할 TBCM의 경우 본 초기화 파라미터의 값을 **Y**로 설정해야 하며, Worker 노드(코디네이터 노드가 아닌 나머지 노드)의 경우 본 초기화 파라미터를 설정하지 않거나, 값을 **N**으로 설정해야 한다. 이때 코디네이터 노드가 아닌 다른 노드의 경우에는 코디네이터 노드의 노드 식별 이름을 설정해야 한다. (기본값: N)
TBCM_COORDINATOR_NAME	ACTIVE_REPLICATION 모드에서 동작하는 클러스터에서 코디네이터 노드 이외의 Worker 노드에서 코디네이터 노드의 노드 식별 이름을 설정한다. Worker 노드에서는 반드시 코디네이터 노드의 노드 식별 이름을 미리 설정해야만 TBCM이 정상적으로 동작한다.
LOCAL_CLUSTER_VIP	클러스터를 구성하는 Tibero RDBMS 서버가 가상 IP 주소를 이용하여 서비스하는 경우, LOCAL_CLUSTER_VIP 초기화 파라미터에 각각의 노드에서 사용할 가상 IP 주소를 설정한다. 클러스터를 구성할 때 특정한 노드에서 장애가 발생한 경우 나머지 노드로, 클라이언트 접속을 Failover 하게 되는데, 기존에 연결된 클라이언트의 네트워크 접속을 빠르게 Failover 하기 위해서는 가상 IP를 이용한 Failover 기능을 구성해야 한다. 이 설정을 할 경우, TBCM을 반드시 OS 관리자 권한으로 기동시켜야 한다.
LOCAL_CLUSTER_NIC	각각의 Tibero RDBMS 서버의 노드가 가상 IP 주소를 이용해 서비스를 하는 경우, 로컬 서버에서 가상 IP를 할당할 NIC의 이름을 반드시 설정한다.
LOCAL_CLUSTER_VIP_NETMASK	각각의 Tibero RDBMS 서버의 노드가 가상 IP 주소를 이용해 서비스를 하는 경우, 로컬 서버에서 가상 IP의 네트워크 마스크(network mask)를 설정한다.
LOCAL_CLUSTER_VIP_BROADCAST	각각의 Tibero RDBMS 서버의 노드가 가상 IP 주소를 이용해 서비스를 하는 경우, 로컬 서버에서 가상 IP의 broadcasting 주소를 설정한다.
LOG_LVL_TBCM	TBCM 로그의 출력 레벨을 설정한다. 1 ~ 6 사이의 값 중에서 설정할 수 있으며, 값이 클수록 더 많은 로그가 출력된다. (기본값: 2)

초기화 파라미터	설명
TBCM_LOG_DEST	TBCM 로그가 저장되는 파일의 디렉터리 경로이다. 단, 절대 경로로 설정을 해야 한다. (기본값: $TB_HOME/instance/$TB_SID/log/tbcm/)
TBCM_LOG_FILE_SIZE	TBCM 로그 파일 하나의 최대 크기를 설정한다. TBCM 로그 파일의 크기가 설정된 값 보다 커지면, 현재의 로그 파일을 백업하고 새로운 로그 파일을 생성한다. (기본값: 10MB)
TBCM_LOG_TOTAL_SIZE_LIMIT	TBCM_LOG_DEST 초기화 파라미터로 설정된 디렉터리에 저장되는 각 로그 파일의 크기를 합한 총합의 크기이다. 설정 값을 초과하는 로그가 발생하는 경우, 가장 오래된 로그 파일을 삭제하고 새로운 로그 파일을 생성한다. (기본값: 4GB)
CM_FENCE	ACTIVE_SHARED 모드에서 공유 스토리지와 물리적 연결이 끊어진 장애가 발생할 경우, 과 서버가 공유 스토리지에 파일을 제대로 읽거나 쓸 수 가 없어 멈추어 있을 가능성이 있다. CM_FENCE 기능을 켜면 이러한 장애 상황을 감지하는 IO fence 데몬이 독립적으로 실행되어,장애가 발생한 경우로 간주하여 데이베이스를 보호하기 위하여 시스템를 강제로 재부팅시켜 준다. IO fence 데몬의 장애 감지와 관련된 파라미터는 CM_WATCHDOG_EXPIRE이다. IO fence 데몬은 으로부터 CM_WATCHDOG_EXPIRE(초) 만큼의 시간동안 heartbeat를 전달받지 못하는 경우 장애로 판단한다. 이 설정을 할 경우, 을 반드시 OS 관리자 권한으로 기동시켜야 한다. (기본값: N)

9.5. VIP

9.5.1. VIP 설정

는 VIP(Virtual IP)를 이용하여 서비스를 할 수 있다. 가상 IP를 이용하여 서비스를 하고자 할 경우, 아래와 같은 관련된 초기화 파라미터를 설치 환경에 맞게 설정해주어야 한다.

```
LOCAL_CLUSTER_VIP=192.168.100.1
LOCAL_CLUSTER_NIC=eth0
LOCAL_CLUSTER_VIP_NETMASK=255.255.255.0
LOCAL_CLUSTER_VIP_BROADCAST=192.168.100.255
```

은 OS 명령을 통해서 시스템에 VIP를 설정하고 해제한다. 따라서 VIP 기능을 사용하기 위해서 은 OS의 관리자 권한으로 실행되어야 한다. 위와 같은 초기화 파라미터를 설정한 후, 을 관리자 권한으로 기동시키면 해당 시스템에 VIP가 추가되며, ifconfig 와 같은 OS 명령을 통해서 이를 확인할 수 있다.

9.5.2. VIP failover

클러스터의 특정 노드에서 장애가 발생하여 그 노드의 까지 죽은 경우, 정상적으로 작동하는 노드들 중에 한 노드의 이 죽은 노드의 VIP를 자신의 돌고 있는 시스템에 설정한다. 이를 통해서 는 죽은 노드의 VIP로도 계속해서 서비스를 할 수 있다. 만약 죽은 노드가 다시 살아나면 죽은 노드의 VIP를 자신의 시스템에 설정했던 은 그 VIP를 해제하고, 살아난 노드의 은 VIP를 자신의 시스템에 다시 설정한다.

9.6. TBCM의 실행

TBCM를 실행하기 위해서는 반드시 TB_HOME 과 TB_SID 환경변수가 적절히 설정되어 있는지를 먼저 확인해야 한다. 자세한 내용은 "9.4. TBCM의 환경설정"을 참고한다.

TBCM의 실행 파일은 $TB_HOME/bin 디렉터리에 있는 tbcm 파일이다.

tbcm 파일의 사용 방법은 다음과 같다.

```
$ tbcm [옵션] [추가 모드]
```

옵션	설명
-b [LOCK|UNLOCK]	TBCM 데몬을 실행한다. $TB_SID.tip 환경설정 파일에 설정한 초기화 파라미터에 따라 동작한다. TBCM 데몬이 이미 동작 중인 경우, 같은 포트 번호로 데몬을 실행하는 것은 불가능하다. 이러한 경우에는 기존의 TBCM 데몬을 종료시키거나 $TB_SID.tip 환경설정 파일에서 포트 번호를 수정한 후 다시 실행해야 한다. 다음은 -b 옵션에서 추가적으로 사용할 수 있는 모드이다. - LOCK: TBCM 데몬이 Tibero RDBMS를 강제 종료시키지 못하는 command lock 모드로 TBCM 데몬을 부팅한다. - UNLOCK: 기본값으로, TBCM 데몬이 TBCM_DOWN_CMD 초기화 파라미터를 이용하여 Tibero RDBMS를 강제로 종료할 수 있는 command unlock 모드로 TBCM 데몬을 부팅한다. (예: tbcm -b LOCK)
-d	TBCM 데몬을 종료시킨다. TBCM 데몬이 실행 중이지 않을 때는 에러가 출력된다.
-s	TBCM 데몬이 실행 중인 경우, 현재 클러스터의 상태 정보를 화면에 출력한다. TBCM 데몬이 실행 중이지 않을 때는 에러가 출력된다.

옵션	설명
	$TB_HOME/scripts 디렉터리에 있는 tbcm_stat.sh 스크립트 파일을 실행시키면 1초에 한 번씩 주기적으로 tbcm -s 명령어가 실행되므로 클러스터의 상태 변화를 지속적으로 확인 할 수 있다.
-v	TBCM의 버전 정보를 출력한다.
-l	TBCM 데몬이 실행 중일 때, TBCM이 Tibero RDBMS를 강제로 종료시키지 못하도록 command lock 모드로 변경한다.
-u	TBCM이 Tibero RDBMS를 자동으로 종료시킬 수 있도록 command unlock 모드로 변경한다.
-c	TBCM에서 클러스터의 구성 상태를 저장하는 **TBCM 클러스터 파일**의 내용을 초기화 시킨다. 만약 TBCM_FILE_NAME 초기화 파라미터에 설정된 파일이 존재하지 않는 경우, 해당 파일을 생성한 후 초기화를 한다. 한번 초기화된 내용은 이전 상태로 복구할 수 없으므로, 주의해야 한다. 이미 클러스터가 구성된 TBCM 데몬이 실행 중이라면, -c 옵션으로 TBCM 클러스터 파일을 초기화하는 것은 데이터베이스의 정합성을 해칠 수 있으므로 주의해야 한다.
-n	동작 중인 TBCM 클러스터에 새로운 노드를 추가하거나, 이미 클러스터에 포함된 노드를 삭제한다. TBCM 클러스터가 수동 멤버십 관리 방법으로 동작 중인 경우에는 반드시 본 옵션을 통해 노드를 추가하거나 삭제해야 한다. 예를 들면 다음과 같다. - 노드 추가: tbcm -n add node1@192.168.1.1:8000 - 노드 삭제: tbcm -n del node2@192.168.1.2:8000

제10장 Tibero Active Cluster

본 장에서는 Tibero Active Cluster의 기본 개념과 구성요소, 프로세스, 실행 및 운영 방법을 설명한다.

10.1. 개요

Tibero Active Cluster (이하 TAC)는 확장성, 고가용성을 목적으로 제공하는 Tibero RDBMS의 주요 기능이다. TAC 환경에서 실행 중인 모든 인스턴스는 공유된 데이터베이스를 통해 트랜잭션을 수행하며, 공유된 데이터에 대한 접근은 데이터의 일관성과 정합성 유지를 위해 상호 통제하에 이뤄진다.

큰 업무를 작은 업무의 단위로 나누어 여러 노드 사이에 분산하여 수행할 수 있기 때문에 업무 처리 시간을 단축할 수 있다.

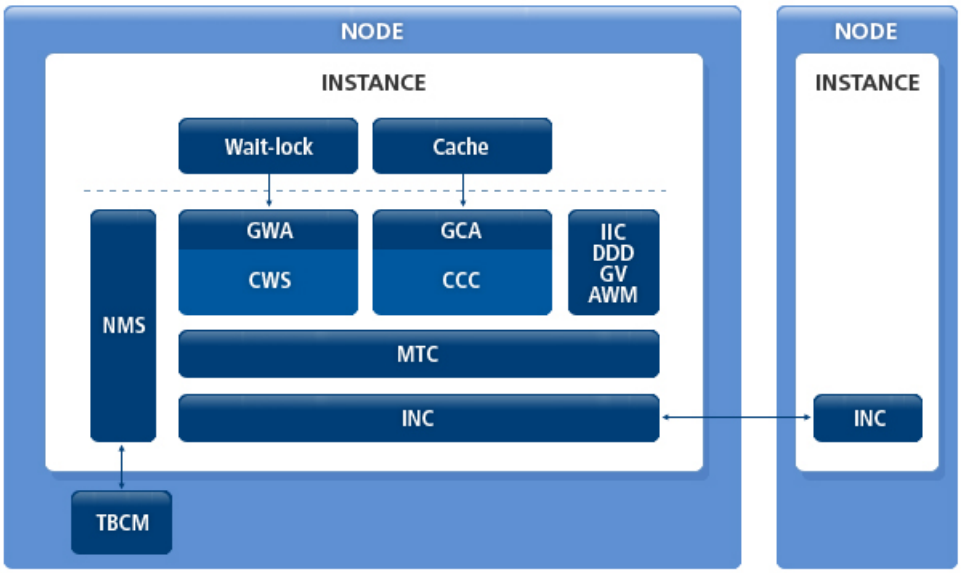
여러 시스템이 공유 디스크를 기반으로 데이터 파일을 공유한다. TAC 구성에 필요한 데이터 블록은 노드 간을 연결하는 고속 사설망을 통해 주고 받음으로써 노드가 하나의 공유 캐시(shared cache)를 사용하는 것처럼 동작한다.

운영 중에 한 노드가 멈추더라도 동작 중인 다른 노드들이 서비스를 지속하게 된다. 이러한 과정은 투명하고 신속하게 처리된다.

10.2. 구성요소

다음은 TAC의 구조를 나타내는 그림이다.

[그림 10.1] TAC의 구조



TAC의 구조는 다음과 같은 모듈로 구성되어 있다.

- CWS(Cluster Wait-lock Service)
  - 기존 Wait-lock(이하 Wlock)이 클러스터 내에서 동작할 수 있도록 구현된 모듈이다. Distributed Lock Manager(이하 DLM)이 내장되어 있다.
  - Wlock은 GWA를 통해 CWS에 접근할 수 있으며, 이와 관련된 배경 프로세스로는 LASW, LKDW, RCOW 이 존재한다.
  - Tibero RDBMS 3에서는 Wlock이 싱글 인스턴스의 리소스만 보호했으나, 멀티 인스터스를 지원하는 TAC 환경에서는 CWS를 통해 다른 노드와의 동기화를 통제할 수 있다.
- GWA(Global Wait-lock Adapter)
  - Wlock은 CWS를 사용하기 위한 인터페이스 역할을 수행하는 모듈이다.
  - CWS에 접근하기 위한 핸들인 CWS Lock Status Block(이하 LKSB)과 파라미터를 설정하고 관리한다.
  - Wlock에서 사용하는 잠금 모드(Lock mode)와 타임아웃(timeout)을 CWS에 맞게 변환하며, CWS에서 사용할 Complete Asynchronous Trap(이하 CAST), Blocking Asynchronous Trap(이하 BAST)을 등록할 수 있다.
- CCC(Cluster Cache Control)
  - 데이터베이스의 데이터 블록에 대한 클러스터 내 접근을 통제하는 모듈이다. DLM이 내장되어 있다.
  - CR Block Server, Current Block Server, Global Dirty Image, Global Write 서비스가 포함되어 있다.
  - Cache layer에서는 GCA(Global Cache Adapter)를 통해 CCC 에 접근할 수 있으며, 이와 관련된 배경 프로세스로 LASC, LKDC, RCOC 이 존재한다.
- GCA(Global Cache Adapter)
  - Cache layer에서 CCC 서비스를 사용하기 위한 인터페이스 역할을 수행하는 모듈이다.
  - CCC에 접근하기 위한 핸들인 CCC LKSB와 파라미터를 설정하고 관리하며, Cache layer에서 사용하는 block lock mode를 CCC에 맞게 변환한다.
  - CCC의 lock-down event에 맞춰 데이터 블록이나 Redo 로그를 디스크에 저장하는 기능과 DBWR가 Global wirte를 요청하거나 CCC에서 DBWR에게 block write를 요구하는 인터페이스를 제공한다.
  - CCC에서는 GCA를 통해 CR block, Global dirty block, current block을 주고받는다.

- MTC(Message Transmission Control)
  - 노드 간의 통신 메시지의 손실과 out-of-order 문제를 해결하는 모듈이다.
  - 문제를 해결하기 위해 retransmission queue와 out-of-order message queue를 관리한다.
  - General Message Control(GMC)을 제공하여, CWS/CCC 이외의 모듈에서 노드 간의 통신이 안전하게 이루어지도록 보장한다.

    현재 Inter-instance call(IIC), Distributed Deadlock Detection(이하 DDD), Automatic Workload Management에서 노드 간의 통신을 위해 GMC를 사용하고 있다.

- INC(Inter-Node Communication)
  - 노드 간의 네트워크 연결을 담당하는 모듈이다.
  - INC를 사용하는 사용자에게 네트워크 토폴로지(network topology)와 프로토콜을 투명하게 제공하며 TCP, UDP등의 프로토콜을 관리한다.

- NMS(Node Membership Service)
  - TBCM으로부터 전달받은 정보(node id, ip address, port, incarnamtion number)와 node workload를 나타내는 가중치(weight)를 관리하는 모듈이다.
  - node membership의 조회, 추가, 삭제 기능을 제공하며 이와 관련된 배경 프로세스로 NMGR이 있다.

10.3. 프로세스

TAC는 **8개**의 프로세스가 추가로 생성된다. 이러한 프로세스는 다음과 같은 그룹에 각각 포함된다.

- **Active Cluster Server(Message handler)**
  - ACSn

    클러스터 내의 원격 노드로부터 CWS/CCC의 lock operation과 reconfiguration 요청(request)을 받아 처리하고 응답하는 프로세스이다.

    ACSn 프로세스는 다음과 같은 특징이 있다.

    - CR block request를 받아 주어진 스냅샷(snapshot)에 해당하는 CR block을 생성하고 요청자에게 전송한다.
    - Current block request를 받으면 local block cache에 존재하는 Current block을 읽어서 요청자에게 전송한다.

- global write request를 받아 주어진 데이터 블록이 dirty이면 BLKW가 이를 디스크에 기록하도록 지시한다.
- MTC IIC request를 받아 처리한다.
- MLD(Master Lookup Directory) lookup/remove request를 받아 처리한다.
- ACS는 여러 개가 실행될 수 있기 때문에 숫자(1, 2, 3, ..., n)로 구분한다.
- 4종류의 스레드(thread)로 구성된다.

스레드	설명
acfm (active cluster frame monitor)	ACS의 메인 스레드로서, 나머지 스레드를 감독한다. - stop - resume - kill
dspc(dispatcher)	원격 노드로부터 메시지가 도착하기를 기다리며, 메시지가 도착하면 해당 소켓을 rcvr 스레드로 전달한다.
rcvr(receiver)	dspc 스레드로부터 받은 소켓을 읽어서 해당 메시지의 요청을 처리한다. rcvr 스레드는 모든 dspc 스레드로부터 사용할 수 있는 공통 풀(pool)로 생성되며, ACF_RCVR_CNT 초기화 파라미터에 설정된 값만큼 생성된다.
sndr(sender)	세션으로부터 전송해야 할 메시지를 넘겨 받고, 보내야 할 원격 노드의 소켓을 찾아 네트워크로 보내는 역할을 수행한다.

- **Lock Assistant Server**
  - LASW, LASC

    CWS/CCC에서 세션을 담당하는 워킹 스레드가 처리해야 할 비동기적 업무를 대신 수행하는 프로세스이다.

    LASW, LASC 프로세스는 다음과 같은 특징이 있다.

    - BAST를 맞거나 스스로 잠금을 설정할 때, 캐시된 lock mode를 downgrade하기 전에 BLKW로부터 disk write notification이나 LOGW로부터 log flush를 기다린 후, master에게 lock downgrade를 통보한다. (이 특징은 LASC 프로세스만이 수행할 수 있다.)
    - BLKW가 master로부터 받은 global write request 처리를 완료한 후 LASC에게 통보해 준다. 또한, master에게 write done notify를 보낸다. 이 특징은 LASC 프로세스만이 수행할 수 있다.

- shadow resource block를 reclaim하기 전에 master에게 MC lock에 대한 제거 요청을 보낸다. 요청을 보낸 후, 이에 대한 응답을 받아서 처리한다.

- **Lock Daemon**

  – LKDW, LKDC

  주기적으로 lock resource을 관리하고 타임아웃을 체크하는 프로세스이다.

  LKDW, LKDC 프로세스는 다음과 같은 특징이 있다.

  - DDD(Distributed Deadlock Detection)를 수행하기 위해 주기적으로 타임아웃이 발생한 lock waiter를 체크한다. 체크 시 교착 상태가 발생하면 DDD를 시작한다.
  - MTC retransmission queue에 설정된 메시지를 주기적으로 검사해서 타임아웃이 발생한 메시지를 다시 전송한다.
  - 주기적으로 TSN의 동기화를 수행한다. (이 특징은 LKDW 프로세스만이 수행할 수 있다.)
  - lock resource를 위한 공유 메모리가 부족하게 되면 resource block reclaiming을 시작하여 필요한 리소스를 확보한다.

- **Reconfiguration process**

  – RCOW, RCOC

  NMGR 프로세스로부터 node join 및 leave event를 받아 CWS/CCC lock remastering/recovery을 수행한다.

- **Node Manager**

  – NMGR

  TBCM과 통신하여 node join 및 leave event를 받아 처리하며, node membership을 관리한다. 또한, RCOC, RCOW 프로세스에 의해 수행되는 CWS/CCC reconfiguration을 통제(suspend 또는 resume)한다.

10.4. TAC 환경설정

TAC는 기본적으로 싱글 인스턴스일 때의 설정은 그대로 사용한다. 이 외에는 추가로 설정해야 할 초기화 파라미터와 주의 사항이 있다.

다음은 TAC를 사용하기 위해 추가로 설정해야 하거나 주의해야 하는 초기화 파라미터의 예이다.

<$TB_SID.tip>

```
TOTAL_SHM_SIZE=4096M
DB_CACHE_SIZE=2048M
CLUSTER_DATABASE=Y
THREAD=0
UNDO_TABLESPACE=UNDO0
LOCAL_CLUSTER_ADDR=192.168.1.1
LOCAL_CLUSTER_PORT=12345
TBCM_CLUSTER_MODE=ACTIVE_SHARED
```

초기화 파라미터	설명
TOTAL_SHM_SIZE	인스턴스가 사용할 전체 공유 메모리의 크기를 설정한다.
DB_CACHE_SIZE	TAC는 버퍼 캐시 이외에도 사용하는 공유 메모리가 많다. 따라서 버퍼 캐시의 크기를 싱글 인스턴스의 경우보다 더 작게 설정해야 한다. 일반적으로 전체 공유 메모리 크기의 절반 정도가 적절하다.
CLUSTER_DATABASE	TAC를 사용할 때 설정한다. 초기화 파라미터의 값은 반드시 **Y**로 설정해야 한다.
THREAD	Redo 스레드의 번호로, 각 인스턴스마다 고유의 번호를 부여한다.
UNDO_TABLESPACE	Undo 테이블스페이스의 이름으로, 각 인스턴스마다 고유하게 부여한다.
LOCAL_CLUSTER_ADDR	TAC 인스턴스끼리 통신할 내부 IP 주소를 설정한다.
LOCAL_CLUSTER_PORT	TAC 인스턴스끼리 통신할 내부 포트 번호를 설정한다.
TBCM_CLUSTER_MODE	초기화 파라미터의 값은 반드시 **ACTIVE_SHARED**로 설정해야 한다. TBCM과 관련된 설명은 “제9장 Tibero Cluster Manager”를 참고한다.

다음은 TAC를 설정할 때 주의해야 할 사항이다.

- Redo 스레드의 번호와 Undo 테이블스페이스의 이름은 동일한 데이터베이스를 서비스하는 서버 사이에서 유일해야 한다.“10.5. TAC를 위한 데이터베이스 생성”에서 생성한 이름이어야 한다.

- 모든 서버의 인스턴스의 설정 파일에 **CONTROL_FILES**와 **DB_CREATE_FILE_DEST**는 물리적으로 같은 파일이거나 또는 같은 디바이스를 가리키도록 설정해야 한다.

10.5. TAC를 위한 데이터베이스 생성

TAC는 공유 디스크 기반의 클러스터 데이터베이스이다. 여러 데이터베이스 서버의 인스턴스가 물리적으로 같은 데이터베이스 파일을 보고 사용하기 때문에 데이터베이스 생성은 한 서버에서 한번만 수행하면 된다.

모든 서버의 인스턴스가 동일한 컨트롤 파일 및 데이터 파일을 읽고 쓰게 된다. 반면 TAC에서는 공유 디스크에서 데이터 접근의 경합을 최소화하기 위해 Redo 로그 및 Undo에 대해서는 인스턴스마다 별도의 파일을 가지고 있어야 한다. Redo 로그 및 Undo 정보는 각 서버의 인스턴스들이 별도의 파일에 저장하지만, 복구 상황 등에 따라 다른 인스턴스의 정보를 읽어야 하므로 반드시 공유 디스크상에 존재해야 한다.

TAC를 위한 데이터베이스 생성 절차는 다음과 같다.

1. Tibero RDBMS와 관련된 환경설정 파일을 설정한 후, TBCM을 기동한다.

```
$ tbcm -c               ... ① ...
$ tbcm -b LOCK          ... ② ...
```

① TBCM 컨트롤 파일을 초기화하는 명령으로, y를 입력하면 초기화된다.

② TBCM을 기동했지만, Tibero RDBMS를 직접 제어하지는 않는다.

2. NOMOUNT 모드로 Tibero RDBMS를 기동한다.

```
$ tbboot -t NOMOUNT -c
```

3. SYS 사용자로 접속한 후 CREATE DATABASE 문을 통해 데이터베이스를 생성한다.

```
[tibero@tester ~]$ tbsql sys/tibero

SQL> CREATE DATABASE "TAC"                ... ① ...
     USER sys IDENTIFIED BY tibero
     MAXINSTACE 8                         ... ② ...
     MAXDATAFILES 256
     CHARACTER SET MSWIN949
     LOGFILE GROUP 0 'log001' SIZE 50M,
     GROUP 1 'log011' SIZE 50M,
     GROUP 2 'log021' SIZE 50M
     MAXLOGFILES 100
     MAXLOGMEMBERS 8
     NOARCHIVELOG
     DATAFILE 'system001' SIZE 512M
     AUTOEXTEND ON NEXT 8M MAXSIZE 3G
     DEFAULT TEMPORARY TABLESPACE TEMP
     TEMPFILE 'temp001' SIZE 512M
     AUTOEXTEND ON NEXT 8M MAXSIZE 3G
     EXTENT MANAGEMENT LOCAL AUTOALLOCATE
     UNDO TABLESPACE UNDO0
```

```
     DATAFILE 'undo001' SIZE 512M
     AUTOEXTEND ON NEXT 8M MAXSIZE 3G
     EXTENT MANAGEMENT LOCAL AUTOALLOCATE

Database created.
```

① DB_NAME으로 **TAC**를 지정한다.

② 접근할 서버의 최대 인스턴스의 개수를 8로 지정한다.

기존의 CREATE DATABASE 문과 비교해 달라진 부분은 없다. 다만, 데이터베이스파일을 공유할 인스턴스의 최대 개수를 나타내는 MAXINSTANCE 파라미터를 주의해야 한다. 이 파라미터의 값은 컨트롤 파일과 데이터 파일의 헤더 등에 영향을 미치며, 설정된 값 이상으로 Tibero RDBMS의 인스턴스를 추가할 수 없으므로, TAC를 위해 데이터베이스를 생성할 때, 충분한 값을 설정해야 한다.

앞서 설명한 것처럼 각 서버의 인스턴스는 별도의 Redo 및 Undo 공간을 가져야 한다. CREATE DATABASE 문을 실행한 시점에 생성된 Undo 테이블스페이스 및 Redo 로그 파일은 첫 번째 인스턴스를 위한 것으로 다른 인스턴스가 데이터베이스에 접근하기 위해서는 별도의 Redo 로그 그룹과 Undo 테이블스페이스를 생성하는 것이 필요하다.

생성된 Redo 로그 그룹은 자동으로 0번의 Redo 스레드가 된다. 따라서 CREATE DATABASE 문을 실행하기 전에 서버 인스턴스의 환경설정 파일($TB_SID.tip)의 THREAD 초기화 파라미터와 UNDO_TABLESPACE 초기화 파라미터는 다음과 같이 설정되어 있어야 한다.

```
THREAD=0
UNDO_TABLESPACE=UNDO0
```

4. CREATE DATABASE 문을 실행하고 나면 Tibero RDBMS가 자동으로 종료된다. Tibero RDBMS를 다시 기동한 후, SYS 사용자로 접속하여 다음과 같이 Undo 테이블스페이스와 새로운 Redo 로그 그룹을 만들고 DDL 문장을 수행한다.

```
[tibero@tester ~]$ tbboot
[tibero@tester ~]$ tbsql sys/tibero

SQL> CREATE UNDO TABLESPACE UNDO1
     DATAFILE 'undo011' SIZE 512M
     AUTOEXTEND ON NEXT 8M MAXSIZE 3G
     EXTENT MANAGEMENT LOCAL AUTOALLOCATE

Tablespace 'UNDO1' created.
```

```
SQL> ALTER DATABASE ADD LOGFILE THREAD 1 GROUP 3 'log031' size 50M;
Database altered.

SQL> ALTER DATABASE ADD LOGFILE THREAD 1 GROUP 4 'log041' size 50M;
Database altered.

SQL> ALTER DATABASE ADD LOGFILE THREAD 1 GROUP 5 'log051' size 50M;
```

```
Database altered.

SQL> ALTER DATABASE ENABLE PUBLIC THREAD 1;     ... ① ...
Database altered.
```

① Redo 스레드를 활성화하는 DDL 문장을 실행한다.

Redo 로그 그룹을 추가하기 위한 기존의 DDL 문장과 같지만 THREAD 번호를 지정했다는 것에 주의해야 한다. 이 예제에서는 두 번째 인스턴스가 사용할 Undo 테이블스페이스 **UNDO1**과 Redo 스레드 1을 위한 Redo 로그 그룹을 추가하고, 활성화시키는 과정을 보여주고 있다.

Redo 스레드는 숫자로 지정하며, CREATE DATABASE 문을 실행할 시점에 생성한 Redo 로그 그룹이 0번 스레드가 되므로 반드시 1부터 지정해야 한다. 0번 스레드는 CREATE DATABASE 문을 실행할 시점에 자동으로 활성화된다.

주의할 점은 Redo 로그 그룹의 번호는 Redo 스레드 내에서가 아니라 데이터베이스 전체에서 유일해야 하므로 이미 사용된 0,1,2를 사용할 수 없다. 또한, 최소한 두 개 이상의 Redo 로그 그룹이 존재해야만 해당 Redo 스레드를 활성화시킬 수 있다.

또 다른 인스턴스를 추가하고 싶다면, 위와 같은 과정을 참고하여 Undo 테이블스페이스와 Redo 스레드를 생성하고, 스레드를 활성화하면 된다.

5. $TB_HOME/scripts 디렉터리에 있는 system.sh 스크립트 파일을 실행한다.

   Windows 환경에서는 system.vbs 파일이다.

```
[tibero@tester scripts]$ system.sh $TB_HOME/bin/tbsvr
Creating the role DBA...
Creating system users & roles...
Creating virtual tables(1)...
Creating virtual tables(2)...
Granting public access to _VT_DUAL...
Creating the system generated sequences...
Creating system packages:
    Running /home/tibero/tibero4/scripts/pkg_standard.sql...
    Running /home/tibero/tibero4/scripts/pkg_dbms_output.sql...
    Running /home/tibero/tibero4/scripts/pkg_dbms_lob.sql...
    Running /home/tibero/tibero4/scripts/pkg_dbms_utility.sql...
    Running /home/tibero/tibero4/scripts/pkg_dbms_obfuscation.sql...
    Running /home/tibero/tibero4/scripts/pkg_dbms_transaction.sql...
    Running /home/tibero/tibero4/scripts/pkg_dbms_random.sql...
    Running /home/tibero/tibero4/scripts/pkg_dbms_lock.sql...
    Running /home/tibero/tibero4/scripts/pkg_dbms_system.sql...
    Running /home/tibero/tibero4/scripts/pkg_dbms_job.sql...
    Running /home/tibero/tibero4/scripts/pkg_utl_raw.sql...
    Running /home/tibero/tibero4/scripts/pkg_utl_file.sql...
    Running /home/tibero/tibero4/scripts/pkg_tb_utility.sql...
```

```
Creating public synonyms for system packages...
                          ........................................
```

이제 다른 서버의 인스턴스를 기동하고, 운영한다.

10.6. TAC 실행

10.6.1. 실행 전 준비 사항

TAC는 모든 인스턴스가 같이 사용할 수 있는 공유 디스크의 공간과 인스턴스 간의 통신이 가능한 네트워크만 있으면 실행할 수 있다.

TAC를 실행하기 전에 준비해야 할 H/W 요구사항과 운영체제 설정은 다음과 같다.

- 공유 디스크의 공간
  - TAC의 실행과 운영을 위해서는 최소 7개의 공유 파일이 필요하다.
    - 컨트롤 파일
    - Redo 로그 파일 2개
    - Undo 로그 파일
    - 시스템 테이블스페이스 파일
    - 임시 테이블스페이스 파일
    - TBCM 파일
  - 인스턴스 하나를 추가 할 때마다 최소 3개의 공유 파일이 추가로 필요하다.
    - Redo 로그 파일 2개
    - Undo 로그 파일
- 공유 디스크의 권한

  공유 파일 시스템을 사용할 경우에는 TAC가 사용할 디렉터리의 권한을, RAW 파일 시스템을 사용할 경우에는 각 RAW 파일의 권한을 TAC가 읽고 쓸 수 있도록 설정해야 한다.
- 내부 네트워크의 설정

TAC의 실행과 운영을 위해 내부 네트워크는 외부 네트워크와 분리하는 것이 데이터베이스 성능에 좋다. 내부 네트워크의 인터페이스와 IP, 속도 등을 점검한다.

10.6.2. 데이터베이스 생성

TAC를 처음 기동할 때에는 데이터베이스를 생성해 주어야 한다. 클러스터로 사용할 한 노드에서 생성하면 된다. TAC로 Tibero RDBMS를 기동할 때에는 TBCM이 필수이므로, TBCM를 먼저 초기화한 후 시작하고 Tibero RDBMS를 NOMOUNT 모드로 기동한다.

참고

TBCM를 초기화한 후, Tibero RDBMS를 처음 시작했을 때는 node membership을 관리하는 초기화 시간이 필요하다. Tibero RDBMS를 너무 빨리 기동하게 되면 TBCM은 node membership에 포함되지 않은 인스턴스로 간주하고 강제로 종료된다. 따라서 15초 이상의 시간으로 기동할 것을 권장한다.

다음은 TBCM를 초기화하고 Tibero RDBMS를 NOMOUNT 모드로 기동하는 예이다.

```
$tac1> tbcm -c
This will erase all contents in [/home/tibero4/instance/tbcm], and cannot be
recovered. Proceed? (y/N) y
SUCCESS
$tac1> tbcm -b

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero cluster manager started up.
Local node name is (tbcm@xxx.x.x.x:xxxx).

$tac1> tbboot -t NOMOUNT
listener port = xxxx
change core dump dir to /home/tibero4/bin/prof

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NOMOUNT mode).

$tac1>
```

NOMOUNT 모드로 기동된 첫 번째 TAC의 인스턴스에 접속하여 데이터베이스를 만든 후, 다른 클러스터의 인스턴스를 위한 Redo 로그, Undo 로그를 생성한다. 인스턴스의 $TB_SID.tip 환경설정 파일에 지정한 THREAD, UNDO_TABLESPACE 초기화 파라미터의 이름과 반드시 일치해야 한다.

10.6.3. TAC 기동

데이터베이스 생성과 다른 인스턴스를 위한 환경설정 파일의 생성이 모두 완료하면, TAC를 기동할 수 있다.

다음은 다른 인스턴스를 모두 실행하고 TBCM, Tibero RDBMS 순으로 TAC를 기동하는 예이다.

```
$tac2> tbcm -b

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero cluster manager started up.
Local node name is (tbcm@xxx.x.x.x:xxxx).

$tac2> tbboot
listener port = xxxx
change core dump dir to /home/tibero4/bin/prof

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

Tibero instance started up (NORMAL mode).

$tac2>
```

10.6.4. TAC 모니터링

$TB_HOME/scripts 디렉터리에 있는 tbcm_stat.sh 스크립트 파일을 통해 노드 및 인스턴스의 상태를 모니터링할 수 있다. TAC에 참여한 노드의 현재 상태, IP 정보 등을 주기적으로 확인할 수 있다.

Tibero RDBMS는 글로벌 뷰(Global View)를 제공한다. 싱글 인스턴스에서 사용하는 모든 모니터링 뷰를 TAC의 인스턴스에서 조회할 수 있다.

다음은 모든 클러스터에 연결되어 있는 세션을 확인하는 예이다. tbSQL 유틸리티, tbAdmin 툴을 통해 다음과 같은 SQL 문장을 실행한다.

[예 10.1] 글로벌 뷰의 조회 - GV$SESSION

```
SQL> SELECT * FROM GV$SESSION;
```

10.7. 로드 밸런싱

사용자의 프로그램은 TAC를 기본적으로 하나의 DBMS처럼 다룰 수 있다. 따라서 사용자는 하나의 DBMS에 접속한 것처럼 연결을 맺고 작업을 수행하면 된다.

장애가 발생했을 때 Failover 및 로드 밸런싱 기능은 클라이언트 라이브러리와 TAC에서 자동으로 처리된다. 이를 위해 클라이언트는 여러 대의 이중화 서버에 접속하는 것과 같이 Failover와 로드 밸런싱 기능을 설정한다.

참고

이중화 서버를 위한 클라이언트 설정은 “A.2. 이중화 서버의 설정”를, 로드 밸런싱 설정은 “A.3. 로드 밸런싱의 설정”를, Failover 설정은 “A.4. Failover의 설정”를 참고한다.

클라이언트의 LOAD_BALANCE, USE_FAILOVER 설정만으로도 기본적인 로드 밸런싱과 Failover 기능이 사용자에게 제공된다. 클라이언트는 TAC를 구성하는 여러 노드 중 임의의 노드를 선택하여 접속하며, 해당 노드에 장애가 발생했을 때 다른 노드로 연결이 자동 전환된다. 이렇게 장애가 발생하면 수행 중이던 SQL 문장에서 에러가 나며, 다음 SQL부터는 새로운 노드에서 작업이 수행된다. **발생하는 SQL은 장애 유형에 따라 다르다.**

이렇게 클라이언트의 설정만으로 제공되는 로드 밸런싱 기능은 별다른 부하를 발생시키지 않는다. 단, 서버의 부하 상황을 고려하지 않으므로, 정확한 의미의 로드 밸런싱은 아니다. 이것은 한꺼번에 서버와의 연결을 많이 맺고 오래 사용하는 목적에 알맞으며 3-tier(multi-tier) 방식에 유용하다. 또한, 서버와의 연결과 끊김이 잦은 2-tier 방식에서도 TAC 서버에 로드 밸런싱 기능을 추가하여 서버의 부하 상황을 고려하는 것이 좋다.

다음은 서버 쪽에 로드 밸런싱을 설정하는 가장 기본적인 예이다.

[예 10.2] 서버 쪽 로드 밸런싱 설정

<$TB_SID.tip>

```
SERVER_SIDE_LOAD_BALANCE=LONG
LOAD_METRIC_AUTO_INTERVAL=Y
SERVER_LOAD_TOLERANCE=30
```

초기화 파라미터	설명
SERVER_SIDE_LOAD_BALANCE	필수 항목으로 서버 쪽에 로드 밸런싱을 하는 방법을 설정하는 파라미터이다. - NONE: 디폴트 값으로, 서버 쪽에서는 로드 밸런싱을 하지 않는다는 의미이다.

초기화 파라미터	설명
	- SHORT: SQL 문장을 수행할 때마다 서버의 연결 환경에 맞게 평균 응답 시간을 기준으로 로드 밸런싱을 한다는 의미이다. - LONG: 서버 쪽에 로드 밸런싱이 필요하지만, 비교적 서버의 연결 유지 시간이 긴 상황에 대비하기 위해 각 서버의 연결 개수를 기준으로 로드 밸런싱을 한다는 의미이다.
LOAD_METRIC_AUTO_INTERVAL	옵션 항목으로 서버의 부하 상황을 수집하는 것을 자동으로 할지의 여부를 결정하는 파라미터이다. 이 파라미터는 부하의 변화가 많은 시간에는 자주 수집하고 반면에 부하의 변화가 거의 없는 시간에는 천천히 수집하도록 수집 주기가 자동으로 조절된다. 서버 쪽에 로드 밸런싱이 시작되면 각각의 TAC 서버는 주기적으로 부하 상황을 주고 받으며 이를 수집한다. 수집 주기는 다음의 설정에 따라 다르다. - 수집 주기는 기본적으로 LOAD_METRIC_COLLECT_INTERVAL_MAX 초기화 파라미터에 설정된 값의 한번이다. - **자동 주기 조절 상태** (LOAD_METRIC_AUTO_INTERVAL=Y로 설정된 경우)라면, LOAD_METRIC_COLLECT_INTERVAL_MIN ~ LOAD_METRIC_COLLECT_INTERVAL_MAX 초기화 파라미터에 설정된 값 사이에서 수집 주기가 자동으로 결정된다.
LOAD_METRIC_COLLECT_INTERVAL_MIN	옵션 항목으로 자동 주기 조절 상태일 때, 자주 수집할 때의 시간 간격을 설정하는 파라미터이다. (기본값 : 1초)
LOAD_METRIC_COLLECT_INTERVAL_MAX	옵션 항목으로 자동 주기 조절 상태일 때, 천천히 수집하도록 시간 간격을 설정하는 파라미터이다. (기본값: 10초)
LOAD_METRIC_HISTORY_COUNT	옵션 항목으로 기록할 수집된 부하 정보의 개수를 설정하는 파라미터이다. 수집된 부하 정보는 뷰(V$INSTANCEMETRIC, V$INSTANCEMETRIC_HISTORY)를 통해 확인할 수 있다.
SERVER_LOAD_TOLERANCE	옵션 항목으로 서버 간의 부하 상황에 대해서 어느 정도의 차이를 허용할 것인가를 설정하는 파라미터이다. 약간의 차이도 허용하지 않으면 클라이언트의 연결이 한쪽으로 치우치는 현상이 발생할 수 있으므로, 이 파라미터를 설정한다. 다음은 SERVER_SIDE_LOAD_BALANCE 초기화 파라미터에 설정된 값에 따라 달라지는 SERVER_LOAD_TOLERANCE 초기화 파라미터의 의미이다. - SERVER_SIDE_LOAD_BALANCE 초기화 파라미터의 값이 LONG인 경우: 각 서버가 담당하는 클라이언트의 연결 개수의 차이를 의미한다.

초기화 파라미터	설명
	- SERVER_SIDE_LOAD_BALANCE 초기화 파라미터의 값이 SHORT인 경우: 각 서버의 SQL 문장에 대한 평균 응답 시간의 차이를 의미하며, 단위는 msec로 나타낸다.

참고

서버 쪽의 로드 밸런싱은 새로 연결을 시도하는 클라이언트에 대해서만 동작한다. 너무 많은 부하가 걸리거나 너무 적게 걸리더라도 이미 맺은 연결은 강제로 끊지 않는다.

제11장 데이터 암호화

본 장에서는 Tibero RDBMS에서 데이터 암호화(Data Encryption) 기능을 사용하고 관리하기 위한 방법을 설명한다.

11.1. 개요

Tibero RDBMS에서는 데이터 보안을 위해 “제5장 사용자 관리와 데이터베이스 보안”에서 설명한 것처럼 사용자 계정 및 특권, 역할 기능을 제공하고 있다. 하지만, 데이터베이스 내에서 데이터에 접근하는 경우가 아니라 운영체제에서 데이터 파일에 직접 접근하는 경우라면 위의 기능만으로는 데이터를 안전하게 보호할 수가 없다. 따라서 이러한 경우에도 데이터를 보호하기 위해서 Tibero RDBMS는 데이터를 암호화하여 디스크에 저장하는 기능을 제공하고 있다.

DBA가 암호화할 데이터(테이블의 컬럼 또는 테이블스페이스)를 지정하면, Tibero RDBMS는 데이터를 저장할 때 내부적으로 암호화하여 저장하고 검색할 때 복호화해서 보여준다. 이때 사용자나 애플리케이션 프로그램은 데이터의 암호화 여부를 고려할 필요가 없다.

Tibero RDBMS는 다음과 같은 암호화 알고리즘을 제공한다.

키워드	설명
DES	DES 64 bits key
3DES168	3 Key Triple DES 168 bits key
AES128	AES 128 bits key
AES192	AES 192 bits key
AES256	AES 256 bits key

11.2. 환경설정

데이터 암호화 기능을 사용하기 위해서는 우선 먼저 보안 지갑을 생성해야 한다. 보안 지갑은 암호화에 사용할 마스터 키를 보관하고 있다. Tibero RDBMS는 마스터 키를 이용하여 각 데이터를 암호화할 암호화 키를 생성하고, 이 암호화 키를 이용하여 데이터를 암호화한다.

데이터 암호화를 사용하기 위해서는 다음의 절차를 수행해야 한다.

1. 보안 지갑의 생성

   $TB_HOME/bin/tbwallet_gen 프로그램을 실행하고 보안 지갑의 파일 이름과 패스워드를 입력하여 보안 지갑을 생성한다.

[예 11.1] 보안 지갑의 생성

```
$ tbwallet_gen

[ Tibero Security Wallet Generator ]
Enter wallet file name: TBWALLET
Enter wallet password: tibero
generate wallet success
```

2. 보안 지갑의 위치 설정

   $TB_SID.tip 파일에 WALLET_FILE 초기화 파라미터를 추가하여 보안 지갑의 위치를 설정한다.

   [예 11.2] 보안 지갑의 위치 설정

   <$TB_SID.tip>

```
WALLET_FILE=/path/to/TBWALLET
```

3. 보안 지갑의 사용

   생성한 보안 지갑을 열고 데이터 암호화 기능을 사용하려면 DBA 권한을 가진 사용자가 다음의 명령을 수행해야 한다. 단, IDENTIFIED BY 절 뒤에 입력하는 패스워드는 보안 지갑을 생성할 때 입력했던 패스워드를 입력한다.

   [예 11.3] 보안 지갑 열기

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "tibero";
System altered.
```

   데이터 암호화 기능을 사용하지 못하도록 보안 지갑을 닫으려면 이 역시 DBA 권한을 가진 사용자가 다음의 명령을 수행해야 한다.

   [예 11.4] 보안 지갑 닫기

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;
System altered.
```

주의

보안 지갑은 데이터베이스 인스턴스를 종료하면 닫히므로, 다시 기동할 때마다 보안 지갑을 열어야 한다.

11.3. 컬럼 암호화

컬럼 암호화(Column Encryption)는 테이블의 특정 컬럼의 데이터를 암호화하여 저장하는 기능이다. 컬럼 암호화는 테이블을 생성 또는 변경하는 DDL 문장을 수행할 때 설정한다.

컬럼 암호화를 설정할 때에는 암호화 알고리즘과 SALT 옵션을 사용할 것인지의 여부를 명시할 수 있다.

암호화 알고리즘은 Tibero RDBMS에서 지원하는 범위에서만 설정할 수 있으며, 설정하지 않으면 AES192 알고리즘을 디폴트로 사용한다. 암호화된 컬럼은 한 테이블에 여러 개가 있을 수 있지만, 한 테이블에는 하나의 암호화 알고리즘만 사용할 수 있다.

SALT 옵션은 같은 값의 데이터를 암호화할 때 항상 같은 암호로 암호화되지 않도록 하는 기능이다. 이 옵션을 사용할 경우 보안의 수준을 높일 수 있다. 하지만, SALT 옵션을 사용하여 암호화한 컬럼에는 인덱스를 생성할 수 없다. 컬럼 암호화를 설정할 때 이 옵션을 별도로 설정하지 않아도 디폴트는 SALT 옵션을 사용한다.

컬럼을 암호화하면 보안의 수준이 높아지는 장점이 있지만, SALT 옵션을 사용하지 않는 컬럼에 한해서만 인덱스를 생성할 수 있고, 인덱스 영역도 스캔할 수 없다는 단점이 있다. 또한, 해당 컬럼에 DML 및 SELECT 명령을 실행하면 내부에서 암호화와 복호화를 수행하기 때문에 성능 저하가 발생한다.

암호화할 수 있는 컬럼의 데이터 타입은 다음과 같다.

- CHAR
- VARCHAR2
- NUMBER
- DATE
- TIMESTAMP
- INTERVAL DAY TO SECOND
- INTERVAL YEAR TO MONTH
- RAW
- NCHAR
- NVARCHAR2

11.3.1. 암호화 컬럼을 갖는 테이블 생성

CREATE TABLE 문에서 암호화할 컬럼을 정의할 때 ENCRYPT 절을 지정하여 테이블을 생성한다.

[예 11.5] 암호화 컬럼을 갖는 테이블 생성 - 디폴트 암호화 옵션 (AES192 알고리즘, SALT)

```
SQL> CREATE TABLE customer (
       cust_id       CHAR(4) NOT NULL CONSTRAINT cust_id_pk PRIMARY KEY,
       cust_name     VARCHAR(20) NULL,
       cust_type     VARCHAR(18) NULL,
       cust_addr     VARCHAR(40) NULL,
       cust_tel      VARCHAR(15) NULL ENCRYPT,
       reg_date      DATE NULL
);
Table 'CUSTOMER' created.
```

암호화 알고리즘은 USING 절을 사용하여 지정할 수 있으며, SALT 옵션의 사용 여부도 지정할 수 있다. 특히 암호화 알고리즘은 한 테이블에 하나만 지정할 수 있기 때문에 다른 컬럼에 또 다른 알고리즘을 지정하면 에러가 발생한다.

[예 11.6] 암호화 컬럼을 갖는 테이블 생성 - AES256 알고리즘, NO SALT 옵션 설정

```
SQL> CREATE TABLE customer (
       cust_id       CHAR(4) NOT NULL CONSTRAINT cust_id_pk PRIMARY KEY,
       cust_name     VARCHAR(20) NULL,
       cust_type     VARCHAR(18) NULL,
       cust_addr     VARCHAR(40) NULL ENCRYPT USING 'AES256',
       cust_tel      VARCHAR(15) NULL ENCRYPT NO SALT,
       reg_date      DATE NULL
);
Table 'CUSTOMER' created.
```

11.3.2. 테이블에 암호화 컬럼 추가

ALTER TABLE 문에서 컬럼을 추가할 때 ENCRYPT 절을 지정하면 암호화 컬럼이 된다. 기존 테이블에 이미 암호화 컬럼이 있으면 기존 컬럼과 동일한 암호화 알고리즘을 사용한다. 반면에 암호화 컬럼이 없으면 새로운 알고리즘을 지정할 수 있다. 또한, SALT 옵션의 사용 여부도 지정할 수 있다.

[예 11.7] 암호화 컬럼 추가

```
SQL> ALTER TABLE customer ADD (cust_password VARCHAR(12) ENCRYPT NO SALT);
Table 'CUSTOMER' altered.
```

11.3.3. 일반 컬럼을 암호화 컬럼으로 변경

ALTER TABLE 문에서 컬럼을 변경할 때 ENCRYPT 절을 지정하면 암호화 컬럼이 된다. 테이블에 암호화 컬럼을 추가할 경우와 마찬가지로 기존 테이블에 이미 암호화 컬럼이 있으면 기존 컬럼과 동일한 암호화 알고리즘을 사용한다. 반면에 암호화 컬럼이 없으면 새로운 알고리즘을 지정할 수 있다. 또한, SALT 옵션의 사용 여부도 지정할 수 있다.

[예 11.8] 일반 컬럼을 암호화 컬럼으로 변경

```
SQL> ALTER TABLE customer MODIFY (reg_date ENCRYPT NO SALT);
Table 'CUSTOMER' altered.
```

11.3.4. 암호화 컬럼을 일반 컬럼으로 변경

ALTER TABLE 문에서 컬럼을 변경할 때 DECRYPT 절을 지정하면 일반 컬럼이 된다.

[예 11.9] 암호화 컬럼을 일반 컬럼으로 변경

```
SQL> ALTER TABLE customer MODIFY (reg_date DECRYPT);
Table 'CUSTOMER' altered.
```

11.3.5. 모든 암호화 컬럼의 알고리즘 변경

ALTER TABLE 문에서 REKEY 명령을 지정하면 해당 테이블의 모든 암호화 컬럼의 암호화 알고리즘이 변경된다.

[예 11.10] 모든 암호화 컬럼의 암호화 알고리즘 변경

```
SQL> ALTER TABLE customer REKEY USING '3DES168';
Table 'CUSTOMER' altered.
```

11.4. 테이블스페이스 암호화

테이블스페이스 암호화(Tablespace Encryption)는 컬럼 암호화와 마찬가지로 데이터베이스의 데이터를 보호하는 또 다른 방법이다. 컬럼 암호화와 다른 점은 테이블 혹은 테이블의 컬럼 단위가 아닌 테이블스페이스 전체에 대해 암호화 여부와 암호화 알고리즘을 지정한다는 것이다.

테이블스페이스 암호화와 복호화 과정은 디스크에서 읽기와 쓰기를 한 후에 바로 수행되며, 데이터베이스 내부의 SQL 처리 또한 기존과 동일하게 진행된다. 이 과정 때문에 컬럼 암호화에 존재했던 제약 즉 일부 타입의 컬럼에 대해서만 컬럼 암호화를 할 수 있다거나 인덱스 영역을 스캔할 수 없다는 문제는 없어진다. 반면에 테이블스페이스의 모든 데이터 블록에 암호화와 복호화가 발생하기 때문에 테이블스페이스의 크기가 크거나 수정과 접근이 잦을수록 성능 저하가 높아질 수 있다. 또한, 컬럼 암호화처럼 데이터 암호화 여부를 바꿀 수 없다는 문제도 있으므로 보호할 데이터의 성격에 따라 적절한 방법을 선택해야 한다.

11.4.1. 암호화된 테이블스페이스의 생성

테이블스페이스를 생성하는 **CREATE TABLESPACE** 문에서 암호화 여부를 지정하는 ENCRYPT 절을 추가하면 그 테이블스페이스는 암호화된 테이블스페이스가 된다. 또한, USING를 사용하여 암호화 알고리즘도 지정할 수 있다. 여기서 암호화 알고리즘은 3DES168, AES128, AES192, AES256 중에서 하나만 지정할 수 있다. USING에서 '암호화 알고리즘'을 생략하면 AES128을 기본으로 사용한다.

[예 11.11] 암호화된 테이블스페이스 생성 - 3DES168 알고리즘 지정

```
SQL> CREATE TABLESPACE encrypted_space
     DATAFILE '/usr/tibero/data/encrypted001.tdf' SIZE 50M
     AUTOEXTEND ON NEXT 1M
     EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K
     ENCRYPTION USING '3DES168';
Tablespace 'ENCRYPTED_SPACE' created.
```

암호화된 테이블스페이스를 생성할 때에는 다음과 같은 사항을 주의해야 한다.

- 암호화 알고리즘 지정

  테이블스페이스 암호화에 사용할 수 있는 암호화 알고리즘의 종류가 컬럼 암호화에 비해 적다. 따라서 3DES168, AES128, AES192, AES256 이외의 암호화 알고리즘을 지정하게 되면 암호화된 테이블스페이스는 생성할 수 없다.

- 보안 지갑 열기

  암호화된 테이블스페이스를 생성하기 전에 반드시 보안 지갑이 열려 있어야 한다. 보안 지갑이 열리지 않은 상태에서 CREATE TABLESPACE 문을 실행하게 되면 다음의 예처럼 에러가 발생하게 되고, 테이블스페이스와 데이터 파일이 생성되지 않게 된다. 따라서 이를 해결하기 위해서는 보안 지갑을 열고, 다시 시도하면 된다.

  [예 11.12] 암호화된 테이블스페이스 - 생성 실패

  ```
  SQL> CREATE TABLESPACE encrypted_space
       DATAFILE '/usr/tibero/data/encrypted001.tdf' SIZE 50M
       AUTOEXTEND ON ENCRYPT;
  TBR-12073: wallet not opened.
  ```

11.4.2. 암호화된 테이블스페이스의 변경

암호화된 테이블스페이스의 저장 공간이 더 필요한 경우 **ALTER TABLESPACE** 문의 **ADD DATAFILE** 절을 이용하여 새로운 데이터 파일을 테이블스페이스에 추가할 수 있다. 새로 추가된 데이터 파일은 처음 테이블스페이스를 생성했을 때 지정한 암호화 여부와 암호화 알고리즘의 속성을 그대로 가지게 된다.

[예 11.13] 암호화된 테이블스페이스 - 데이터 파일 추가

```
SQL> ALTER TABLESPACE encrypted_space
     ADD DATAFILE '/usr/tibero/data/encrypted002.tdf' SIZE 50M;
Tablespace 'ENCRYPTED_SPACE' altered.
```

일반 테이블스페이스를 암호화된 테이블스페이스로 바꾸거나 암호화된 테이블스페이스를 일반 테이블스페이스로 바꾸는 것은 불가능하다. 또한, 한 테이블스페이스에 포함된 데이터 파일에 대해서도 암호화 여부와 암호와 알고리즘을 다르게 지정하는 것도 불가능하다. 따라서 테이블스페이스를 생성할 때에는 이러한 사항을 고려하고 신중히 결정해야 한다.

이러한 사항에도 일반 테이블스페이스를 암호화된 테이블스페이스로 바꾸기 위해서는 새로 암호화된 테이블스페이스를 만든 다음 일반 테이블스페이스에 포함된 데이터를 모두 암호화된 테이블스페이스로 옮겨야 한다.

11.4.3. 암호화된 테이블스페이스의 사용

테이블 혹은 인덱스를 생성할 때, 암호화된 테이블스페이스를 지정하면 해당 세그먼트는 물리적으로 그 테이블스페이스에 저장되고, 데이터 블록을 읽고 쓰는 과정에서 자동으로 데이터 암호화와 복호화가 발생한다.

[예 11.14] 암호화된 테이블스페이스 - 테이블 생성

```
SQL> CREATE TABLE customer (
       cust_id       CHAR(4) NOT NULL CONSTRAINT cust_id_pk PRIMARY KEY,
       cust_name     VARCHAR(20) NULL,
       cust_type     VARCHAR(18) NULL,
       cust_addr     VARCHAR(40) NULL,
       cust_tel      VARCHAR(15) NULL,
       reg_date      DATE NULL
     ) TABLESPACE secure_space;
Table 'CUSTOMER' created.
```

암호화된 테이블스페이스에 테이블을 생성하고, SQL 문장을 통해 데이터를 조회하고 갱신하는 과정 동안에는 보안 지갑은 항상 열려 있어야 한다. 보안 지갑이 열리지 않은 상태에서 암호화된 테이블스페이스가 포함된 동작을 수행하게 되면 [예 11.12]와 같은 에러가 발생한다.

테이블스페이스 중에서 SYSTEM, UNDO, TEMP 테이블스페이스는 암호화 여부를 지정할 수 없다. 즉 모든 테이블스페이스를 암호화할 수 있는 것은 아니다. 또한, Redo 로그 파일에 대해서도 암호화 여부를 지정할 수 없다. 하지만, 암호화된 테이블스페이스에 포함된 데이터를 수정하는 과정에서 생성된 UNDO, Redo 로그는 자동으로 암호화되어 디스크에 저장되고, 이 두 로그가 필요한 순간에는 자동으로 복호화되어 데이터베이스 내부적으로 사용된다.

그리고 정렬을 수행하는 과정에서 TEMP 테이블스페이스에 임시로 저장되는 데이터가 암호화된 테이블스페이스에 속한 것이라면 TEMP 영역도 자동으로 암호화하고 복호화된다. 따라서 데이터베이스의 다른

파일(UNDO, Redo, TEMP 등)을 통해 암호화된 테이블스페이스의 데이터가 일부라도 노출되지 않도록 보호할 수 있다.

11.4.4. 암호화된 테이블스페이스의 정보 조회

Tibero RDBMS에서는 암호화된 테이블스페이스의 정보를 관리하기 위해 다음 표에 나열된 뷰를 제공하고 있다.

뷰	설명
DBA_TABLESPACES	테이블스페이스의 전체 정보를 조회하는 뷰이다. ENCRYPTED 컬럼을 통해 암호화 여부를 조회할 수 있다.
V$ENCRYPTED_TABLESPACES	암호화된 테이블스페이스의 정보만을 조회하는 뷰이다. 테이블스페이스 ID와 암호화 알고리즘을 조회할 수 있다.

참고

정적 뷰와 동적 뷰에 대한 자세한 내용은 "Tibero RDBMS 참조 안내서"를 참고한다.

제12장 통신 암호화

본 장에서는 Tibero RDBMS에서 통신 암호화 기능을 사용하고 관리하기 위한 방법을 설명한다.

12.1. 개요

Tibero RDBMS는 서버와 클라이언트 간에 송수신되는 메시지에 대한 기밀성을 보장하기 위하여, 통신 암호화 기능을 제공한다. Tibero RDBMS의 통신 암호화 기능은 Netscape사의 SSL 통신 프로토콜을 채택하였으며 Openssl Project에서 제공하는 라이브러리를 사용하여 개발되었다.

12.2. 환경설정

통신 암호화 기능을 사용하기 위해서는, 통신 양자 간에 보안 통신을 위한 설정이 필요하다. 서버에서는 보안 통신에 사용될 인증서와 개인키를 미리 소지하고 있거나, 없으면 새로 생성해야한다. 그 다음 인증서와 개인키의 위치를 환경 설정 파일에 지정해주게 되면, 보안 통신 전용 포트가 열리게 된다.

클라이언트에서는 보안 통신 사용 여부를 환경 설정 파일에 지정해준다. 이에 따라 일반 혹은 보안 접속이 이루어지게 된다.

12.2.1. 개인키 및 인증서 생성

개인키 및 인증서 생성은 X.509 v3(PKI ITU-T 표준)을 따른다. 개인키 및 인증서를 생성하는 방법은 다음과 같다.

[예 12.1] 개인키 및 인증서의 생성

```
$ ./tb_cert_manager

Enter new password: tibero
Repeat: tibero
Enter basename to make a certificate and a private key.(default: $TB_SID): (enter)
TB
Enter the number of days to make a certificate valid for.(default: 3650): (enter)
3650

=== STEP 1. Generate private key ===
Generating RSA private key, 1024 bit long modulus
....++++++
..........++++++
e is 65537 (0x10001)
```

```
=== STEP 2. Generate CSR(Certificate Signing Request) ===
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: 82
State or Province Name (full name) [Some-State]: Kyounggi
Locality Name (eg, city) []: Seongnam
Organization Name (eg, company) [Internet Widgits Pty Ltd]: TIBERO
Organizational Unit Name (eg, section) []: RnD
Common Name (eg, YOUR name) []: Hong Gil Dong
Email Address []: gdhong@tibero.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: (enter)
An optional company name []: (enter)

=== STEP 3. Generate certificate ===
```

입력을 마치면 전자지갑 저장소($TB_HOME/config/tb_wallet/)에 다음과 같은 파일이 생성된다.

```
공개키 기반 개인키: [$TB_SID].key
인증서 생성 요청서: [$TB_SID].csr
인증서: [$TB_SID].crt
```

12.2.2. 개인키 및 인증서 위치 설정

$TB_SID.tip 환경설정 파일에 CERTIFICATE_FILE과 PRIVKEY_FILE 초기화 파라미터를 추가하여 인증서와 개인키의 위치를 설정한다.

[예 12.2] 개인키 및 인증서의 위치설정

<$TB_SID.tip>

```
CERTIFICATE_FILE=$TB_HOME/config/tb_wallet/[$TB_SID].crt
PRIVKEY_FILE=$TB_HOME/config/tb_wallet/[$TB_SID].key
```

12.2.3. 클라이언트 설정

tbdsn.tbr 파일에서 통신 암호화 사용을 설정하는 방법은 다음과 같다.

[예 12.3] 클라이언트 설정

<tbdsn.tbr>

```
TB=(
    (INSTANCE=(HOST=서버 IP)
              (PORT=서버 포트)
              (DB_NAME=데이터베이스 이름)
              (USE_SSL=Y)
    )
   )
```

SID 내의 USE_SSL값을 Y로 설정하면 통신 암호화 기능을 사용할 수 있다. 반면에 USE_SSL 초기화 파라미터가 없거나 Y 외의 다른 값을 입력하면 일반 접속이 된다.

PORT는 서버 포트(LISTENER_PORT)를 명시하면 되는데, 내부적으로 SSL 전용 포트(_LSNR_SSL_PORT = LISTENER_PORT + 2)가 추가 지정되게 된다. 일반 포트를 통한 통신으로 일단 SSL 소켓을 생성하기 위한 핸드쉐이킹을 수행하게 되고, SSL 소켓이 생성된 이후에는 내부적으로 지정된 SSL 전용 포트를 통해 보안통신을 하게된다.

제13장 Parallel Execution

본 장에서는 Parallel Execution의 기본개념과 동작원리를 소개하고, 이를 유형별로 실행하는 방법을 기술한다.

13.1. 개요

Parallel Execution(이하 PE)은 한 개의 작업을 여러 개로 나누어 동시 처리를 실행하는 방법으로, 데이터웨어하우스(DW: Data Warehouse)와 BI(Business Information)를 지원하는 대용량 DB에서 발생하는 데이터 처리의 응답 시간을 획기적으로 줄일 수 있다. 또한, OLTP (online transaction processing) 시스템에서도 배치 처리의 성능을 높일 수 있다.

PE는 시스템 리소스의 활용을 극대화하여 데이터베이스 성능을 향상시키고자 하는 것이 기본 사상이기 때문에, 다음과 같은 작업에 대해 성능 향상을 기대할 수 있다.

- 대용량 테이블 혹은 파티션 인덱스 스캔(partitioned index scan)이나 조인 등의 쿼리
- 대용량 테이블의 인덱스 생성
- Bulk insert나 update, delete
- Aggregations

PE가 시스템 리소스를 최대한 활용하는 방식인 만큼, 잘못 사용했을 때는 리소스 고갈로 중요한 데이터 처리가 지연되거나, 오히려 기대했던 성능이 나오지 못하는 경우가 발생할 수 있기 때문에 사용에 주의가 필요하다.

PE로 성능 향상을 기대할 수 있는 시스템 구성과 환경은 다음과 같다.

- CPU, 클러스터링 등 시스템 측면의 parallel 구성
- 충분한 입출력 대역폭
- 여유 있는 CPU 사용률

  예) CPU 사용량이 30% 이하일 때
- 정렬, 해싱 그리고 입출력 버퍼와 같은 작업을 수행하기 충분한 메모리

13.2. Degree of Parallelism

Degree of Parallelism(이하 DOP)이란 하나의 연산에 얼마나 많은 시스템 리소스를 할당하여 동시에 처리하도록 할 것인지를 결정하기 위해 사용되는 개념으로, 단순하게는 하나의 연산을 함께 수행하는 워킹 스레드(이하 WTHR)의 개수를 의미하기도 한다.

참고

연산이란 order by, full table scan과 같이 하나의 쿼리에서 해당 WTHR에 할당되는 규모의 작업 단위를 말한다.

PE는 하나의 연산을 여러 개의 WTHR로 동시에 수행하도록 하는 **intra-operation PE**와 서로 다른 연산을 파이프 스트림(pipe stream) 방식으로 연결해 동시에 수행하도록 하는 **inter-operation PE**로 나눌 수 있다. Tibero RDBMS에서는 DOP 크기만큼의 WTHR를 할당하여 intra-operation PE를 구성하고, inter-operation PE에 대해서는 2-set을 구성하는 방식으로 최적의 PE를 수행한다.

따라서 PE의 실행 과정을 통제하는 Query Coordinator(이하 QC)는 PE를 위해 최대 2*DOP 개수만큼의 WTHR를 PE_SLAVE(Parallel Execution Slave)로 활용하게 된다. 단, 동시에 두 개 이상의 연산을 수행하는 inter-operation PE는 지원하지 않는다.

13.2.1. DOP 결정

하나의 SQL 문장(쿼리)에서 하나의 DOP로 PE를 수행하며, 여러 개의 parallel hint가 있거나 혹은 parallel hint도 있고 parallel option을 가진 테이블을 쿼리에 포함하는 경우, 그 중 가장 큰 DOP를 그 쿼리의 DOP로 결정한다. 명시된 DOP가 0보다 작거나 같은 경우에는 기본값 4로 DOP를 결정한다. DOP는 parallel hint에 명시할 수 있으며, 사용 방법은 "Tibero RDBMS SQL 참조 안내서"를 참고한다.

DOP를 결정할 때 참고해야 할 요소는 다음과 같다.

- 시스템의 CPU 개수
- 시스템의 최대 프로세스 및 스레드 개수
- 테이블이 분산된 경우, 그 테이블이 속해 있는 디스크의 개수
- 데이터의 위치나 쿼리의 종류
- 객체 파티션 개수

보통 한 사용자만 PE를 수행한다고 하면, 정렬과 같은 CPU bound 작업은 CPU 개수의 1~2배로 DOP를 설정하는 것이 적당하고, 테이블 스캔과 같은 I/O bound 작업은 디스크 드라이브 개수의 1~2배로 DOP를

설정하는 것이 적당하다. 이처럼 쿼리의 종류와 시스템 환경을 고려하여 DOP를 설정하면 PE를 통해 더 나은 성능을 기대할 수 있다.

13.2.2. DOP에 따른 워킹 스레드 할당

Tibero RDBMS에서는 쿼리를 수행하는 과정에서 parallel 연산을 수행할 차례가 되면, QC가 계산된 DOP 만큼의 WTHR를 요청하고, 이에 가용한 만큼의 WTHR를 PE_SLAVE로 얻어오게 된다.

다음과 같이 연산 하나로 수행하는 쿼리는 DOP 개수만큼 WTHR를 가져와서 하나의 연산을 담당할 하나의 PE_SLAVE set을 구성한다. 그 이외는 2*DOP 개수만큼 WTHR를 얻어오고 두 개의 PE_SLAVE set을 구성한다. 이때, WTHR의 개수가 사용자가 명시한 DOP로 수행하기에 부족하면 사용할 수 있는 WTHR만 가지고서 DOP를 재정의하여 수행하게 된다.

```
SELECT * FROM table1;
```

예를 들어 DOP를 4로 결정하여 힌트에 명시한 경우, 2*DOP만큼의 WTHR가 필요한 상황이고 WTHR가 5개뿐이라면 DOP를 2로 재정의하고 4개의 WTHR를 얻어와 PE를 수행한다. 또는 사용 가능한 WTHR가 1개뿐이라면 parallel plan을 만들었더라도 차례로 수행한다. 즉 PE를 수행하지 않는다.

그리고 QC가 얻어온 PE_SLAVE는 쿼리 수행이 끝나면 다시 활용이 가능한 WTHR로 반환되며, 다음 쿼리 수행을 위해 리소스를 점유하지 않게 되어 있다.

13.3. 동작 원리

Tibero RDBMS에서는 parallel hint가 있거나 parallel option이 있는 테이블을 포함한 쿼리를 실행하면, parallel plan을 작성하게 된다.

이때, 생성된 parallel plan의 수행 순서는 다음과 같다.

1. SQL을 수행하는 WTHR가 QC 역할을 담당한다.
2. parallel operator가 있으면, QC는 DOP에 따라 필요한 만큼의 WTHR를 PE_SLAVE로 얻어 온다.
3. PE 수행에 필요한 만큼의 WTHR가 확보되지 않으면, 사용자에게 알리지 않고 내부에서 차례로 처리한다.
4. QC는 순서에 따라 연산이 2-set 구조를 기반으로 수행되도록 PE_SLAVE를 통제하는 역할을 담당한다.
5. 쿼리 수행이 끝나면, QC는 PE_SLAVE에게서 취합한 쿼리 결과를 사용자에게 보내고, 작업을 종료한 PE_SLAVE를 다시 사용 가능한 WTHR로 반환한다.

13.3.1. 2-set 구조

PE는 QC가 생성한 실행 계획을 모든 PE_SLAVE가 공유하도록 구성되어 있으며, 각 PE_SLAVE는 실행 계획의 특정 부분만을 실행하도록 QC가 메시지로 제어한다. PE는 같은 부분의 실행 계획을 DOP 개수만큼의 PE_SLAVE로 나누어 수행하는 intra-parallelism을 지원한다. 이렇게 같은 연산을 수행하는 PE_SLAVE의 묶음을 **PE_SLAVE set**라고 한다.

PE는 한 번에 최대 2개의 PE_SLAVE set을 구성함으로써 전체 실행 계획 중 서로 다른 두 개의 연산을 동시에 수행하는 inter-parallelism을 지원한다. 동시에 수행되는 PE_SLAVE set는 Producer set와 Consumer set으로 나뉜다.

- **Producer set**

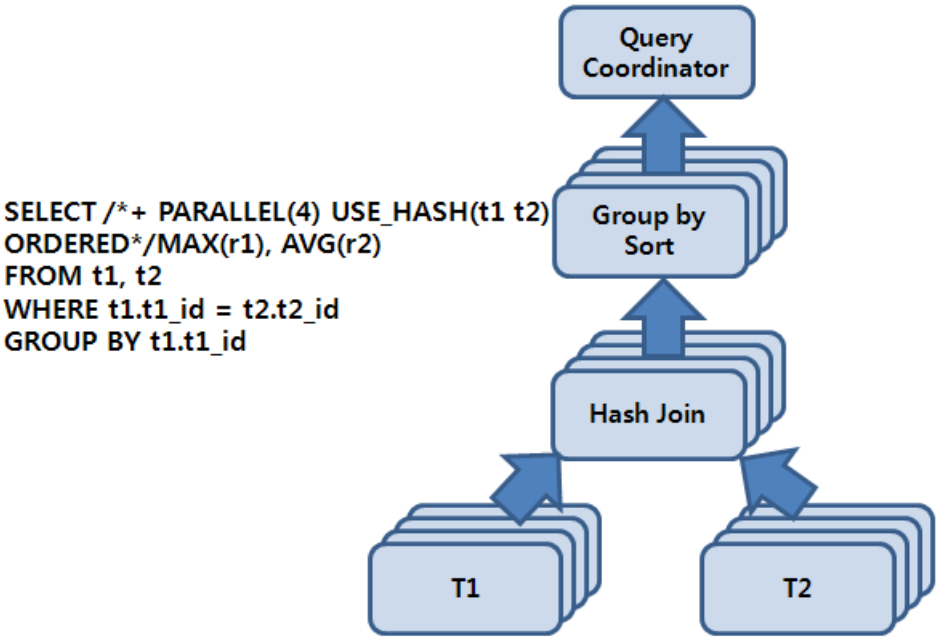
  특정 부분의 실행 계획을 수행하면서 로우(중간 결과)를 추출하여 실행 계획에 명시된 방법에 따라 Consumer set의 PE_SLAVE에게 나눠준다.

- **Consumer set**

  Producer set로부터 로우를 받아 주어진 연산을 수행하고 나면, 다시 Producer로 역할이 바뀌게 된다. PE_SLAVE set는 Producer일 때만 결과를 다음 수행 계획을 갖고 있는 Consumer set에 전달하며, 이러한 PE_SLAVE set을 어떤 순서로 실행 계획 중 어느 부분을 할당해 줄지는 QC가 제어한다.

다음은 2-set 구조와 Producer set, Consumer set를 설명하는 그림이다.

[그림 13.1] Parallel Execution



위 [그림 13.1]의 쿼리에서 parallel hint에 의해 DOP를 고려한 PE 실행 계획이 만들어지면 각각의 연산이 4개(최종으로 결정된 DOP)의 작업 단위로 분할된다. 다시 말해 각각의 PE가 4개의 스레드를 가지기 때문

에 하나의 PE_SLAVE set가 4개의 PE_SLAVE로 구성되고, 이를 다시 Producer Set와 Consumer Set의 2-set 구조로 만들기 위해 총 8 (= 2 x 4)개의 PE_SLAVE가 할당된다.

이때, 할당된 2-set의 PE_SLAVE을 각각 set1, set2라고 하면 각 set가 수행하는 역할에 따라 Producer Set(Tibero Producer Set, 이하 TPS)와 Consumer Set(Tibero Consumer Set, 이하 TCS)로 전환되면서 전체 실행 계획이 수행된다. 즉, TCS가 해시 조인을 처리하기 위해 TPS가 T1 테이블에 스캔을 통해 로우를 공급해 주기를 기다리게 되며, TPS가 테이블 스캔을 시작함으로써 PE 실행 계획에 대한 작업이 시작된다.

set1이 TPS를 담당하여 T1을 스캔하고, set2가 TCS를 담당하여 set1이 보내는 로우에 대해 해시 테이블을 구성하게 되며, set1이 T1 스캔을 마치면 계속해서 T2 스캔을 수행하면서 TCS를 담당하는 set2에게 로우를 공급한다. set1이 T2의 스캔을 마치게 되면, TCS로 역할이 전환되면서 sort group by를 수행하게 되고, 반대로 set2는 TPS로 역할이 전환되면서 해시 조인의 결과를 set1에 공급한다. 마지막으로 set1이 TPS가 되어 sort group by 수행의 결과를 QC에게 보낸다.

이것이 2-set 구조를 이용하여 PE의 실행 계획에서 각각의 연산을 병렬로 수행하는 intra-operation parallelism을 수행하면서 동시에 다양한 연산을 교차하여 inter-operation parallelism을 수행하도록 하는 기본 동작 원리이다.

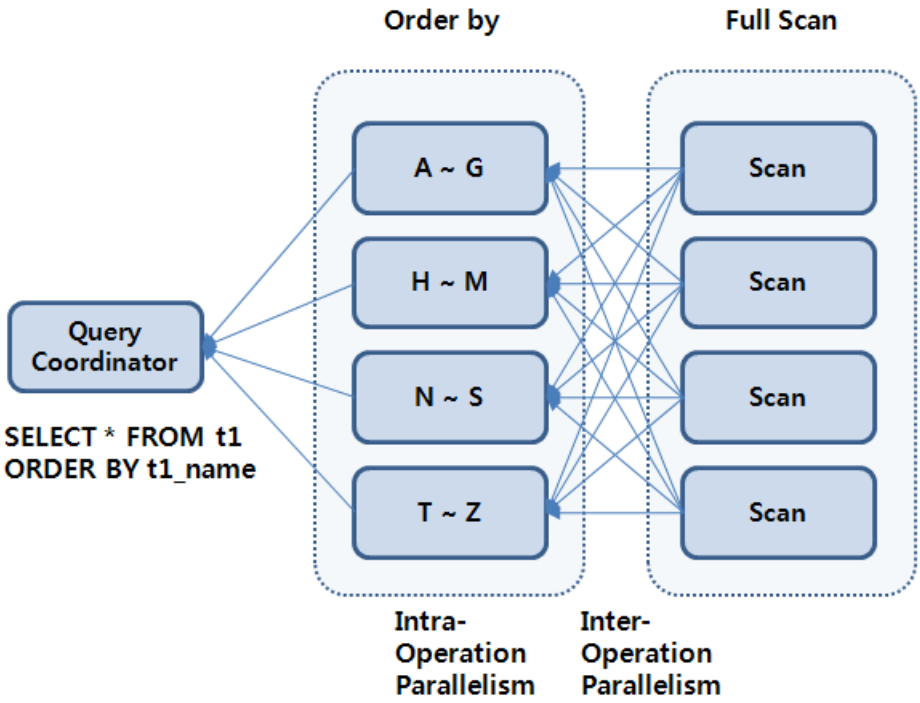
13.3.2. TPS의 분배

TPS는 TCS에 연산의 결과물을 공급하여 TCS가 다음 연산을 처리할 수 있도록 한다. TPS가 결과물을 분배하는데 Tibero RDBMS는 hash, range, broadcast, round-robin, send idxm 5가지 방식을 지원하고 있다. 주로 hash, range, broadcast가 자주 사용된다.

분배방식	설명
hash	TCS가 hash join, hash group by 등의 해시 기반의 연산을 담당하는 경우에 사용한다. send key의 hash value에 따라 해당 로우가 보내질 consumer 스레드가 정해진다.
range	TCS가 order by, sort group by와 같은 정렬 기반의 연산을 담당하는 경우에 사용한다. send key 값의 range에 따라 해당 로우가 보내질 consumer 스레드가 정해진다.
broadcast	Nested Loop 조인이나 pq_distribute 힌트를 사용하여 broadcast를 강제하는 조인을 TCS가 담당하는 경우에 사용한다. 자세한 내용은 “13.4.1. Parallel Query”를 참고한다. 모든 consumer 스레드에 로우가 보내진다.
round-robin	로우를 보낼 consumer를 round-robin 방식으로 실행할 때 사용한다.
send idxm	Parallel DML에서 인덱스와 참조 제약조건을 위해 로우를 보낼 때 사용한다.

본 절에서는 [그림 13.2]에서와 같이 range 방식을 예를 들어 설명한다.

[그림 13.2] Parallel Operations



DOP가 4로 결정된 PE 실행 계획에서 inter-operation parallelism으로 TPS와 TCS 2개의 set가 상호 연동하는 방식을 선택하면, 총 PE_SLAVE 8개가 전체 PE 수행에 참여하게 된다.

PE_SLAVE set 하나가 TPS가 되어 테이블 스캔을 하고, TCS로 설정된 다른 set가 order by를 수행할 수 있도록 로우를 공급해 준다. 이때 4개의 PE_SLAVE는 각각 A~G, H~M, N~S 그리고 T~Z로 정렬 키에 대한 범위를 정하여 정렬하게 되며, 이에 맞게 TPS는 range 방식으로 해당 로우가 담당하는 PE_SLAVE에 전달될 수 있도록 한다.

TCS가 정렬 작업을 완료하고 나면 TPS로 역할이 전환되면서 PE_SLAVE가 설정된 range의 순서대로 정렬 결과를 QC에게 전달함으로써 전체 데이터에 대한 결과가 얻어진다.

13.4. Parallelism 유형

Tibero RDBMS가 제공하는 parallel 연산은 다음과 같다.

- 액세스 방법

  테이블 스캔 그리고 index fast full scan 등

- 조인 방법

  Nested Loop, Sort Merge 그리고 해시 조인

- DDL

CREATE TABLE AS SELECT, CREATE INDEX, REBUILD INDEX 그리고 REBUILD INDEX PARTITION

- DML

  INSERT AS SELECT, UPDATE, DELETE

- 기타 연산

  GROUP BY, ORDER BY, SELECT DISTINCT, UNION, UNION ALL 그리고 Aggregations

13.4.1. Parallel Query

parallel hint가 사용되었거나, parallel option이 있는 테이블에 쿼리를 수행하면 Tibero RDBMS는 parallel plan을 만들어 낸다. 단, parallel로 수행하기 충분한 WTHR가 있을 때만 parallel plan대로 PE를 수행한다.

```
SQL> select /*+ parallel (3) */ * from t1;
SQL> create table t1 (a number, b number) parallel 3;
SQL> select * from t1;
```

Autotrace (explain plan)

Parallel Query에서는 Autotrace (explain plan)를 이용하여 parallel plan을 확인할 수 있다.

```
SQL> set autot on
     select * from t1;

Explain Plan
-----------------------------------------------------------------------------
1  PE MANAGER (0,1)
2    PE SEND QC (RANDOM) (0,1)
3      PE BLOCK ITERATOR (0,1)
4        TABLE ACCESS (FULL): T1 (2,1)
```

위의 실행 계획에서는 PE SEND와 PE RECV(혹은 leaf 노드) 사이를 하나의 PE_SLAVE라고 볼 수 있으며 PE_SLAVE의 최 상단에는 PE SEND가 있어 TPS인 경우 TCS에게 로우를 분배한다.

- PES(parallel execution set) parallel plan에서 하나의 PE_SLAVE set가 수행될 단위를 의미한다. 이를 PE_SLAVE(parallel execution set)라고도 부른다. parallel execution에서 하나의 연산을 하나의 PE_SLAVE set가 담당하지 않는다.

- producer slave: Producer set의 PE_SLAVE

- consumer slave: Consumer set의 PE_SLAVE

PE_SLAVE의 최 하단에는 로우를 만드는 테이블 스캔(혹은 index fast full scan) 연산자가 있거나 PE RECV가 있다. PE RECV는 PE_SLAVE가 TCS일 때 TPS가 분배하는 로우를 받아들인다.

예를 들면, 다음과 같다.

```
SQL> select /*+ parallel (3) use_hash(t2) ordered */ *
from t1, t2
where t1.a=t2.c;

Explain Plan
-------------------------------------------------------------------------------
1  PE MANAGER (0,1)
2    PE SEND QC (RANDOM) (0,1)
3      HASH JOIN (BUFFERED (6,1)
4        PE RECV (0,1)
5          PE SEND (HASH) (0,1)
6            PE BLOCK ITERATOR (0,1)
7              TABLE ACCESS (FULL): T1 (2,1)
8        PE RECV (0,1)
9          PE SEND (HASH) (0,1)
10           PE BLOCK ITERATOR (0,1)
11             TABLE ACCESS (FULL): T2 (2,1)
```

- PE MANAGER

  PE를 시작하고 PE_SLAVE set을 coordination 하는 연산이다. serial execution과 parallel execution 사이의 경계로 이 연산 아래부터는 parallel로 수행된다.

- PE SEND

  producer slave가 로우를 분배하는 연산이다. PE_SLAVE 간의 분배 방법으로는 hash, range, broadcast, round-robin, send idxm 이 있다. 자세한 내용은 “13.3.2. TPS의 분배”를 참고한다.

  PE_SLAVE에서 QC로 로우를 보내는 방법은 다음과 같다.

방법	설명
QC random	QC 역할을 하는 스레드에 보내되 순서와 상관없이 보낸다.
QC order	QC 역할을 하는 스레드에 보내되 Producer가 순서에 맞게 보낸다.

- PE RECV

  consumer slave에서 producer slave가 PE SEND를 통해 분배한 로우를 받아들이는 역할을 하는 연산이다.

- PE BLOCK ITERATOR

테이블 스캔, 인덱스 스캔에서 사용할 **granule**을 요청하고 받는 역할을 하는 연산이다. granule이란 PE를 수행할 때 각 PE_SLAVE에 할당되는 일의 단위 크기로, 크기를 어느 정도로 하느냐에 따라 PE의 성능에 영향을 미친다.

- PE IDXM

  Parallel DML에서 인덱스와 참조 제약조건을 하는 연산이다.

Nested Loop

연산의 특성상 조인의 양쪽 child를 독립된 PES로 구분하지 않고, 한쪽 child를 조인하는 PES와 합쳐 하나의 PES에서 수행되도록 하며, 반대편 child에서 PE SEND를 broadcast 방식으로 PE를 수행한다.

```
SQL> SELECT /*+parallel(3) use_nl(t1 t2) ordered*/ *
     FROM t1, t2
     WHERE a < c;

Explain Plan
------------------------------------------------------------------------------
1 PE MANAGER (0,40)
2  PE SEND QC (RANDOM) (0,40)
3    NESTED LOOPS  (5,40)
4      BUFF (0,9)
5        PE RECV (0,9)
6          PE SEND (BROADCAST) (0,9)
7            PE BLOCK ITERATOR (0,9)
8              TABLE ACCESS (FULL): T1 (2,9)
9      PE BLOCK ITERATOR (0,9)
10       TABLE ACCESS (FULL): T2 (2,9)
```

보통은 오른쪽 child를 조인 연산을 수행하는 slave에 포함시켜 수행하는 parallel plan을 만든다. 하지만 pq_distribute 힌트를 이용하여 어느 쪽 child를 합칠 것인지를 정하면 PE의 성능을 개선할 수 있다.

```
SQL> SELECT /*+parallel(3) pq_distribute(t2 broadcast none) use_nl(t2) ordered*/ *
     FROM t1, t2
     WHERE a=c;

Explain Plan
------------------------------------------------------------------------------
1 PE MANAGER (0,40)
2  PE SEND QC (RANDOM) (0,40)
3    NESTED LOOPS  (5,40)
4      BUFF (0,9)
5        PE RECV (0,9)
6          PE SEND (BROADCAST) (0,9)
```

```
7            PE BLOCK ITERATOR (0,9)
8              TABLE ACCESS (FULL): T1 (2,9)
9      PE BLOCK ITERATOR (0,9)
10        TABLE ACCESS (FULL): T2 (2,9)
```

```
SQL> SELECT /*+parallel(3) pq_distribute (t2 none broadcast) use_nl(t2) ordered*/ *
     FROM t1, t2
     WHERE a=c;

Explain Plan
-----------------------------------------------------------------------------
1 PE MANAGER (0,40)
2  PE SEND QC (RANDOM) (0,40)
3    NESTED LOOPS  (5,40)
4      PE BLOCK ITERATOR (0,9)
5        TABLE ACCESS (FULL): T1 (2,9)
6      TABLE ACCESS (FULL): T2 (2,9)
```

제약 사항

다음의 연산은 parallel로 수행하지 않는다.

- rownum을 생성하는 연산
- cube, rollup을 포함한 group by
- 분석 함수를 포함한 연산
- top-N order by

  예를 들면 다음과 같다.

  ```
  SQL> SELECT * FROM (select * from t1 order by t1.a)
       WHERE rownum < 5;
  ```

- 임시 테이블에 대한 테이블 스캔
- dynamic performance view에 대한 스캔
- 외부 테이블에 대한 스캔
- DBLink를 수행하는 연산
  - index range scan, index full scan, index skip scan, index unique scan

- connect by

13.4.2. Parallel DDL

Tibero RDBMS는 "create table ... as select, create index, rebuild index"를 parallel로 수행할 수 있다.

create table ... as select는 의사 결정 지원 애플리케이션(decision support application)에서 요약 테이블을 만들 때 유용하게 사용할 수 있으며, select 부분과 insert 부분을 parallel로 처리하여 보다 빠른 DDL의 수행을 기대할 수 있다.

DDL 문에 parallel option을 명시하는 방법으로 사용할 수 있으며, DDL과 관련된 문법은 "Tibero RDBMS SQL 참조 안내서"를 참고한다.

13.4.3. Parallel DML

Tibero RDBMS는 insert, update, delete 문에 대해 Parallel DML을 지원하지만, insert into ... values ... 문의 Parallel DML은 지원하지 않는다.

Parallel DML은 주로 큰 데이터를 insert, update, delete 처리하는 배치 작업에 유용하게 사용한다. 다시 말해 트랜잭션이 적은 작업에는 효과적이지 않다.

enable parallel DML

Parallel DML을 **'alter session enable parallel dml'**로 설정한 후 insert, update, delete 문 뒤에 parallel 힌트를 사용해야 DML을 parallel로 수행한다. enable parallel DML을 하지 않으면 DML 문에 힌트를 명시해도 parallel로 수행하지 않는다. 즉 disable parallel dml로 Parallel DML을 못하도록 설정할 수 있다.

Parallel DML은 DOP + 1 개의 트랜잭션으로 동작하며, 커밋은 Two-phase commit으로 이루어진다. 롤백 또한 DML이 여러 개의 트랜잭션으로 이루어졌기 때문에 parallel로 진행된다.

Parallel DML은 여러 개의 트랜잭션으로 진행되기 때문에, Parallel DML 후 커밋 이전까지는 같은 세션에서 select 문으로 해당 테이블을 조회할 수 없으며, Parallel DML로 수정된 테이블에 대한 조회는 커밋 후에 가능하다.

```
SQL> alter session enable parallel dml;
SQL> insert /*+ parallel (3) */ into PE_test3 select * from PE_test3;
10001 rows created.

SQL> select * from PE_test3;
TBR-12066: Unable to read or modify an object after modifying it with PDML.

SQL> insert /*+ parallel (3) */ into PE_test3 select * from PE_test3;
TBR-12067: Unable to modify an object with PDML after modifying it.
```

insert

select 서브쿼리에 parallel 힌트를 주어 insert 뿐만 아니라 insert할 로우를 추출하는 select도 parallel로 수행하여 더 빠르게 DML을 수행할 수 있다.

```
SQL> insert /*+ parallel (3) */ into PE_test3
      select /*+ parallel (3) */ *
      from PE_test3;

10001 rows created.
```

제약 사항

다음의 경우는 Parallel DML로 수행하지 않는다.

- insert into ... values ... 문인 경우
- 반환 문이 있는 DML인 경우
- Merge 문인 경우
- Parallel DML로 수정된 테이블인 경우 커밋을 하기 전까지는 같은 세션에서 DML이나 쿼리로 접근할 수 없다.
- 트리거가 있는 테이블에 대한 DML인 경우
- LOB 타입의 컬럼이 있는 테이블에 대한 DML인 경우
- self reference, delete cascade 제약조건이 있는 DML인 경우
- online rebuild 중인 인덱스를 갖는 테이블에 대한 DML인 경우
- Standby가 있는 경우

제14장 Automatic Performance Monitoring

본 장에서는 Tibero의 성능 진단을 위해서 제공되는 APM에 대해서 설명한다.

14.1. 개요

Tibero DBMS는 DBA가 성능 문제를 진단하는데 도움을 주기 위해 다양한 종류의 통계를 제공하고 있다. Automatic Performance Monitoring(이하 APM)은 이러한 통계 정보를 주기적으로 자동 수집하여 DBA가 이를 위한 작업을 따로 할 필요가 없어졌고, 수집한 통계 자료에 대한 자체적인 분석 리포트 출력 기능을 제공하여 시스템 부하 분석에 도움을 줄 수 있는 기능이다.

APM은 다음의 주요 기능를 수행한다.

- 스냅샷 저장 기능

  스냅샷 저장 기능은 v$sysstat, v$system_event, v$sqlstats, v$sgastat 등 Tibero의 각종 성능 통계 정보를 주기적(보통 1시간)으로 테이블에 저장하여 둔다. 이렇게 저장된 정보를 스냅샷이라 부른다. 이렇게 저장해 놓은 스냅샷 정보를 이용하여 성능 분석 리포트를 만드는 기능을 제공한다. DBA는 특정 구간을 지정하여 리포트를 생성하고 이를 이용해 DB의 성능 문제를 진단할 수 있다.

- 세션 상태 저장 기능

  세션 상태 저장 기능은 1초에 한번씩 현재 RUNNING 상태인 세션들의 ID와 대기 중인 이벤트 정보를 메모리에 저장해 둔다. 이렇게 저장해 놓은 정보는 v$active_session_history 뷰로 조회할 수 있다. 이 뷰를 이용해 DB의 성능 문제를 보다 세밀하게 진단할 수 있다. (현재 세션 상태 저장 기능은 부하 상황에 취약할 수 있으므로, 환경에 따라 주기를 더 늘리기를 권고한다.).

14.2. APM 사용법

14.2.1. tip 설정

스냅샷 저장 기능을 사용하려면 tip 파일에 AUTOMATIC_PERFORMANCE_MONITORING=Y로 설정하고, 세션 상태 저장 기능을 사용하려면 ACTIVE_SESSION_HISTORY=Y로 설정하면 된다. 대부분의 경우 이 정도면 충분하다. 그 밖의 설정을 하기 위해서는 다음과 같은 파라미터를 조절하면 된다.

파라미터	설명
AUTOMATIC_PERFORMANCE_MONITORING	Y로 설정하면 스냅샷 저장 기능 활성화한다. (기본값: N)

파라미터	설명
ACTIVE_SESSION_HISTORY	Y로 설정하면 세션 상태 저장 기능 활성화한다. (기본값: N)
_APM_SNAPSHOT_SAMPLING_INTERVAL	스냅샷 저장 주기를 설정한다. (기본값: 60, 단위: 분)
_ACTIVE_SESSION_HISTORY_SAMPLING_INTERVAL	세션 상태 저장 주기를 설정한다. (기본값: 1초, 단위: 초)
GATHER_SQL_PLAN_STAT	Y로 설정하면 상세한 SQL 통계 정보 수집한다. (기본값 : N)

14.2.2. 관련 테이블과 뷰

저장한 스냅샷과 세션 상태는 관련 테이블과 뷰를 통해 확인할 수 있다. 7일이 지난 스냅샷과 세션 상태는 테이블에서 삭제된다.

- 스냅샷 저장 기능

테이블	설명
_APM_SNAPSHOT	저장된 스냅샷의 ID와 시간에 관한 정보를 관리하는 테이블이다.
_APM_SYSSTAT	V$SYSSTAT 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_SQLSTATS	V$SQLSTATS 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_LATCH:	V$LATCH 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_SYSTEM_EVENT	V$SYSTEM_EVENT 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_SGASTAT	V$SGASTAT 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_LIBRARYCACHE	V$LIBRARYCACHE 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_SQLTEXT	V$SQLTEXT 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_FILESTAT	V$FILESTAT 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_PROCESS	V$PROCESS 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_UNDOSTAT	V$UNDOSTAT 뷰의 스냅샷 정보를 관리하는 테이블이다.
_APM_MISC	세션 수와 같은 기타 정보의 스냅샷 정보를 관리하는 테이블이다.

다음은 저장된 스냅샷의 ID와 시간에 관한 정보를 관리하는 '_APM_SNAPSHOT' 테이블 정보이다.

```
TABLE '_APM_SNAPSHOT'
-------------------------------------------------------------
COLUMN_NAME                      TYPE             CONSTRAINT
-------------------------------- ---------------- ----------
SNAP_ID                          NUMBER
THREAD#                          NUMBER
INSTANCE_NUMBER                  NUMBER
```

```
BEGIN_INTERVAL_TIME               DATE
END_INTERVAL_TIME                 DATE
```

- 세션 상태 저장 기능

  세션 상태 저장 주기에 활동 중인 세션들의 정보를 저장한다.

테이블	설명
_APM_ACTIVE_SESSION_HISTORY	활동 중인 세션 상태 정보를 관리하는 테이블이다.
V$ACTIVE_SESSION_HISTORY	최근 1시간 동안의 세션 상태 정보를 관리하는 테이블이다.

  다음은 최근 1시간 동안의 세션 상태 정보를 관리하는 'V$ACTIVE_SESSION_HISTORY' 테이블 정보이다.

```
VIEW 'V$ACTIVE_SESSION_HISTORY'
-------------------------------------------------------
COLUMN_NAME                     TYPE           CONSTRAINT
------------------------------- -------------- -----------
SAMPLE_ID                       NUMBER
SAMPLE_TIME                     DATE
SID                             NUMBER
SESS_SERIAL_NO                  NUMBER
USER_NO                         NUMBER
WAIT_EVENT                      NUMBER
TIME_WAITED                     NUMBER
SQL_ID                          NUMBER
CURR_HASHVAL                    NUMBER
MODULE_NAME                     VARCHAR(64)
ACTION_NAME                     VARCHAR(64)
CLIENT_INFO_NAME                VARCHAR(64)
```

14.2.3. 수동 스냅샷 생성 기능

APM을 설정하면 정해진 주기에 자동으로 스냅샷이 생성되지만, 원하는 경우 현재 시점의 스냅샷을 남길 수 있다.

```
SQL> exec dbms_apm.create_snapshot();
```

14.2.4. 리포트 작성 기능

저장된 스냅샷과 세션 상태는 테이블과 뷰를 통해 직접 이용할 수도 있지만 보통은 이를 이용해 성능 분석 리포트를 만들게 된다. 현재 저장된 스냅샷 정보를 분석해 성능 분석 리포트를 만드는 기능은 구현되어 있다. 그러나 저장된 세션 상태 정보를 분석해 성능 분석 리포트를 만드는 기능은 구현되어 있지 않다.

스냅샷을 이용한 성능 분석 리포트 작성

_APM_SNAPSHOT 테이블을 조회하여 원하는 기간에 대한 시작, 종료 시각을 확인한다.

다음은 원하는 기간의 시작 과 종료 시점을 각각 begin, end라고 설정한경우 tbsql에서 다음과 같이 입력하여 성능 분석 리포트를 작성하는 예이다.

```
SQL> exec dbms_apm.report_text(begin, end);
```

성능 분석 리포트는 다음의 경로에 파일로 생성된다.

```
$TB_HOME/instance/$TB_SID/apm_report.{mthr_pid}.{current_time}
```

성능 분석 항목

다음은 성능 분석 항목에 대한 설명이다.

- Report Summary
  - Cache Sizes
  - Load Profile
  - Instance Efficiency Percentages (Target 100%)
  - Shared Pool Statistics
  - Top 5 Timed Wait Events
- Wait Events & Active Stats
  - Wait Event Class
  - Wait Events
  - Instance Activity Stats
- SQL Statistics

  - SQL Ordered by Elapsed Time
  - SQL Ordered by Gets
  - SQL Ordered by Executions
  - SQL Ordered by Parse Calls

- IO Stats
  - Tablespace IO Statistics
  - File IO Statistics

- SGA Statistics
  - Buffer Pool Statistics
  - Library Cache Statistics

- PGA Statistics
  - PGA Statistics

- Wlock Statistics
  - Wlock Statistics

- Undo Statistics
  - Undo Statistics

- Spinlock Statistics
  - Spinlock(Latch) Statistics

- Tibero Init. Parameters(.tip)

Appendix A. tbdsn.tbr

tbdsn.tbr 파일은 클라이언트가 Tibero RDBMS의 데이터베이스에 접속하기 위해 필요한 정보를 가지고 있는 환경설정 파일이다.

기본적으로 tbdsn.tbr 파일에는 SID, 호스트, 포트 번호 등의 정보가 포함되어 있다.

예를 들면 다음과 같다.

```
tb=(
    (INSTANCE=(HOST=168.1.1.33)
              (PORT=8629)
              (DB_NAME=tibero)
    )
)
```

A.1. tbdsn.tbr의 구조

tbdsn.tbr 파일은 다음과 같은 구조로 이루어져 있다.

```
SID_1=(
       (INSTANCE=(항목1=값1)
                 (항목2=값2)
         ...
       )
)

SID_2=(
       (INSTANCE=(항목1=값1)
                 (항목2=값2)
         ...
      )
)

TB_NLS_LANG=EUCKR

TBCLI_LOG_LVL=TRACE

...
```

SID는 클라이언트에서 해당 서버를 식별하기 위한 고유한 이름으로 위 구조와 같이 세부 항목이 포함되어 있으며, 병렬 구조의 형태를 띤다.

tbdsn.tbr에서 SID로 사용할 수 있는 표준문자는 문자(A-Z, a-z), 숫자(0-9), 및 하이픈(-)이다.

SID의 세부 항목은 다음과 같다.

항목	설명
HOST	서버의 IP 주소이다. (예: HOST=168.1.1.33)
PORT	서버의 포트 번호이다. (예: PORT=8629)
DB_NAME	데이터베이스의 이름이다. (예: DB_NAME=tibero)

SID 정보 외에 클라이언트 환경설정도 할 수 있다. 이 설정들은 환경변수로도 지정 가능하며 양쪽 다 설정되어 있을 경우 tbdsn.tbr 파일 설정을 우선적으로 따른다.

항목	설명
TB_NLS_LANG	클라이언트에서 사용하는 캐릭터셋을 지정할 수 있다. (예: TB_NLS_LANG=EUCKR)
TBCLI_LOG_LVL	CLI의 로그 레벨을 지정할 수 있다. (예: TBCLI_LOG_LVL=TRACE)

A.2. 이중화 서버의 설정

이중화 서버((Replication server)는 물리적으로 독립된 여러 개의 서버를 동일하게 복제한 것을 의미한다. 동일하게 복제된 서버를 임의로 접속할 수 있고, 하나의 서버가 정지했을 때에도 다른 서버 대신 접속하여 같은 작업을 수행할 수 있다.

tbdsn.tbr 파일에서 이중화 서버를 구성하려면 다음과 같이 하나의 SID로 이중화 서버를 INSTANCE 항목으로 설정하면 된다.

```
tb=(
    (INSTANCE=(HOST=168.1.1.33)
              (PORT=8629)
              (DB_NAME=tibero)
    )
    (INSTANCE=(HOST=192.168.1.25)
              (PORT=8629)
              (DB_NAME=tibero2)
```

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

A.3. 로드 밸런싱의 설정

Tibero RDBMS에서는 이중화 서버 중에서 특정 서버의 집중적인 접속을 막으려고 로드 밸런싱(Load balancing) 기능을 지원한다. 즉, 특정 서버의 집중적인 접속을 분산시킴으로써 시스템의 과부하를 막고 더 나은 접속 환경을 제공한다.

tbdsn.tbr 파일에서 로드 밸런싱 기능을 설정하는 방법은 다음과 같다.

```
tb=(
    (INSTANCE=(HOST=168.1.1.33)
              (PORT=8629)
              (DB_NAME=tibero)
    )
    (INSTANCE=(HOST=192.168.1.25)
              (PORT=8629)
              (DB_NAME=tibero2)
    )
    (INSTANCE=(HOST=localhost)
              (PORT=8629)
              (DB_NAME=tibero)
    )
    (LOAD_BALANCE=Y)
)
```

위 예제에서 보듯이 SID 내의 LOAD_BALANCE 값을 Y로 설정하면 로드 밸런싱 기능을 사용할 수 있다. 반면에 LOAD_BALANCE 값이 없거나 Y 외의 다른 값을 입력하면 로드 밸런싱 기능은 동작하지 않는다.

A.4. Failover의 설정

Tibero RDBMS가 TAC 또는 이중화된 서버로 설정된 상태에서 장애로 인해 중단되면 CLI 모듈은 다른 인스턴스 또는 이중화된 서버로 접속하여 해당 세션을 자동으로 복구한다.

tbdsn.tbr 파일에서 Failover 기능을 설정하는 방법은 다음과 같다.

```
tb=(
    (INSTANCE=(HOST=168.1.1.33)
              (PORT=8629)
              (DB_NAME=tibero)
    )
```

```
    (INSTANCE=(HOST=192.168.1.25)
              (PORT=8629)
              (DB_NAME=tibero2)
    )
    (INSTANCE=(HOST=localhost)
              (PORT=8629)
              (DB_NAME=tibero)
    )
    (USE_FAILOVER=Y)
)
```

위 예제에서 보듯이 SID 내의 USE_FAILOVER 값을 Y로 설정하면 Failover 기능을 사용할 수 있다. 단, 문장(Statement) 레벨까지의 복구는 지원하지 않는다.

Appendix B. V$SYSSTAT

동적 뷰인 **V$SYSSTAT**를 조회하면 시스템의 각종 통계 정보를 조회할 수 있다. DBA는 조회된 항목을 통해 Tibero RDBMS 서버를 모니터링하고 튜닝하는 데 활용할 수 있다.

다음은 V$SYSSTAT에 포함되어 있는 항목에 대한 설명이다.

항목	설명
consistent block gets	CR block을 요청한 횟수
consistent multi gets	Multi CR block을 요청한 횟수
block gets (CRX)	CR block examine을 요청한 횟수
block examine rowlock	Current block examine을 요청한 횟수
current block gets - waits	Current block 을 얻지 못하고 기다린 횟수
block disk read	디스크로부터 block을 읽은 횟수
multi block disk read - requested	디스크로부터 multi block을 읽은 횟수
current block gets	Current block을 요청한 횟수
current block gets - no wait	Current block을 기다리지 않고 얻은 횟수
block disk read undo header block	디스크로부터 undo block을 읽은 횟수
block changes - current + consistent	버퍼 캐시의 block을 변경한 횟수
multi block read complete	Multi-block을 요청한 후 읽기(read)가 끝날 때 까지 기다린 횟수
current block cleanout	Current Block에 대한 Cleanout의 횟수
current block partial cleanout	Current Block에 대한 부분 Cleanout의 횟수
buffer cache invalidate	버퍼 캐시를 무효화한 횟수
block corrupt logging	Buffer cache corrupt의 표시 횟수
block pin - not conflict	Buffer cache pin의 횟수
block unpin	Buffer cache unpin의 횟수
block wait (ckpt + writing)	버퍼 캐시가 체크포인트와 flush를 완료하기를 기다린 횟수
block wait (writing)	버퍼 캐시가 flush 되기를 기다린 횟수
block copy in ckpt progress or write	체크포인트 중인 버퍼 캐시를 복사한 횟수
block copy on write	Flush 중인 버퍼 캐시를 복사한 횟수
candidate bh scanned	버퍼 캐시 체인에서 버퍼를 찾은 횟수

항목	설명
block cleanouts	블록 전체를 Cleanout 한 횟수
block partial cleanout - total	부분 Cleanout의 횟수
checkpoint requests	체크포인트의 요청 횟수
checkpoint requested - media recovery	미디어 복구에 의한 체크포인트 횟수
checkpoint requested - instance recovery	인스턴스 복구에 의한 체크포인트 횟수
switching logfile waits - incomplete checkpoint	로그 파일 스위치에 의한 체크포인트 횟수
checkpoint requests - timeout	타임아웃에 의한 체크포인트 횟수
checkpoint requests - cache invalidate	버퍼 무효화에 의한 체크포인트 횟수
checkpoint requests - log block interval	CKPT_LOG_INTERVAL에 의한 체크포인트 횟수
checkpoint requests - next logfile	다음의 로그 파일 스위치를 대비한 체크포인트 횟수
consistent block created - converted from current	Current Buffer가 CR Buffer로 변경된 횟수
consistent block cleanout	CR Block에 대한 cleanout의 횟수
consistent block partial cleanout	CR Block에 대한 partial cleanout의 횟수
consistent rollback	CR Block에 대한 Rollback 횟수
consistent block created - clone	CR Block을 만들기 위해 복제한 횟수
consistent gets - no clone	복제 없이 CR Block을 얻은 횟수
tx table consistent rollback	트랜잭션 테이블에 대한 Rollback 횟수
dbwr flush requests - make clean	빈 버퍼를 만들기 위해 DBWR에 flush를 요청한 횟수
dbwr flush requests - force	TAC에서 다른 인스턴스 요청에 의해 block을 flush한 횟수
dbwr flush requests - checkpoint	체크포인트를 위해 DBWR에 flush를 요청한 횟수
dbwr lru list scans	DBWR가 LRU를 검색한 횟수
dbwr writing list scans	DBWR가 flush할 LRU를 검색한 횟수
dbwr write - OS	DBWR가 OS에 write를 요청한 횟수
dbwr write block count - OS	DBWR가 flush한 block의 수
dbwr written block count	DBWR가 다른 인스턴스를 위해 flush한 block의 수
dbwr written block count - checkpoint list	DBWR가 체크포인트를 위해 flush한 block의 수
dbwr written block count - lru list	DBWR가 빈 버퍼를 위해 flush한 block의 수
dbwr written block count - adjacent	DBWR가 multi block write를 위해 flush한 block의 수
dbwr writes for incremental checkpoint	DBWR가 incremental 체크포인트를 위해 작업한 횟수
dbwr multi block writes	DBWR가 multi block write한 횟수
dbwr log flush requests	DBWR가 LGWR 에 flush를 요청한 횟수

항목	설명
log flush	LGWR 가 Log buffer를 flush한 횟수
lgwr logfile switch	로그 파일 스위치의 횟수
lgwr wait for copy	LBWR가 로그 버퍼에 복사하는 세션을 기다린 횟수
redo wait for lgwr	Log buffer flush가 완료되기를 기다린 횟수
loga write	LARC가 기록한 횟수
free blocks scanned block count	빈 버퍼를 찾기 위해 검색한 버퍼의 개수
free blocks scanned - pinned seen	빈 버퍼를 찾던 중 pin된 버퍼의 수
free blocks scanned - dirty seen	빈 버퍼를 찾던 중 dirty 버퍼의 수
free blocks scanned - writing clean seen	빈 버퍼를 찾던 중 flush 중인 버퍼의 수
free blocks requested	빈 버퍼를 찾아 본 횟수
free blocks total waits	빈 버퍼를 찾지 못하고 기다린 횟수
free blocks - invoke dbwr	빈 버퍼를 찾지 못해 DBWR에 flush를 요청한 횟수
redo entries	Redo 로그를 로그버퍼에 복사한 횟수
redo space allocation trials	Redo 로그를 복사할 공간을 할당 받은 횟수
redo log space requests	Redo 버퍼가 부족하여 flush를 요청한 횟수
redo wait for logfile switch	Redo 로그를 남기기 위해 로그 파일 스위치를 기다린 횟수
redo wait for flush	Redo 로그 flush를 기다린 횟수
redo write	Redo flush의 횟수
redo write multi	Redo flush를 여러 요청을 한꺼번에 처리한 횟수
block updates	Redo 로그를 남기고 block을 수정한 횟수
[CCC,MASTER] cr request on the master(requester = master)	[CCC] master에서 holder에게 CR 요청을 한 횟수
[CCC,MASTER] cr request fail on the master(requester = master)	[CCC] master에서 요청한 CR이 실패한 횟수
[CCC,MASTER] cr make on the master(holder = master)	[CCC] master에서 만들어 준 CR block의 수
[CCC,MASTER] cr make fail on the master(holder = master)	[CCC] master에서 CR을 만들다가 실패한 횟수
[CCC,SHADOW] cr request on the shadow	[CCC] shadow 에서 요청한 CR request의 수
[CCC,SHADOW] cr request fail on the shadow	[CCC] shadow에서 요청한 CR request가 실패한 횟수
[CCC,SHADOW] cr make on the holder	[CCC] holder에서 받은 CR request의 수
[CCC,SHADOW] cr make fail on the holder	[CCC] holder에서 CR 생성에 실패한 횟수

항목	설명
Number of times to wait & process CR reply for prefetch	Multi-block prefetch 시에 Lock을 처리하기 위해 기다린 횟수 및 시간
Number of sleeps to wait CR reply for prefetch	JC_TAC_PREFETCH_POST 중에서 다른 노드의 응답을 기다린 횟수 및 시간
Number of locks prefetched	DPL 및 IndexFastBuild를 위해 Lock을 미리 설정한 횟수
Number of waiting previous CR reply	CR 요청을 할 때 이미 CR 요청 중이라 종료를 기다린 횟수
Number of waiting BUSY flag to cancel lock	Lock cancel 을 위해 대기한 횟수
Number of waiting CCC remastering	Cache Lock이 remastering 중이라 Lock 요청을 일시 중지한 횟수
Number of waiting RCOC status	Cache에서 다른 노드의 상태를 알아보기 위해 기다린 횟수
Number of waiting RCOC start	Cache Recovery가 시작하기를 기다린 횟수
Number of times to reclaim CCC during reconfiguration	Reconfiguration 중에 Cache RSB등이 모자라서 메모리를 회수한 횟수
Number of times retrying to reclaim CCC	Cache RSB 메모리 회수가 모자라서 재시도한 횟수
Number of times to wait the other reclaim CCC	다른 스레드에서 Cache RSB 메모리 회수가 종료되기를 기다린 횟수
Number of times to allocate new CCC RSB	RSB를 새로 만든 횟수
Number of times to reclaim CCC RSB	메모리 회수 작업의 횟수
Number of blocks received from a remote CR Server	다른 노드에서 대신 만든 CR를 받은 횟수
Number of locks granted from the master	다른 노드에서 Lock을 받은 횟수
Number of UP locks processed as master	master로서 Lock UP을 처리한 횟수
Number of DOWN locks processed as master	master로서 Lock Down을 처리한 횟수
Number of waiting CWS remastering	Wlock Lock이 remastering 중이라 Lock 요청을 일시 중지한 횟수
Number of waiting RCOW status	Wlock에서 다른 노드의 상태를 알아보기 위해 대기한 횟수
Number of waiting RCO start	Wlock Recovery가 시작하기를 기다린 횟수
Number of times to reclaim CWS during reconfiguration	Reconfiguration 중에 Wlock RSB 등이 모자라서 메모리를 회수한 횟수
Number of times retrying to reclaim CWS	Wlock RSB 메모리 회수가 모자라서 재시도한 횟수

항목	설명
Number of times to wait the other reclaim CWS	다른 스레드에서 Wlock RSB 메모리 회수가 종료되기를 기다린 횟수
Number of times to wait ctx sync	다른 노드와 reconfiguration 상태를 맞추기 위해 대기한 횟수
Number of times to wait acf task ready	TAC 배경 프로세스의 준비를 기다린 횟수
Number of times to retry receive INC packet	TAC 패킷을 받는 중에 EAGAIN이 발생한 횟수
Number of receiving INC packet	TAC 용의 패킷을 받은 횟수
Number of errors during receiving INC packet	TAC 패킷을 받는 중에 발생한 에러의 횟수
Number of times to retry send INC packet	TAC 패킷을 보내는 중에 EAGAIN이 발생한 횟수
Number of sending INC packet	TAC 용의 패킷을 보낸 횟수
Number of errors during sending INC packet	TAC 패킷을 보내는 중에 발생한 에러의 횟수
Number of re-scaning send msg Q	보낼 TAC 패킷이 남아서 큐를 재검색한 횟수
Number of times to wait DSPC working	DSPC 스레드가 resume 되기를 기다린 횟수
Number of times to wait DSPC suspend	DSPC 스레드가 suspend 되기를 기다린 횟수
csr fetch insert	DML 문장의 INSERT의 수행 시간
csr fetch delete	DML 문장의 DELETE의 수행 시간
csr fetch update	DML 문장의 UPDATE의 수행 시간
csr fetch select	SELECT 쿼리의 수행 시간
ex workarea cache hit count	쿼리 실행에 필요한 private context 에 대한 cache hit의 횟수
ex workarea cache miss count	쿼리 실행에 필요한 private context 에 대한 cache miss의 횟수
ex cache miss count for different param types	쿼리 실행에 필요한 private context 에 대한 cache miss 중 바인드 파라미터의 타입이 달라서 생긴 횟수

Appendix C. 문제 해결

본 장에서는 Tibero RDBMS를 사용할 때 발생할 수 있는 문제를 해결하는 방법을 설명한다.

C.1. 데이터베이스 접속

- 문제

```
$ tbsql SYS/tibero

Tibero RDBMS 4SP1

Copyright (c) 2008, 2009, 2011 Tibero Corporation. All rights reserved.

[UNIX Domain Socket] connect failed : Connection refused
TBR-2131: Generic I/O error

No longer connected to server.
```

tbSQL 유틸리티를 이용하여 Tibero RDBMS의 데이터베이스에 접속할 때 이러한 메시지가 출력되는 문제이다.

- 해결

우선 먼저 Tibero RDBMS가 기동 되어 있는지 확인한다.

Tibero RDBMS의 기동 여부에 따라 이를 해결하는 방법이 다르다.

– Tibero RDBMS가 기동 되지 않은 경우

tbctl 명령어를 사용하여 Tibero RDBMS의 프로세스가 실행되고 있는지 확인한다.

```
$ tbctl pid

  ...Tibero RDBMS 프로세스가 없다.
```

또는 ps와 grep 명령어로 Tibero RDBMS의 프로세스가 실행되고 있는지 확인한다.

```
$ ps -ef | grep tbsvr
  tibero   22919 22590  0 15:37 pts/10   00:00:00 grep tbsvr

  ...Tibero RDBMS 프로세스가 없다.
```

Tibero RDBMS 프로세스가 없다면 **tbboot** 명령어를 사용하여 Tibero RDBMS를 기동한다. tbboot 명령어를 사용하는 방법에 대한 내용은 “2.5.1. tbboot”를 참고한다.

– Tibero RDBMS가 기동 되어 있는 경우

다음과 같이 Tibero RDBMS 프로세스는 실행되고 있으나, 데이터베이스에 접속되지 않는 경우는 대체로 서버와 클라이언트의 환경설정이 서로 맞지 않을 때 발생한다.

```
$ ps -ef | grep tbsvr
tibero   32036 32035  0 20:00 pts/7    00:00:00 tbsvr         -n ...
tibero   32037 32036  0 20:00 pts/7    00:00:00 tbsvr_WT001   -n ...
tibero   32038 32036  0 20:00 pts/7    00:00:00 tbsvr_SEQW    -n ...
tibero   32039 32036  0 20:00 pts/7    00:00:00 tbsvr_LOGW    -n ...
tibero   32040 32036  0 20:00 pts/7    00:00:00 tbsvr_LOGA    -n ...
tibero   32041 32036  0 20:00 pts/7    00:00:00 tbsvr_CKPT    -n ...
tibero   32042 32036  0 20:00 pts/7    00:00:00 tbsvr_BLKW000 -n ...
tibero   32057 30048  0 20:00 pts/7    00:00:00 grep tbsvr
```

우선 $TB_HOME/client/config 디렉터리에 있는 **tbdsn.tbr** 파일을 확인하여 클라이언트의 설정 부분을 확인한다.

<tbdsn.tbr>

```
Tb4=(
     (INSTANCE=(HOST=localhost)
               (PORT=8629)
               (DB_NAME=Tb4)
     )
)
```

위 예제에서 보듯이, 호스트는 **localhost**, 포트 번호는 **8629**로 입력되어 있다. 이와 같은 정보가 서버의 환경설정 파일과 일치하는지 확인한다.

확인하는 방법은 다음과 같다.

<$TB_HOME/config/$TB_SID.tip>

```
# -----*----- conf -----*-----
#
# Tibero RDBMS 초기화 파라미터
#

LISTENER_PORT=6666
......
```

$TB_SID.tip 파일을 보면 클라이언트에 설정된 포트 번호와 일치하지 않다는 것을 확인할 수 있다. 이로 인해 Tibero RDBMS의 데이터베이스에 접속할 수 없는 원인이 된다.

또한, 포트 번호가 설정되지 않으면 Tibero RDBMS 내부에 테스트 용인 임의의 포트 번호가 설정된다. 그러면 포트 번호의 충돌 등의 문제를 유발할 수 있다. **이러한 경우 Tibero RDBMS의 안정적인 동작을 보장할 수 없다.** 그러므로 반드시 포트 번호를 동일하게 설정해야 한다.

단, 로컬에서 접속할 때에는 tbdsn.tbr 파일에 포트 번호가 서버의 $TB_SID.tip 파일과 달라도 Tibero RDBMS는 기동된다. 반면에 원격으로 접속할 때에는 반드시 서버 쪽의 포트 번호와 일치해야 Tibero RDBMS가 기동된다.

Appendix D. 클라이언트 환경 변수

다음은 클라이언트에서 설정할 수 있는 환경 변수에 대한 설명이다.

환경 변수	설명
TB_NLS_LANG	클라이언트의 캐릭터 셋이다. 다음의 값 중에 하나를 선택한다. - ASCII - EUCKR - MSWIN949 (기본값) - UTF8 - SJIS - JA16SJIS - JA16SJISTILDE - JA16EUC - JA16EUCTILDE - GBK
TB_NLS_DATE_FORMAT	DATE 표현 형식이다. 다음은 유효한 DATE 형식 마스크이다. - YYYY/MM/DD (기본값) - MM/DD/YYYY - DD RM YYYY - RR/MM/DD
TB_NLS_TIMESTAMP_FORMAT	TIMESTMAP 표현 형식이다. 다음은 유효한 TIMESTAMP 형식 마스크이다. - YYYY-MM-DD HH:MI:SS.FF - YYYY/MM/DD HH24:MI:SS

환경 변수	설명
	- RR/MM/DD HH24:MISSXFF (기본값)
TBCLI_LOG_LVL	tbCLI의 로그 레벨을 설정한다. - WINDOWS : C:\tbcli_날짜시간.log - UNIX : /tmp/tbcli_날짜시간.log 다음의 값 중에 하나를 설정한다. – FATAL : 심각한 오류로 인해 응용 프로그램의 중단이 필요한 경우 사용한다. – ERROR : 응용 프로그램 작업이 계속 수행된다. – WARN : 오류는 아니지만, 문제가 될 소지가 있다. – INFO : 응용 프로그램의 진행 상황을 추적할 수 있는 정보, 일반적으로 coarse-grained 정보이다. (예 : API 호출) – DEBUG : 응용 프로그램을 디버깅하는데 도움이 되는 fine grained 로깅이다. (예 : 로깅 매개 변수, 계산하는 동안의 임시 값 등) (기본값) – TRACE : 매우 자세한 정보, DEBUG보다 자세한 로깅 정보를 제공한다. (예 : 정형화되지 않은 이진 데이터 덤프, 메모리 덤프 등과 같이 덤핑 확장 로그) – INTERNAL : 가장 자세한 정보를 로깅한다. 이 수준은 내부 디버깅을 위해서만 사용된다. (예 : lock 정보)
TBXA_LOG_LVL	tbxa의 로그 레벨을 설정한다. - WINDOWS : C:\tbxa_날짜시간.log - UNIX : /tmp/tbxa_날짜시간.log 다음의 값 중에 하나를 설정한다. – FATAL : 심각한 오류로 인해 응용 프로그램의 중단이 필요한 경우 사용한다. – ERROR : 응용 프로그램 작업이 계속 수행된다. – WARN : 오류는 아니지만, 문제가 될 소지가 있다. – INFO : 응용 프로그램의 진행 상황을 추적할 수 있는 정보, 일반적으로 coarse-grained 정보이다. (예 : API 호출)

환경 변수	설명
	– DEBUG : 응용 프로그램을 디버깅하는데 도움이 되는 fine grained 로깅이다. (예 : 로깅 매개 변수, 계산하는 동안의 임시 값 등) (기본값) – TRACE : 매우 자세한 정보, DEBUG보다 자세한 로깅 정보를 제공한다. (예 : 정형화되지 않은 이진 데이터 덤프, 메모리 덤프 등과 같이 덤핑 확장 로그) – INTERNAL : 가장 자세한 정보를 로깅한다. 이 수준은 내부 디버깅을 위해서만 사용된다. (예 : lock 정보)

색인

M

N

O

P

Q

R

S

T

U

V

W

X

ㄱ

ㄴ

ㄷ

ㄹ

ㅁ

ㅂ

ㅅ

ㅇ

ㅈ