Tibero

관리자 안내서

Tibero 7

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

Website

http://www.tmaxtibero.com

기술서비스센터

Tel : +82-1544-8629

E-Mail : info@tmax.co.kr

Open Source Software Notice

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

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

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

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

안내서 정보

안내서 제목: Tibero 관리자 안내서

발행일: 2024-08-22

소프트웨어 버전: Tibero 7.2.2

안내서 버전: v7.2.2

내용 목차

그림 목차

예 목차

안내서에 대하여

안내서의 대상

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

안내서의 전제 조건

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

참고

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

안내서 구성

Tibero 관리자 안내서는 총 18개의 장과 Appendix로 구성되어 있다.

각 장의 주요 내용은 다음과 같다.

- 제1장: Tibero 소개

  DBA 측면에서 Tibero의 기본 개념과 이를 구성하는 프로세스와 디렉터리 구조를 기술한다.

- 제2장: 관리의 기본

  Tibero를 관리하기 위한 기본적인 사항을 기술한다.

- 제3장: 파일과 데이터의 관리

  Tibero의 파일과 데이터를 관리하는 방법을 기술한다.

- 제4장: 스키마 객체의 관리

  Tibero의 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: 데이터 암호화

  데이터 암호화 기능을 사용하고 이를 관리하는 방법을 기술한다.

- 제7장: 통신 암호화

  통신 암호화 기능을 사용하고 이를 관리하는 방법을 기술한다.

- 제8장: Tibero Separation of Duties

  Tibero의 관리자의 권한을 분리시킨 Tibero Separation of Duties의 설정 및 활용 방법을 기술한다.

- 제9장: Tibero Virtual Private Database

  Tibero의 보안상 혹은 특정 서비스를 목적으로 필요한 유저에 해당하는 데이터만 다룰 수 있게 하는 Tibero Virtual Private Database의 사용 방법을 기술한다.

- 제10장: Tibero Label Security

  레이블을 통하여 사용자의 접근을 제어하는 기능인 Label Security에 대해서 설명한다

- 제11장: 백업과 복구

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

- 제12장: 분산 트랜잭션

  분산 트랜잭션을 지원하기 위해 Tibero가 제공하는 기능을 기술한다.

- 제13장: Tibero Standby Cluster

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

- 제14장: Tibero Cluster Manager

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

- 제15장: Tibero Active Cluster

  확장성, 고가용성을 목적으로 하는 Tibero Active Cluster를 기술한다.

- 제16장: Parallel Execution

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

- 제17장: Tibero Performance Repository

  Tibero의 성능 진단을 위해서 제공되는 Tibero Performance Repository의 사용 방법을 기술한다.

- 제18장: Tibero Recovery Catalog

  Tibero Recovery Catalog의 구성요소와 동작 및 운영 방법을 설명한다.

- Appendix A: tbdsn.tbr

  클라이언트가 Tibero에 접속하기 위해 필요한 tbdsn.tbr 환경설정 파일을 기술한다.

- Appendix B: V$SYSSTAT

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

- Appendix C: 문제 해결

  Tibero를 사용할 때 발생할 수 있는 문제를 해결하는 방법을 기술한다.

- Appendix D: 클라이언트의 환경변수

  클라이언트에서 설정할 수 있는 환경변수를 기술한다.

안내서 규약

표기	의미
<<AaBbCc123>>	프로그램 소스 코드의 파일명
<Ctrl>+C	Ctrl과 C를 동시에 누름
[Button]	GUI의 버튼 또는 메뉴 이름
진하게	강조
" "(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
'입력항목'	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일 계정, 웹 사이트
>	메뉴의 진행 순서
+----	하위 디렉터리 또는 파일 있음
|----	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
주의	주의할 사항
[그림 1.1]	그림 이름
[예 1.1]	예제 이름
`AaBbCc123`	Java 코드, XML 문서
[*command argument*]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨
|	선택 사항. 예) A|B: A나 B 중 하나
…	파라미터 등이 반복되어서 나옴
${ }	환경변수

시스템 사용 환경

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

관련 안내서

안내서	설명
Tibero 설치 안내서	설치 시 필요한 시스템 요구사항과 설치 및 제거 방법을 기술한 안내서이다.
Tibero tbCLI 안내서	Call Level Interface인 tbCLI의 개념과 구성요소, 프로그램 구조를 소개하고 tbCLI 프로그램을 작성하는 데 필요한 데이터 타입, 함수, 에러 메시지를 기술한 안내서이다.
Tibero 애플리케이션 개발자 안내서	각종 애플리케이션 라이브러리를 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero External Procedure 안내서	External Procedure를 소개하고 이를 생성하고 사용하는 방법을 기술한 안내서이다.
Tibero JDBC 개발자 안내서	Tibero에서 제공하는 JDBC 기능을 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero tbESQL/C 안내서	C 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbESQL/COBOL 안내서	COBOL 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbPSM 안내서	저장 프러시저 모듈인 tbPSM의 개념과 문법, 구성요소를 소개하고, 프로그램을 작성하는 데 필요한 제어 구조, 복합 타입, 서브 프로그램, 패키지와 SQL 문장을 실행하고 에러를 처리하는 방법을 기술한 안내서이다.
Tibero tbPSM 참조 안내서	저장 프러시저 모듈인 tbPSM의 패키지를 소개하고, 이러한 패키지에 포함된 각 프러시저와 함수의 프로토타입, 파라미터, 예제 등을 기술한 참조 안내서이다.
Tibero 유틸리티 안내서	데이터베이스와 관련된 작업을 수행하기 위해 필요한 유틸리티의 설치 및 환경설정, 사용 방법을 기술한 안내서이다.
Tibero TAS 안내서	Tibero Active Cluster (TAS)를 사용해서 Tibero의 파일을 관리하고자 하는 관리자를 대상으로 기술한 안내서이다.
Tibero	Tibero를 사용하는 도중에 발생할 수 있는 각종 에러의 원인과 해결 방법을 기술한 안내서이다.

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

제1장 Tibero 소개

본 장에서는 Tibero의 주요 기능과 프로세스 구조, 디렉터리 구조를 간단히 소개한다.

1.1. 개요

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.

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

기존 DB의 단점을 보완하기 위해 Tibero는 독자적인 Tibero Thread Architecture를 채택하고 구현하였다. 한정된 서버 프로세스의 CPU 및 메모리 등의 시스템 리소스를 효율적으로 사용하면서 뛰어난 성능과 안정성 및 확장성을 보장하고 편리한 개발 환경과 관리 기능을 제공한다. Tibero는 초기 설계부터 대규모 사용자, 대용량 데이터, 강화된 안정성, 향상된 호환성 측면 등에서 다른 DBMS와 차별화를 고려하여 개발되었다.

Tibero는 이처럼 기업이 원하는 최적의 데이터베이스 환경을 제공하는 대표적인 DB이다.

1.2. 주요 기능

대용량의 데이터를 관리하고 안정적인 비즈니스의 연속성을 보장하는 데이터 관리 솔루션인 Tibero는 DB 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 DB 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 DB의 최대 이슈인 고가용성과 고성능을 모두 해결하는 기능이다. Tibero는 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 "Tibero 관리자 안내서"의 "제15장 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.

  Tibero는 이러한 요구사항에 맞추어 온라인 트랜잭션 처리(On-Line Transaction Processing, 이하 OLTP) 환경에 최적화된 기능을 제공할 뿐만 아니라 OLAP(On-Line Analytical Processing) 환경에 최적화된 SQL 병렬 처리 기능을 제공하고 있다. 이로 인해 쿼리는 빠른 응답 속도로 수행되며 기업의 빠른 의사결정을 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 계산해 낸다.
  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 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 첫 번째 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 테이블 간의 조인일 때 어떤 테이블을 먼저 조인할지에 대한 순서와 각각의 조인에 있어서 중첩 루프 조인, 합병 조인, 해시 조인 등의 다양한 방법 중 어떤 것을 사용할지가 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.

1.3. 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 적용되지 않아야 함을 의미한다. Tibero에서는 이를 위하여 undo 데이터를 사용한다.

- Consistency

  트랜잭션이 데이터베이스의 Consistency를 깨뜨리는 일은 여러 방면에서 생겨날 수 있다. 간단한 예는 테이블과 인덱스 간에 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 접근할 때는 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 같은 데이터에 접근한다고 해도 단지 기다림으로써 이를 해결한다.

- Durability

  Tibero에서는 이를 위하여 Redo 로그와 write-ahead logging 기법을 사용한다.

트랜잭션이 커밋할 때에 해당 Redo 로그가 디스크에 기록되어 트랜잭션의 영속성을 보장해 준다. 또한 블록이 디스크에 내려가기 전에 항상 Redo 로그가 먼저 내려가서 데이터베이스 전체가 일관성을 지니게 한다.

1.4. Row level locking

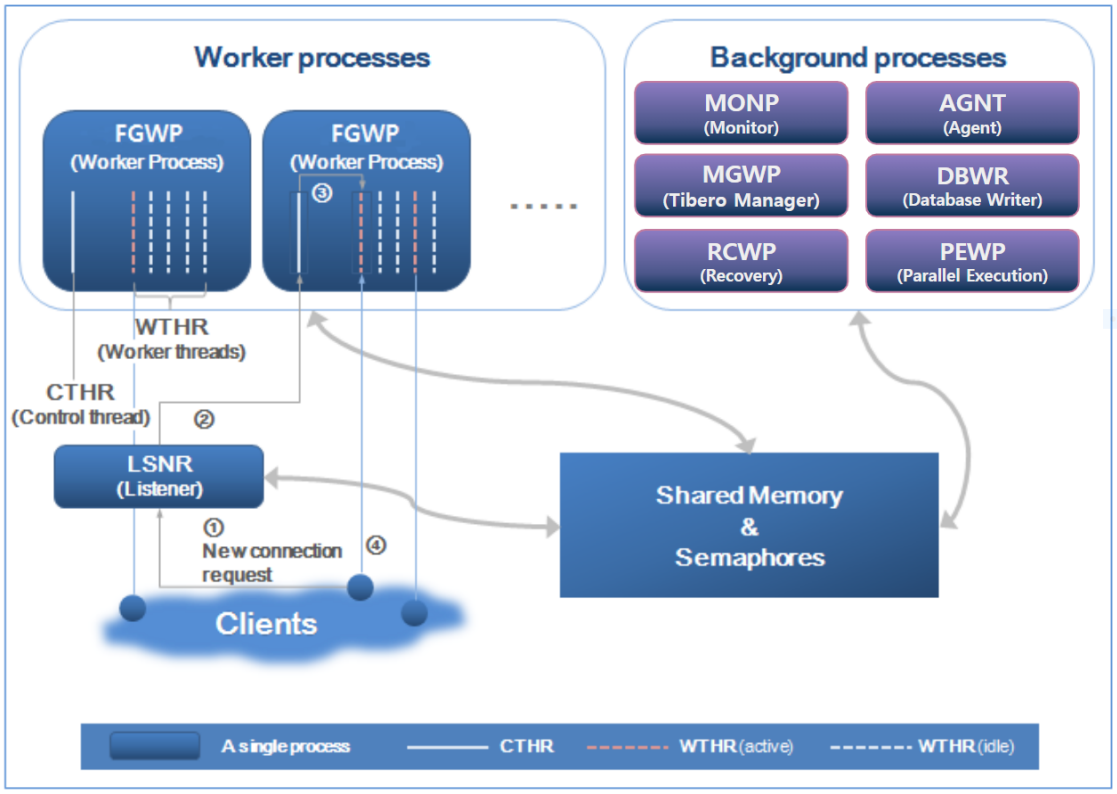
Tibero는 fine-grained lock control을 보장해 주기 위하여 row level locking을 사용한다. 즉, 데이터 최소 단위인 Row 단위의 locking을 통하여 최대한의 concurrency를 보장해 준다. 많은 Row들을 수정한다고 해도 테이블에 lock이 걸려서 concurrent한 DML이 수행되지 못하는 상황은 발생하지 않는다. 이러한 기법을 통해 OLTP 환경에서 더욱 강력한 성능을 발휘하고 있다.

1.5. 프로세스 구조

Tibero는 대규모 사용자 접속을 수용하는 다중 프로세스 및 다중 스레드 기반의 아키텍처를 갖추고 있다.

다음은 Tibero의 프로세스 구조를 나타내는 그림이다.

[그림 1.1] Tibero 프로세스 구조



Tibero의 프로세스는 크게 3가지로 구성된다.

- 리스너(Listener)
- 워커 프로세스(Worker Process 또는 Foreground Process)

- 백그라운드 프로세스(Background Process)

1.5.1. 리스너

리스너(Listener)는 클라이언트의 새로운 접속 요청을 받아 이를 유휴한 워커 프로세스에 할당한다. 즉, 클라이언트와 워커 프로세스간의 중계 역할을 담당하며 이는 별도의 실행 파일인 **tblistener**를 사용하여 작업을 수행한다. Tibero 6부터 MONP에 의해서 생성되며 외부에서 강제 종료하더라도 다시 생성된다.

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

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

1.5.2. 워커 프로세스

워커 프로세스(Worker Process)는 클라이언트와 실제로 통신을 하며 사용자의 요구 사항을 처리하는 프로세스이다. 이 프로세스의 개수는 `WTHR_PROC_CNT` 초기화 파라미터로 조절할 수 있으며, 일단 Tibero가 기동된 뒤에는 변경할 수 없다. 따라서 시스템 환경을 고려하여 적절한 값을 설정해야 한다.

Tibero 6부터 워커 프로세스는 용도에 따라 두 그룹으로 나눌 수 있다. 포어그라운드 워커 프로세스(Foreground Worker Process)는 리스너를 통해 들어온 온라인 요청을 처리하는 반면, 백그라운드 워커 프로세스(Background Worker Process)는 인터널 태스크(Internal Task)나 잡 스케줄러에 등록된 배치 작업을 수행한다. 그룹은 `MAX_BG_SESSION_COUNT` 초기화 파라미터로 조절할 수 있다.

참고

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

Tibero는 효율적인 리소스의 활용을 위해 **스레드**(Thread) 기반으로 작업을 수행한다. Tibero를 설치하면 기본적으로 하나의 워커 프로세스 안에는 1개의 컨트롤 스레드와 10개의 워커 스레드가 존재한다.

프로세스당 워커 스레드 개수는 `WTHR_PER_PROC` 초기화 파라미터로 조절할 수 있으며, `WTHR_PROC_CNT` 처럼 일단 Tibero가 기동된 뒤에는 변경할 수 없다. 따라서 시스템 환경을 고려하여 적절한 값을 설정해야 한다.

`WTHR_PROC_CNT`와 `WTHR_PER_PROC` 초기화 파라미터 값을 직접 바꾸는 것보다는 `MAX_SESSION_COUNT` 초기화 파라미터를 통해 서버에서 제공하는 최대 세션 개수를 지정할 것을 권장한다.

MAX_SESSION_COUNT 값에 따라 WTHR_PROC_CNT와 WTHR_PER_PROC 값이 자동으로 설정된다. 만약 WTHR_PROC_CNT와 WTHR_PER_PROC를 직접 설정할 경우 WTHR_PROC_CNT * WTHR_PER_PROC 값이 MAX_SESSION_COUNT 값과 같게 두 값을 설정해야 한다.

MAX_BG_SESSION_COUNT 값은 MAX_SESSION_COUNT 보다 작은 값을 설정해야하며 WTHR_PER_PROC 값의 배수가 되어야 한다.

워커 프로세스는 컨트롤 스레드와 워커 스레드를 통해 작업을 수행한다.

- **컨트롤 스레드**

  워커 프로세스마다 하나씩 존재하며 다음과 같은 역할을 담당한다.

  – Tibero가 기동될 때 초기화 파라미터에 설정된 수만큼 워커 스레드를 생성한다.

  – 클라이언트의 새로운 접속 요청이 오면 현재 유휴한 워커 스레드에 클라이언트의 접속을 할당한다.

  – 시그널 처리를 담당한다.

  – Tibero 6부터 I/O multiplexing을 지원하며 필요한 경우 워커 스레드 대신 메시지를 보내거나 받는 역할을 수행한다.

- **워커 스레드**

  워커 스레드는 클라이언트와 1:1로 통신하며, 클라이언트가 보내는 메시지를 받아 처리하고, 그 결과를 돌려준다. 주로 SQL 파싱, 최적화 수행 등 DBMS가 하는 작업 대부분이 워커 프로세스에서 일어난다.

  그리고 워커 스레드는 하나의 클라이언트와 접속하므로 Tibero에 동시 접속이 가능한 클라이언트 수는 WTHR_PROC_CNT * WTHR_PER_PROC이다. Tibero는 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_PROC를 연산한 값과 같다.

  워커 스레드는 클라이언트와 접속이 끊긴다고 해도 없어지지 않으며, Tibero가 기동될 때 생성된 이후부터 종료할 때까지 계속 존재하게 된다. 이러한 구조에서는 클라이언트와 접속을 빈번하게 발생시키더라도 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 워커 스레드가 동시에 작업을 수행하려고 할 때 심한 경우 운영체제에 과도한 부하를 일으켜 시스템 성능이 크게 저하될 수 있다. 그러므로 대규모 시스템을 구축할 경우에는 Tibero와 클라이언트의 애플리케이션 프로그램 사이에 미들웨어를 설치하여 3-Tier 구조로 시스템을 구축할 것을 권장한다.

1.5.3. 백그라운드 프로세스

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

백그라운드 프로세스에 속해 있는 프로세스는 다음과 같다.

- Monitor Process(MONP: 감시 프로세스)

  Tibero 6부터 영문 약자가 스레드에서 프로세스로 변경되었으며 실제로 하나의 독립된 프로세스이다. Tibero가 기동할 때 최초로 생성되며 Tibero가 종료하면 맨 마지막에 프로세스를 끝마친다.

  Tibero가 기동할 때 리스너를 포함한 다른 프로세스를 생성하거나 주기적으로 각 프로세스의 상태를 점검하는 역할을 담당한다. 또한 교착 상태(deadlock)도 검사한다.

- Tibero 매니저 프로세스(MGWP)

  시스템을 관리하기 위한 용도의 프로세스이다. 관리자의 접속 요청을 받아 이를 시스템 관리 용도로 예약된 워커 스레드에 접속을 할당한다. 기본적으로 워커 프로세스와 동일한 역할을 수행하지만 리스너를 거치지 않고 스페셜 포트를 통해 직접 접속을 처리한다. SYS 계정만 접속이 허용된다.

- Agent Process(AGNT: 에이전트 프로세스)

  시스템 유지를 위해 주기적으로 처리해야 하는 Tibero 내부의 작업을 담당한다.

  Tibero 4 SP1까지는 시퀀스 캐시(sequence cache)의 값을 디스크에 저장하는 작업도 담당했으나, Tibero 5 이후로 각 워커 스레드가 담당하는 것으로 변경되었다. 이전에는 SEQW라는 명칭을 사용했으나 Tibero 6부터 AGNT로 명칭이 변경되었다. 시퀀스를 사용하는 방법은 "Tibero 관리자 안내서"의 "4.7.1. 시퀀스 생성, 변경, 제거"를 참고한다.

  Tibero 6부터 다중 스레드(Multi-threaded) 기반 구조로 동작하며, 서로 다른 용도의 업무를 스레드별로 나누어 수행한다.

- DataBase Wirte Process(DBWR: 데이터베이스 쓰기 프로세스)

  데이터베이스에서 변경된 내용을 디스크에 기록하는 일과 연관된 스레드들이 모여 있는 프로세스이다. 사용자가 변경한 블록을 디스크에 주기적으로 기록하는 스레드, Redo 로그를 디스크에 기록하는 스레드, 이 두 스레드를 통해 데이터베이스의 체크포인트 과정을 관할하는 체크포인트 스레드 등이 이 프로세스에 포함되어 있다.

- Recovery Worker Process(RCWP: 리커버리 워커 프로세스)

  리커버리와 백업을 담당하는 프로세스이다. 부팅과정에서 NOMOUNT 이후의 부팅 단계를 올리는 역할을 수행하고 리커버리가 필요한 상황인지 여부를 판단하여 필요할 경우 리두로그를 읽어서 리커버리를 진행한다. tbrmgr을 이용하여 백업 작업을 하거나 백업본을 이용하여 미디어 리커버리를 진행하는 경우에도 이 프로세스에서 진행한다.

- Parallel Execution Worker Process(PEWP: Parallel Execution 워커 프로세스)

Parallel Execution Process(이하 PEP)는 PE 수행을 위해 도입된 PE 전용 프로세스이다. PE SQL을 처리할 때에 locality를 극대화하기 위해서 WTHR들을 하나의 PEP에서 할당한다. 또한 일반적인 클라이언트 세션을 위한 WTHR과 분리되어 모니터링 및 관리가 용이하다.

1.6. 디렉터리 구조

Tibero가 설치되면 다음과 같은 디렉터리가 생성된다.

```
$TB_HOME
    +- bin
    |   |
    |   +- update
    |
    +- client
    |   |
    |   +- bin
    |   +- config
    |   +- include
    |   +- lib
    |   |   |
    |   |   +- jar
    |   |   +- php
    |   +- ssl
    |   |   |
    |   |   +- misc
    |   +- epa
    |       |
    |       +- java
    |          |
    |          +- config
    |          +- lib
    |
    +- config
    |
    +- database
    |   +- $TB_SID
    |       |
    |       +- java
    |
    +- instance
    |   |
    |   +- $TB_SID
    |       |
    |       +- audit
    |       +- dump
    |       |  |
```

```
|       |  +- act
|       |  +- diag
|       |  +- tracedump
|       +- log
|       |  +- dlog
|       |  +- ilog
|       |  +- lsnr
|       |  +- slog
|       |  +- sqltrace
|       +- path
|
+- lib
|
+- license
|  |
|  +- oss_licenses
|
+- nls
|  |
|  +- zoneinfo
|
+- scripts
     |
     +- pkg
```

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

Tibero에서 사용하는 기본 디렉터리는 다음과 같다.

bin

Tibero의 실행 파일과 서버 관리를 위한 유틸리티가 위치한 디렉터리이다. 이 디렉터리에 속한 파일 중에서 tbsvr과 tblistener는 Tibero를 구성하는 실행 파일이며, tbboot와 tbdown은 각각 Tibero를 기동하고 종료하는 역할을 담당한다. tbsvr과 tblistener 실행 파일은 반드시 tbboot 명령어를 이용하여 실행되어야 하며, 절대로 직접 실행해서는 안 된다.

client

다음은 하위 디렉터리에 대한 설명이다.

하위 디렉터리	설명
bin	Tibero의 클라이언트 실행 파일이 있는 디렉터리이다. 이 디렉터리에는 다음과 같은 유틸리티가 있다. – tbSQL : 기본적인 클라이언트 프로그램으로 사용자가 직접 SQL 질의를 하고 그 결과를 확인할 수 있다.

하위 디렉터리	설명
	– T-Up : 다른 데이터베이스에서 Tibero로의 호환성 평가와 마이그레이션을 지원한다. – tbExport : 논리적 백업이나 데이터베이스 간에 데이터 이동을 위해 데이터베이스의 내용을 외부 파일로 저장한다. – tbImport : 외부 파일에 저장된 내용을 데이터베이스로 가져온다. – tbLoader : 대량의 데이터를 데이터베이스로 한꺼번에 읽어 들인다. – tbpc : C 언어로 작성된 프로그램 안에서 내장 SQL(Embedded SQL)을 사용하는 프로그램을 개발할 때 이를 C 프로그램으로 변환한다. 이렇게 변환된 프로그램을 C 컴파일러를 통해 컴파일할 수 있도록 도와주는 역할도 담당한다. 유틸리티에 대한 내용은 "Tibero 유틸리티 안내서"를 참고한다. 단, **tbpc** 유틸리티는 "Tibero tbESQL/C 안내서"를 참고한다.
config	Tibero의 클라이언트 프로그램을 실행하기 위한 설정 파일이 위치하는 디렉터리이다.
include	Tibero의 클라이언트 프로그램을 작성할 때 필요한 헤더 파일이 위치하는 디렉터리이다.
lib	Tibero의 클라이언트 프로그램을 작성할 때 필요한 라이브러리 파일이 위치하는 디렉터리이다. 자세한 내용은 "Tibero 애플리케이션 개발자 안내서"와 "TiberotbESQL/C 안내서"를 참고한다.
ssl	서버 보안을 위한 인증서와 개인 키를 저장하는 디렉터리이다.
epa	External Procedure와 관련된 설정 파일과 로그 파일이 있는 디렉터리이다. 이에 대한 자세한 내용은 "Tibero External Procedure 안내서"를 참고한다.

config

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

database

다음은 하위 디렉터리에 대한 설명이다.

– $TB_SID

Tibero의 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.

- $TB_SID/java

  JAVA_CLASS_PATH가 정의되지 않은 경우 Java EPA Class File이 저장되는 디렉터리이다.

instance

다음은 하위 디렉터리에 대한 설명이다.

- $TB_SID/audit

  데이터베이스 사용자가 시스템 특권 또는 스키마 객체 특권을 사용하는 것을 감시(AUDIT)한 내용을 기록한 파일이 저장되는 디렉터리이다.

- $TB_SID/log

  Tibero의 시스템 로그 파일(slog)과 DBMS 로그(dlog), Internal 로그(ilog), Listener 로그(lsnr), memlog 파일이 저장되는 디렉터리이다.

파일	설명
시스템 로그 파일(slog)	디버깅을 위한 파일이다. 서버가 하는 중요한 일이 기록되는 파일이며, 서버 성능이 저하되는 원인을 찾거나 Tibero 자체의 버그를 해결하는 데 사용될 수 있다.
DBMS 로그 파일(dlog)	시스템 로그 파일에 기록되는 정보보다 좀 더 중요한 정보가 기록되는 파일이며, 서버 기동 및 종류, DDL 문장의 수행 등이 기록되는 파일이다.
Internal 로그 파일(ilog)	스레드별로 설정된 이벤트에 대한 시스템 로그가 기록되는 파일이며, Internal 로그를 보려면 tbiv를 이용해야 한다.
Listener 로그 파일(lsnr)	Listener의 디버깅을 위한 파일이다. 리스너에서 일어난 중요한 일이 기록되는 파일이며, 리스너의 버그를 해결하는 데 사용될 수 있다.

시스템 로그 파일, DBMS 로그 파일, Internal 로그 파일, Listener 로그 파일은 데이터베이스를 사용할수록 계속 누적되어 저장된다. 또한 전체 디렉터리의 최대 크기를 지정할 수 있으며, Tibero는 그 지정된 크기를 넘어가지 않도록 오래된 파일을 삭제한다.

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.
SLOG_FILE_SIZE	시스템 로그 파일 하나의 최대 크기를 설정한다.

초기화 파라미터	설명
SLOG_TOTAL_SIZE_LIMIT	시스템 로그 파일이 저장된 디렉터리의 최대 크기를 설정한다.
ILOG_FILE_SIZE	Internal 로그 파일 하나의 최대 크기를 설정한다.
ILOG_TOTAL_SIZE_LIMIT	Internal 로그 파일이 저장된 디렉터리의 최대 크기를 설정한다.
LSNR_LOG_FILE_SIZE	Listener 로그 파일 하나의 최대 크기를 설정한다.
LSNR_LOG_TOTAL_SIZE_LIMIT	Listener 로그 파일이 저장된 디렉터리의 최대 크기를 설정한다.

– $TB_SID/dump

Tibero의 DDL 또는 실행 중 에러에 의해 발생하는 덤프 파일이 저장되는 디렉터리이다.

하위 디렉터리	설명
act	스레드 액티비티 모니터에 의한 정보가 남는 디렉터리이다.
diag	TAC의 diag 기능을 사용하는 경우 로그 및 덤프가 남는 디렉터리이다.
tracedump	SQL 수행 중 에러가 발생하는 경우 디버깅을 위해 sql, psm 정보를 수집해서 남긴다. 이외에도 DDL dump 명령을 통해 남긴 덤프가 남는 디렉터리이다.

– $TB_SID/path

Tibero의 프로세스 간에 통신을 위한 소켓 파일이 있는 디렉터리이다. Tibero가 운영 중일 때 이 위치에 존재하는 파일을 읽거나 수정해서는 안 된다.

lib

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

license

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

다음은 하위 디렉터리에 대한 설명이다.

하위 디렉터리	설명
oss_licenses	반드시 준수해야 하는 오픈소스 라이선스의 대한 정보를 확인할 수 있는 디렉터리이다.

nls

다음은 하위 디렉터리에 대한 설명이다.

하위 디렉터리	설명
zoneinfo	Tibero에서 사용하는 시간대 파일이 있는 디렉터리이다.

scripts

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

다음은 하위 디렉터리에 대한 설명이다.

하위 디렉터리	설명
pkg	Tibero에서 사용하는 패키지의 생성문이 저장되는 디렉터리이다.

제2장 관리의 기본

본 장에서는 DBA가 Tibero를 관리하기에 앞서 기본적으로 알아야 할 사항을 설명한다.

2.1. 사용자 정의

Tibero를 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? – CPU와 메모리 사용량, 클라이언트의 응답 시간은 어떠한가? 문제가 발생하면 H/W를 확장하거나 데이터베이스를 튜닝한다.

항목	설명
데이터베이스	– 자주 사용되는 스키마 객체는 무엇인가? – 자주 사용되는 SQL 문장은 무엇인가? – 자주 사용되는 SQL 문장이 효율적으로 실행되고 있는가?
유지보수	– Tibero의 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 백업하고, 문제가 발생할 경우 복구한다.

참고

Tibero는 DBA가 위와 같은 항목을 효율적이고 유연하게 관리할 수 있도록 유틸리티를 제공하고 있다. 자세한 내용은 "Tibero 유틸리티 안내서"를 참고한다.

2.1.2. SYS

Tibero를 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.

Tibero의 데이터 사전, 기반 테이블, 뷰 등은 모두 SYS 사용자의 스키마에 저장된다. 특히 기반 테이블, 뷰는 Tibero가 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. SYS 사용자는 하나의 스키마를 가진다. 스키마의 이름은 사용자 이름인 SYS와 동일하다.
패스워드	패스워드는 Tibero를 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 개발자(Application Developers)는 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.

애플리케이션 프로그램 개발자의 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 사용자(Database Users)는 Tibero를 사용하는 DBA, 업무 분석가, 애플리케이션 프로그램 개발자, 사용자 모두를 뜻한다.

2.2. 설치 환경

Tibero를 정상적으로 설치했다면 시스템에 다음과 같은 환경변수가 설정된다.

환경변수	설명
$TB_HOME	Tibero가 설치된 홈 디렉터리이다. Tibero 서버, 클라이언트 라이브러리, 다양한 부가 기능을 수행하는 파일이 설치된다.
$TB_SID	한 머신에서 Tibero의 인스턴스를 여러 개로 운영할 때 필요한 서비스 ID이다.

환경변수	설명
$PATH	파일 시스템을 통해 특정 파일에 접근하기 위한 디렉터리 경로를 설정한다.

환경변수를 제대로 설정하지 않으면 Tibero를 사용할 수 없다. 따라서 환경변수를 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 환경변수를 확인하는 방법은 다음과 같다.

- $TB_HOME

```
$ echo $TB_HOME
/home/tibero/tibero7
```

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

- $TB_SID

```
$ echo $TB_SID
Tb7
```

서비스 ID는 데이터베이스의 이름과 동일하게 설정할 것을 권장한다.

- $PATH

```
$ echo $PATH
...:/home/tibero/tibero7/bin:/home/tibero/tibero7/client/bin:...
```

$PATH는 다음의 디렉터리를 포함하고 있어야 한다.

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

2.3. tbSQL 유틸리티 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 7

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

SQL>
```

tbSQL 유틸리티가 정상적으로 실행되면 이처럼 **SQL 프롬프트**가 나타난다. 이 프롬프트에서 데이터베이스 사용자는 SQL 문장을 실행할 수 있다. 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 7

TmaxData Corporation Copyright (c) 2008-. 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	패스워드로 대소문자를 구분하지 않는다. 단, 작은따옴표(' ')에 패스워드를 입력하는 경우는 예외이다.

tbSQL 유틸리티 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 다음과 같다. 대괄호([ ])에 포함된 내용은 입력하지 않아도 명령어를 실행할 수 있다.

명령어	설명
!	운영체제의 명령어를 실행한다. HOST 명령어와 동일하다.
%	히스토리 버퍼에 저장된 명령어를 재수행한다.
@, @@	스크립트를 실행한다. START 명령어와 동일하다.
/	현재 SQL 버퍼 내의 SQL 문장 또는 tbPSM 프로그램을 실행한다. RUN 명령어와 동일하다.
ACC[EPT]	사용자의 입력을 받아 바인드 변수의 속성을 설정한다.
ARCHIVE LOG	Redo 로그 파일 정보를 출력한다.
C[HANGE]	SQL 버퍼의 현재 라인에서 패턴 문자를 찾아 주어진 문자로 변환한다.
CL[EAR]	설정된 옵션을 초기화하거나 지우는 명령어이다.
COL[UMN]	컬럼의 출력 속성을 설정한다.
CONN[ECT]	특정 사용자 ID로 데이터베이스에 접속한다.
DEF[INE]	바인드 변수를 정의하거나 출력한다.
DEL	SQL 버퍼에 저장된 라인을 지우는 명령어이다.
DESC[RIBE]	지정된 객체의 컬럼 정보를 출력한다.
DISC[ONNECT]	현재 데이터베이스로부터 접속을 해제한다.
ED[IT]	특정 파일 또는 SQL 버퍼의 내용을 외부 편집기를 이용하여 편집한다.
EXEC[UTE]	단일 tbPSM 문장을 수행한다.
EXIT	tbSQL 유틸리티를 종료한다. QUIT 명령어와 동일하다.
EXP[ORT]	SELECT 문장의 수행 결과나 테이블 데이터를 파일로 출력한다.
H[ELP]	도움말을 출력한다.
HIS[TORY]	실행한 명령어의 히스토리를 출력한다.
HO[ST]	운영체제 명령어를 실행한다. ! 명령어와 동일하다.
I[NPUT]	SQL 버퍼의 현재 라인 뒤에 새로운 라인을 추가한다.
L[IST]	SQL 버퍼 내의 특정 내용을 출력한다.
LOAD[FILE]	Tibero의 테이블을 Oracle의 SQL*Loader 툴이 인식할 수 있는 형식으로 저장한다.
LOOP	단일 명령어를 무한 반복 수행한다.

명령어	설명
LS	현재 사용자가 생성한 데이터베이스 객체를 출력한다.
PASSW[ORD]	사용자 패스워드를 변경한다.
PAU[SE]	사용자가 <Enter> 키를 누를 때까지 실행을 멈춘다.
PING	특정 데이터베이스에 대해 접속 가능한 상태인지를 출력한다.
PRI[NT]	사용자가 정의한 바인드 변수의 값을 출력한다.
PRO[MPT]	사용자가 정의한 SQL 문장이나 빈 라인을 그대로 화면에 출력한다.
Q[UIT]	tbSQL 유틸리티를 종료한다. EXIT 명령어와 동일하다.
REST[ORE]	선택한 정보를 파일로부터 복원한다.
R[UN]	현재 SQL 버퍼 내의 SQL 문장이나 tbPSM 프로그램을 실행한다. / 명령어와 동일하다.
SAVE	선택한 정보를 파일에 저장한다.
SET	tbSQL 유틸리티의 시스템 변수를 설정한다.
SHO[W]	tbSQL 유틸리티의 시스템 변수를 출력한다.
SPO[OL]	화면에 출력되는 내용을 모두 외부 파일에 저장하는 과정을 시작하거나 종료한다.
STA[RT]	스크립트 파일을 실행한다. @ 명령어와 동일하다.
TBDOWN	Tibero를 종료한다.
UNDEF[INE]	하나 이상의 바인드 변수를 삭제한다.
VAR[IABLE]	사용자가 정의한 바인드 변수를 선언한다.
WHENEVER	에러가 발생한 경우의 동작을 정의한다.

참고

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

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

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

[예 2.3] LS 명령어의 실행

```
SQL> LS

NAME                           SUBNAME       OBJECT_TYPE
------------------------------ ------------- --------------------
SYS_CON100                                   INDEX
SYS_CON400                                   INDEX
SYS_CON700                                   INDEX
```

```
_DD_CCOL_IDX1                                   INDEX
        ......중간 생략......
UTL_RAW                                         PACKAGE
DBMS_STATS                                      PACKAGE BODY
TB_HIDDEN2                                      PACKAGE 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 Tibero가 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 **테이블 스페이스**를 조회하는 예이다.

[예 2.5] LS 명령어의 실행 - 테이블 스페이스 조회

```
SQL> LS TABLESPACE

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

SQL 문장의 실행

다음은 **V$SESSION**이라는 뷰를 이용하여 TYPE 컬럼이 **'WTHR'**인 데이터를 조회하는 SQL 문장을 실행하는 예이다. V$SESSION은 각각의 세션 ID를 나열하는 뷰이다.

[예 2.6] SQL 문장의 실행 (1)

```
SQL> SELECT SID, STATUS, TYPE, WTHR_ID FROM V$SESSION WHERE TYPE = 'WTHR';

       SID STATUS                           TYPE        WTHR_ID
---------- -------------------------------- -------- ----------
        10 ACTIVE                           WTHR              1
        13 RUNNING                          WTHR              1

2 rows selected.
```

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 언어이다. 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 포함되는 COMMIT, ROLLBACK 등의 SQL 명령어 일부를 트랜잭션 및 세션 언어로 재구성하였다. 따라서 전체 DCL에 대한 내용은 관련 문서를 참고한다.

DML에 포함되는 **SELECT 문**을 다음과 같이 실행한다.

[예 2.7] SQL 문장의 실행 (2)

```
SQL> select SID, STATUS, TYPE, WTHR_ID from v$session where type = 'WTHR';
......[예 2.6]의 실행 결과와 동일......

SQL> select SID, STATUS, TYPE, WTHR_ID from V$SESSION where type = 'WTHR';
......[예 2.6]의 실행 결과와 동일......
```

```
SQL> select SID, STATUS, TYPE, WTHR_ID From v$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 의미도 같다.

그러나 다음과 같은 SQL 문장을 실행하면 그 결과와 의미는 달라진다.

```
SQL> SELECT SID, STATUS, TYPE, WTHR_ID FROM V$SESSION WHERE TYPE = 'wthr';

0 row selected.
```

2.4. 사용자 및 테이블 생성

본 절에서는 데이터베이스를 사용하기 위해 사용자와 테이블을 생성하는 방법을 설명한다.

사용자의 생성

사용자를 생성하려면 **CREATE USER 문**을 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.
SQL>
```

[예 2.8]의 과정을 모두 완료하면 Tibero에 'ADMIN'이라는 새로운 사용자가 추가된다.

테이블의 생성

테이블을 생성하려면 **CREATE TABLE 문**을 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.
```

Tibero는 **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 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 Tibero에 접속한다.

```
$ tbsql ADMIN/password123

tbSQL 7

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

Connected to Tibero.

SQL> SELECT * FROM "PRODUCT";

0 row selected.
```

위 예를 보면 [예 2.9]에서 처럼 4개의 로우 데이터가 출력되지 않는 것을 알 수 있다. 그 이유는 Tibero는 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 commit 명령을 실행해야 한다.

commit 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 진행 중인 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 “5.4. 역할”을 참고한다.

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

```
$ tbsql SYS/tibero

tbSQL 7

TmaxData Corporation Copyright (c) 2008-. 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 발생하고, ADMIN 사용자는 더 이상 데이터베이스에 접속할 수 없게 된다.

2.5. 기동과 종료

본 절에서는 Tibero를 기동하고 종료할 때 사용하는 명령어와 이를 사용하는 방법을 설명한다.

2.5.1. tbboot

tbboot는 Tibero가 설치된 머신에서 실행해야 한다. 또한 “2.2. 설치 환경”에서 설명했듯이 tbboot를 실행하기 전에 반드시 환경변수가 제대로 설정되어 있어야 한다. 이 명령어와 관련된 환경변수는 $TB_HOME과 $TB_SID이다. 또한 tbboot는 실행 파일을 실행할 수 있는 권한이 있는 사용자라면 어느 누구든 Tibero를 기동할 수 있다. 따라서 이는 보안 문제가 발생할 수 있어 정책상 Tibero를 설치한 사용자만 Tibero의 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 tiberotibero 4096 Dec 28 18:12 ./
drwxr-xr-x 13 tiberotibero 4096 Dec 20 11:59 ../

        ...... 중간 생략......

-rwx------  1 ... tbboot*
-rwx------  1 ... tbctl*
-rwx------  1 ... tbsvr*
lrwxrwxrwx    1 ... tblistener*
lrwxrwxrwx    1 ... tbdown -> $TB_HOME/bin/tbsvr*
```

tbboot의 사용법은 다음과 같다.

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

옵션	설명
	옵션이 없는 경우 Tibero를 부트 모드(bootmode) 중 NORMAL로 기동하는 옵션이다.
-h	tbboot를 사용하기 위한 간단한 도움말을 보여주는 옵션이다.
-v	Tibero의 버전 정보를 보여주는 옵션이다.
-C	Tibero에서 사용 가능한 character set 정보와 nls_date_lang 정보를 보여주는 옵션이다.
-c	Tibero가 replication mode로 설정되어 있을 경우 replication mode를 사용하지 않는 옵션이다.
-t	Tibero 서버를 기동할 수 있는 옵션이다. 이 옵션은 생략이 가능하다. Tibero에서는 tbboot에 부트 모드(bootmode)를 제공한다. 각 모드에 대한 자세한 내용은 해당 절의 내용을 참고한다. – NORMAL : 정상적으로 데이터베이스의 모든 기능을 사용할 수 있는 모드이다. – NOMOUNT : Tibero의 프로세스만 기동시키는 모드이다. – MOUNT : 미디어 복구를 위해 사용하는 모드이다. – RESETLOGS : Tibero 서버를 기동하는 과정에서 로그 파일을 초기화하며 미디어 복구 이후에 사용하는 모드이다.

옵션	설명
-w	Tibero가 보안 지갑을 열고 기동하는 옵션이다. 기동할 때 안내에 따라 보안 지갑의 패스워드를 입력 해야한다. 입력된 보안 지갑의 패스워드가 일치하는 경우 NORMAL 모드로 기동하며, 불일치 하는 경우 MOUNT 모드로 기동한다. MOUNT 모드로 기동하는 경우에는 해당 옵션으로 보안 지갑을 열 수 없다.

NORMAL

정상적으로 데이터베이스의 모든 기능을 사용할 수 있는 모드이다.

사용 예는 다음과 같다.

```
$ tbboot NORMAL
listener port = 8629

Tibero 7

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

Tibero instance started suspended at NORMAL mode.
```

참고

데이터베이스를 비정상적으로 종료한 경우 Tibero가 기동할 때 자동으로 파손 복구(crash recovery)를 실행한다. 자세한 내용은 “11.3.2. 파손 복구”를 참고한다.

NOMOUNT

Tibero의 프로세스만 기동시키는 모드이다. 일반적으로는 이 모드를 사용하는 경우는 거의 없고 Tibero가 기동한 다음에 CREATE DATABASE 문을 이용하여 데이터베이스를 생성하는 것밖에 없다.

사용 예는 다음과 같다.

```
$ tbboot NOMOUNT
listener port = 8629

Tibero 7

TmaxData Corporation Copyright (c) 2008-. 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 7

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

Tibero instance started suspended at MOUNT mode.
```

RESETLOGS

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

사용 예는 다음과 같다.

```
$ tbboot RESETLOGS
listener port = 8629

Tibero 7

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

Tibero instance started suspended at NORMAL mode.
```

참고

RESETLOGS 부트 모드는 직전에 데이터베이스가 비정상적으로 종료되었을 경우에는 사용할 수 없으며, 일단 Tibero가 기동되면 NORMAL 모드와 동일하게 동작한다. 자세한 내용은 “11.3.3. 미디어 복구”를 참고한다.

2.5.2. tbdown

tbdown은 현재 동작 중인 Tibero를 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 Tibero를 정상 모드로 종료하는 옵션이다.
-h	tbdown을 사용하기 위한 간단한 도움말을 보여주는 옵션이다.
-t	Tibero 서버를 종료할 수 있는 옵션이다. 이 옵션은 생략이 가능하다. Tibero에서는 tbdown에 다운 모드(downmode)를 제공한다. 각 모드에 대한 자세한 내용은 해당 절의 내용을 참고한다. – NORMAL : 일반적인 종료 모드이다. – POST_TX : 모든 트랜잭션이 끝날 때까지 기다리고 나서 Tibero를 종료하는 모드이다. – IMMEDIATE : 현재 수행 중인 모든 작업을 강제로 중단시키며, 진행 중인 모든 트랜잭션을 롤백하고 Tibero를 종료하는 모드이다. – ABORT : Tibero의 프로세스를 강제로 종료하는 모드이다. – SWITCHOVER : Standby DB와 Primary DB를 동기화시킨 후 Primary DB를 NORMAL 모드처럼 종료하는 모드이다. – ABNORMAL : Tibero 서버에 접속하지 않고, 서버 프로세스를 무조건 강제로 종료시키는 모드이다.
clean	Tibero 서버가 비정상 종료된 상태에서 운영 중에 사용하였던 공유 메모리나 세마포어 자원들을 해제하는 옵션이다. Tibero 서버가 운영 중일 때는 사용할 수 없다.

참고

Tibero 서버가 kill과 같은 시스템 내부 명령어에 의해 비정상적으로 종료된 경우, 운영 중에 사용하였던 공유 메모리나 세마포어 자원들이 해제되지 않을 수 있다. 이러한 경우 재부팅에 실패하고, 관리자는 에러 메시지를 통해서 서버가 비정상 종료되었다는 것을 인지하게 된다. 이런 상황에서는 서버를 재부팅하기 전에 반드시 tbdown clean으로 이전에 사용한 자원을 해제시켜야 한다.

서버가 비정상적으로 종료되었더라도 초기화 파라미터 BOOT_WITH_AUTO_DOWN_CLEAN을 Y로 설정하면 자동으로 이전에 사용한 자원을 해제하고 부팅시킬 수 있다. 이 경우 콘솔에 "tbdown clean" is executed automatically. 메시지가 출력된다. Tibero 7의 경우 기본적으로 Y로 설정되어 있으므로 위 메시지를 확인한 경우 외부 프로그램 또는 사용자가 프로세스를 강제로 kill 하였는지 확인해볼 필요가 있다.

tbdown 명령을 이용하여Tibero 서버를 종료시킬 때 ABNORMAL 모드를 제외한 모든 다운 모드(downmode)에서는 tbdown 프로세스가 서버에 직접 접속하여 세션을 맺고 명령을 내려 서버를 종료시킨다.

tbdown 프로세스가 서버에 접속하기 위한 호스트 네임은 locahost로 고정되어 있으며 포트 번호는 초기화 파라미터 '_LSNR_SPECIAL_PORT'와 같다. _LSNR_SPECIAL_PORT의 기본값은 LISTENER_PORT + 1이다. 다만 직접적으로 _LSNR_SPECIAL_PORT로 접속하려 할 때에는 호스트 네임에 localhost는 사용할 수 없으며 다른 호스트 네임이나 IP를 명시적으로 사용해야 한다.

리스너는 tbdown프로세스의 접속 요청을 받아서 일반 워킹 프로세스가 아닌 전용 워킹 프로세스의 워킹 스레드에게 접속을 할당한다. tbdown 프로세스가 ABNORMAL 모드로 Tibero 서버를 종료시킬 때는 서버에 직접 접속하지 않고 OS의 강제 종료 시그널을 사용하여 서버 프로세스들을 강제로 종료시킨다.

참고

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

2. 이 장에서 설명하는 tbdown 명령과 다운 모드는 SYS 사용자로 접속한 tbSQL의 SQL 프롬프트에서도 가능하다. 이를 통하여 Tibero 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 서비스를 중지시킬 경우 기본적으로 IMMEDIATE 모드로 Tibero 서버가 종료된다. 하지만 사용자가 원한다면 초기화 파라미터 'SERVICE_CONTROL_STOP_DOWN'을 이용하여 종료 모드를 설정할 수 있다.

NORMAL

일반적인 종료 모드이다. Tibero에 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 Tibero를 종료하는 모드이다.

POST_TX는 Tibero에 SYS 사용자로 접속한 다음 모든 **트랜잭션**이 끝날 때까지 기다린다. 그 다음 Tibero를 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 또는 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 Tibero가 종료되었음을 알게 된다.

예를 들면 다음과 같다.

```
$ tbsql admin/password123

tbSQL 7

TmaxData Corporation Copyright (c) 2008-. 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-2131: Generic I/O error.
        ...서버가 종료되었으므로 tbSQL 유틸리티의 프롬프트에서는 TBR-2131 에러가 발생된다.
```

IMMEDIATE

현재 수행 중인 모든 작업을 강제로 중단시키며, 진행 중인 모든 트랜잭션을 롤백하고 Tibero를 종료하는 모드이다. IMMEDIATE는 Tibero에 SYS 사용자로 접속한 다음 현재 수행 중인 모든 작업을 강제로 종료하고 진행 중이던 모든 트랜잭션을 롤백한 후 Tibero를 종료시킨다. 클라이언트에서 Tibero 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

Tibero의 프로세스를 강제로 종료하는 모드이다. ABORT는 Tibero에 SYS 사용자로 접속한 다음 Tibero의 MONP 프로세스가 모든 프로세스를 OS의 강제 종료 시그널을 전달하여 강제로 종료시키는 모드이다. 따라서 이 모드는 비상시에 사용하며 다음 번에 Tibero를 기동할 때 파손 복구 과정이 필요하다.

사용 예는 다음과 같다.

```
$ tbdown abort

Tibero instance terminated (ABORT mode).
```

ABORT는 Tibero가 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 Tibero를 기동할 때 파손 복구에 많은 시간이 걸릴 수도 있다.

ABORT는 다음과 같은 경우에만 제한적으로 사용할 것을 권장한다.

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

SWITCHOVER

SWITCHOVER는 Standby DB와 Primary DB를 동기화시킨 후 Primary DB를 NORMAL 모드처럼 종료하는 모드이다. 이와 관련된 자세한 내용은 “제13장 Tibero Standby Cluster”의 “13.7.1. Switchover”를 참고한다.

ABNORMAL

Tibero 서버에 접속하지 않고 서버 프로세스를 무조건 강제로 종료시키는 모드이다.

ABNORMAL은 Tibero 서버에 접속하지 않고 현재 Tibero 서버 상태와 상관없이 OS의 강제 종료 시그널을 사용하여 무조건 서버 프로세스를 강제로 종료시키는 모드이다. 따라서 이 모드는 비상 시에 사용하며, 다음번에 Tibero를 기동할 때 파손 복구 과정이 필요하다.

사용 예는 다음과 같다.

```
$ tbdown abnormal

Tibero instance terminated (ABNORMAL mode).
```

ABNORMAL는 Tibero가 강제로 종료시키므로 사용하던 시스템 리소스를 해제 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 Tibero를 기동할 때 파손 복구에 많은 시간이 걸릴 수도 있다.

ABNORMAL은 다음과 같은 경우에만 제한적으로 사용할 것을 권장한다.

- Tibero의 내부 에러로 인한 정상적인 종료가 불가능한 경우
- ABNORMAL 외 다른 종료 모드로 종료 명령을 내린 후 종료가 지연되고 있을 때 강제로 즉시 종료해야 하는 경우
- H/W나 OS 등의 외부적인 요인으로 인하여 문제가 발생하여 ABNORMAL 외 다른 종료 모드의 실행이 실패한 경우
- H/W에 문제가 발생하여 Tibero를 즉시 종료해야 하는 경우
- 해킹 등의 비상 상태가 발생하여 Tibero를 즉시 종료해야 하는 경우

2.6. Binary TIP 사용

Binary TIP(이하 BTIP)은 Tibero Initialization Parameter 값을 바이너리 파일 형태로 저장하는 기능이다. 기존의 TIP 파일과 차이점은 운영 중 변경한 파라미터의 값을 저장하여 다음 부팅할 때 반영할 수 있다.

BTIP의 생성은 다음의 DDL 구문을 실행한다.

```
SQL> CREATE BTIP FROM TIP;
      ...$TB_HOME/config 하위의 TIP 파일로부터 BTIP을
      ...$TB_HOME/config/$TB_SID.btip로 생성한다.

SQL> CREATE BTIP='PATH' FROM TIP;
      ...BTIP을 $TB_HOME/config 아닌 곳에 생성하고
      ...싶은 경우 BTIP 파일명의 절대 경로를 주면 된다.
      ...파일 확장명은 반드시 .btip이어야 한다.
```

BTIP 파일로 부팅하기 위해서는 $TB_HOME/config 하위에 $TB_SID.tip에 BTIP_FILE_PATH를 추가해야 한다. 이 파라미터는 BTIP 파일의 절대경로를 나타낸다. 기본적으로 부팅할 때 $TB_SID.tip 파일에 BTIP_FILE_PATH 파라미터가 설정되어 있다면, BTIP_FILE_PATH의 BTIP 파일에서 파라미터를 읽어온다. BTIP 파일을 사용하기 위해서는 $TB _SID.tip 파일에는 BTIP_FILE_PATH만 설정되어 있어야 한다.

사용 중인 TIP 파일의 정보는 다음처럼 조회할 수 있다.

```
SQL> SELECT TIP_FILE FROM V$INSTANCE;
TIP_FILE
--------------------------------------------------------------------------------
/home/tibero/work/7/config/t7.btip

...$TB_HOME/config 하위의 BTIP 파일을 사용 중이다.
```

운영 중 파라미터 동적 변경을 BTIP에 저장하려면 다음의 DDL 구문을 이용하여 반영하려고 하는 scope를 지정하면 된다.

```
SQL> ALTER SYSTEM SET PARAMETER1 = NEW_VALUE SCOPE BTIP ;
...PARAMETER1의 변경된 값 NEW_VALUE는 BTIP에만 반영하여
...다음 부팅부터 PARAMETER1의 값은 NEW_VALUE로 동작한다.
...DDL 구문 수행 후 PARAMETER1의 값은 OLD_VALUE이다.

SQL> ALTER SYSTEM SET PARAMETER1 = NEW_VALUE SCOPE MEMORY ;
...PARAMETER1의 변경된 값 NEW_VALUE는 운영 중인 시점에만 반영하고
...다음 부팅시 PARAMETER1의 값은 OLD_VALUE이다.

SQL> ALTER SYSTEM SET PARAMETER1 = NEW_VALUE SCOPE BOTH ;
...PARAMETER1의 변경된 값 NEW_VALUE는 운영 시점과 BTIP 모두 반영하며
...다음 부팅시 PARAMETER1의 값은 NEW_VALUE이다.
...DDL 구문 수행 후 PARAMETER1의 값도 NEW_VALUE이다.
```

BTIP에서 TIP 파일 생성하는 것은 운영 중인 서버가 BTIP을 사용 중일 때 가능하다.

```
SQL> CREATE TIP FROM BTIP;
```

제3장 파일과 데이터 관리

본 장에서는 Tibero의 파일과 데이터를 관리하는 방법을 설명한다.

3.1. 데이터 저장 구조

Tibero의 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 > 테이블 스페이스 > 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. 테이블 스페이스

테이블 스페이스는 논리적 저장 영역과 물리적 저장 영역에 공통적으로 포함된다. 논리적 저장 영역에는 Tibero의 모든 데이터가 저장되며, 물리적 저장 영역에는 데이터 파일이 하나 이상 저장된다. 테이블 스페이스는 논리적 저장 영역과 물리적 저장 영역을 연관시키기 위한 단위이다.

3.2.1. 테이블 스페이스 구성

테이블 스페이스는 크게 두 가지 구성으로 Tibero의 데이터를 저장한다.

테이블 스페이스의 논리적 구성

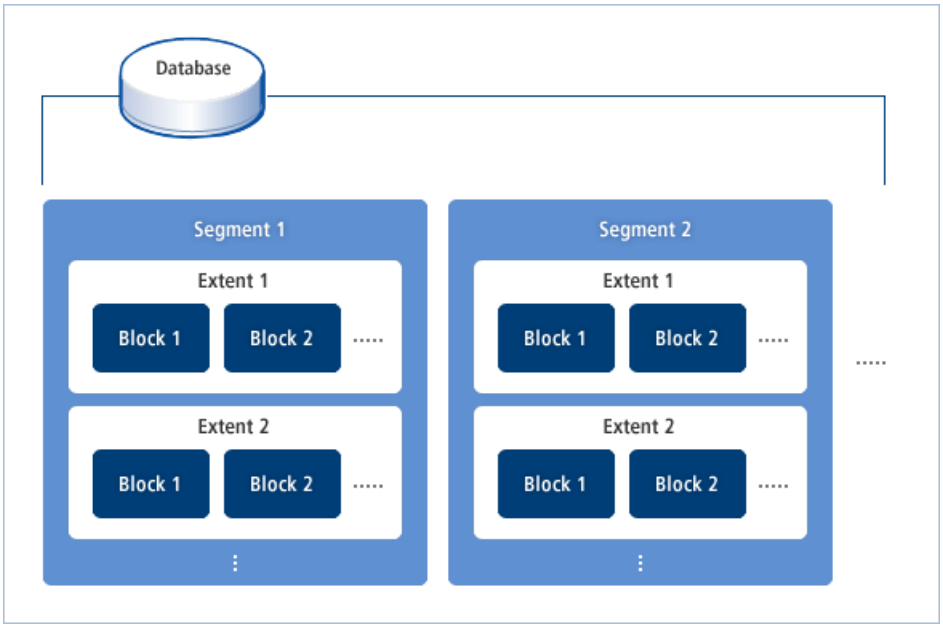
테이블 스페이스는 세그먼트(Segment), 익스텐트(Extent), 데이터 블록(Block)으로 구성된다.

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

구성요소	설명
익스텐트	연속된 데이터 블록의 집합이다. 세그먼트를 처음 만들거나 세그먼트의 저장 공간이 더 필요한 경우 Tibero는 테이블 스페이스에서 연속된 블록의 주소를 갖는 데이터 블록을 할당 받아 세그먼트에 추가한다.
데이터 블록	데이터베이스에서 사용하는 데이터의 최소 단위이다. Tibero는 데이터를 블록(Block) 단위로 저장하고 관리한다.

다음은 테이블 스페이스의 논리적 구성을 나타내는 그림이다.

[그림 3.1] 테이블 스페이스의 논리적 구성



참고

논리적 저장 영역을 관리하는 방법에 대한 자세한 내용은 “제4장 스키마 객체 관리”를 참고한다.

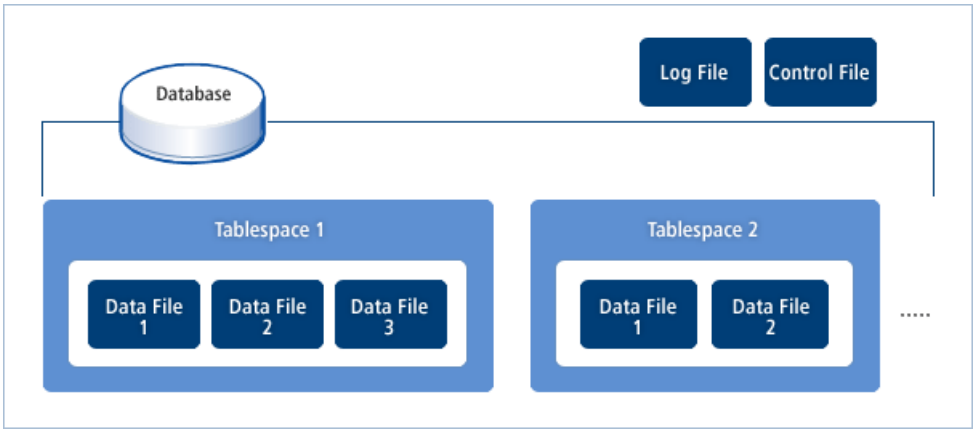
테이블 스페이스의 물리적 구성

테이블 스페이스는 물리적으로 여러 개의 데이터 파일로 구성된다. Tibero는 데이터 파일 외에도 컨트롤 파일과 로그 파일을 이용하여 데이터를 저장할 수 있다.

빈번하게 사용되는 두 테이블 스페이스(예: 테이블과 인덱스)는 물리적으로 서로 다른 디스크에 저장하는 것이 좋다. 왜냐하면 한 테이블 스페이스를 액세스하는 동안에 디스크의 헤드가 그 테이블 스페이스에 고정되어 있기 때문에 다른 테이블 스페이스를 액세스할 수 없다. 따라서 서로 다른 디스크에 각각의 테이블 스페이스를 저장하여 동시에 액세스하는 것이 데이터베이스 성능을 향상시키는 데 도움이 된다.

다음은 테이블 스페이스의 물리적 구성을 나타내는 그림이다.

[그림 3.2] 테이블 스페이스의 물리적 구성



참고

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

3.2.2. 테이블 스페이스 생성, 제거

테이블 스페이스는 생성되는 유형에 따라 **시스템 테이블 스페이스**(System Tablespace)와 **비시스템 테이블 스페이스**(Non System Tablespace)로 구분된다. 시스템 테이블 스페이스는 데이터베이스가 생성될 때 자동으로 생성되는 테이블 스페이스이며, 비시스템 테이블 스페이스는 일반 사용자가 생성한 테이블 스페이스이다.

본 절에서는 비시스템 테이블 스페이스를 생성하고 제거하는 방법을 설명한다.

테이블 스페이스 생성

테이블 스페이스를 생성하기 위해서는 **CREATE TABLESPACE** 문을 사용해야 한다. 테이블 스페이스의 이름, 테이블 스페이스에 포함되는 데이터 파일과 데이터 파일의 크기, 익스텐트의 크기 등을 설정할 수 있다. UNIFORM SIZE 옵션을 설정하면 일정한 크기로 익스텐트를 할당한다. 해당 옵션을 설정하지 않으면 기본 방식인 auto allocate를 사용한다. auto allocate는 세그먼트 크기에 따라 할당할 익스텐트의 크기를 유동적으로 결정하여 익스텐트를 할당하는 방식이다.

기본 옵션

다음은 하나의 데이터 파일 my_file.dtf로 구성되는 테이블 스페이스 my_space를 기본 옵션으로 생성하는 예이다. UNIFOMR SIZE를 설정하지 않으면 auto allocate 방식으로 익스텐트를 할당한다.

```
CREATE TABLESPACE my_space DATAFILE '/usr/tibero/dtf/my_file.dtf' SIZE 50M;
```

데이터 파일 my_file.dtf는 SQL 문장을 실행함과 동시에 생성된다. 만약 동일한 이름의 파일이 이미 사용중이라면, 에러를 반환하게 된다. 데이터 파일 my_file.dtf의 크기는 50MB이며, 테이블 스페이스의 크기도 50MB가 된다.

데이터블록 크기 8KB 기준, 세그먼트 크기 별 할당하는 익스텐트 크기는 아래와 같다.

세그먼트 크기	익스텐트 크기
1MB 미만	128KB
64MB 미만	1MB
1GB 미만	8MB
1GB 이상	64MB

UNIFORM SIZE 옵션

UNIFORM SIZE 옵션을 설정하면 하나의 테이블 스페이스 내의 모든 익스텐트의 크기를 항상 일정하게 관리할 수 있다. 아래는 하나의 데이터 파일 my_file.dtf로 구성되는 테이블 스페이스 my_space를 생성하면서 UNIFORM SIZE를 설정하는 예이다. UNIFORM SIZE를 통해 할당할 익스텐트 크기를 256KB로 고정한다.

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

데이터 파일 my_file.dtf는 SQL 문장을 실행함과 동시에 생성된다. 만약 동일한 이름의 파일이 이미 사용중이라면, 에러를 반환하게 된다. 데이터 파일 my_file.dtf의 크기는 50MB이며, 테이블 스페이스의 크기도 50MB가 된다.

참고

데이터블록 크기 8KB 기준, UNIFORM SIZE를 128KB 보다 작게 설정하더라도 익스텐트 최소 크기인 128KB로 설정된다.

참고

Tibero에서는 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.

예를 들어 하나의 익스텐트의 크기가 256KB이고, 데이터 블록의 크기가 4KB라고 한다면 하나의 익스텐트에는 총 64개의 데이터 블록이 포함된다. 또한 하나의 테이블 스페이스를 두 개 이상의 데이터 파일로 구성할 수도 있다.

예를 들면 다음과 같다.

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

① 20MB 크기의 데이터 파일 my_file21.dtf를 정의한다.

② 30MB 크기의 데이터 파일 my_file22.dtf를 정의한다. 테이블 스페이스 my_space2의 전체 크기는 총 50MB가 된다.

③ 테이블 스페이스 my_space2는 하나의 익스텐트의 크기가 64KB로 설정한다.

하나의 테이블 스페이스에 포함되는 데이터 파일의 개수는 데이터베이스와 시스템 환경에 따라 달라진다. 하나의 테이블 스페이스에 많은 데이터가 저장되면 여러 개의 데이터 파일로 테이블 스페이스를 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. 테이블 스페이스를 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/dtf/my_file.dtf' SIZE 50M
    AUTOEXTEND ON NEXT 1M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;
```

Tibero에서는 데이터 블록을 할당한 정보를 테이블 스페이스에 비트맵 형태로 저장한다. 따라서 테이블 스페이스 내의 익스텐트의 최대 개수는 "테이블 스페이스의 크기 / 익스텐트의 크기"보다 작은 값이 된다.

테이블 스페이스 제거

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

다음은 테이블 스페이스를 제거하는 예이다.

```
DROP TABLESPACE my_space;
```

이 SQL 문장을 실행하면 데이터베이스에서 테이블 스페이스는 제거되지만 테이블 스페이스를 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

테이블 스페이스를 생성하거나 제거하면 그러한 내용이 컨트롤 파일에 동시에 반영된다. CREATE TABLESPACE, DROP TABLESPACE 문에 대한 자세한 내용은 "Tibero SQL 참조 안내서"를 참고한다.

3.2.3. 테이블 스페이스 변경

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

다음은 테이블 스페이스 my_space에 새로운 데이터 파일 my_file02.dtf를 추가하는 예이다.

```
ALTER TABLESPACE my_space ADD DATAFILE 'my_file02.dtf' 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

1. 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.

2. Hot Backup 중에 데이터 파일 추가는 가능하지만 Drop은 불가능 하다.

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.dtf의 크기를 변경하는 예이다.

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

특정 테이블 스페이스에 읽고 쓰는 모든 접근을 허용하지 않으려면 ALTER TABLESPACE 문에서 **OFFLINE** 절을 이용하여 테이블 스페이스를 오프라인 상태로 변경하면 된다.

테이블 스페이스 오프라인은 NORMAL과 IMMEDIATE 두 가지 모드를 지원한다.

모드	**설명**
NORMAL	체크포인트를 수행한 후 테이블 스페이스 오프라인을 수행한다. 향후 테이블 스페이스 온라인을 수행할 때 미디어 복구가 필요없다.
IMMEDIATE	체크포인트를 수행하지 않고 테이블 스페이스 오프라인을 수행한다. 향후 테이블 스페이스 온라인을 수행할 때 미디어 복구가 필요하다. 따라서 ARCHIVELOG 모드에서만 가능하다.

다음은 테이블 스페이스 my_space를 NORMAL 모드로 오프라인 상태로 만든 후 다시 온라인 상태로 만드는 예이다.

```
ALTER TABLESPACE my_space OFFLINE [NORMAL];
ALTER TABLESPACE my_space ONLINE;
```

참고

SYSTEM, UNDO, TEMP 테이블 스페이스는 오프라인 상태로 변경할 수 없다.

다음은 테이블 스페이스 my_space를 IMMEDIATE 모드로 오프라인 상태로 만든 후 미디어 복구를 수행한 뒤 다시 온라인 상태로 만드는 예이다.

```
ALTER TABLESPACE my_space OFFLINE IMMEDIATE;
ALTER DATABASE RECOVER AUTOMATIC TABLESPACE my_space;
ALTER TABLESPACE my_space ONLINE;
```

참고

미디어 복구에 대한 자세한 내용은 “11.3.3. 미디어 복구”를 참고한다.

3.2.4. 테이블 스페이스 정보 조회

Tibero에서는 테이블 스페이스를 효율적으로 관리하기 위해 다음 표에 나열된 뷰(정적 뷰, 동적 뷰 포함)를 통해 테이블 스페이스의 정보를 제공하고 있다.

테이블 스페이스 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 내의 모든 테이블 스페이스의 정보를 조회하는 뷰이다.
USER_TABLESPACES	현재 사용자에 속한 테이블 스페이스의 정보를 조회하는 뷰이다.
V$TABLESPACE	Tibero 내의 모든 테이블 스페이스에 대한 간략한 정보를 조회하는 뷰이다.

참고

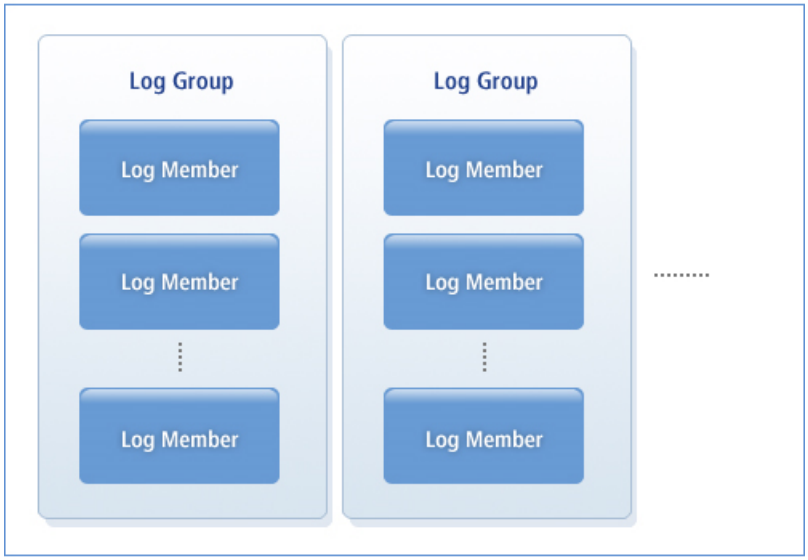
정적 뷰와 동적 뷰에 대한 자세한 내용은 "Tibero 참조 안내서"를 참고한다.

3.3. 로그 파일

로그 파일은 Redo 로그를 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 로그의 구조를 나타내는 그림이다.

[그림 3.3] Redo 로그의 구조



Redo 로그

[그림 3.3]에서 보듯이 Redo 로그는 두 개 이상의 로그 그룹(Log Group)으로 구성된다. Tibero에서는 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.

Tibero는 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 모드 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.

Tibero에서는 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 실행된다. DBA는 이러한 일이 발생하지 않도록 로그 그룹의 개수를 충분히 설정해야 한다.

NOARCHIVELOG 모드 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 아카이브되기를 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.

3.3.1. 로그 파일 구성

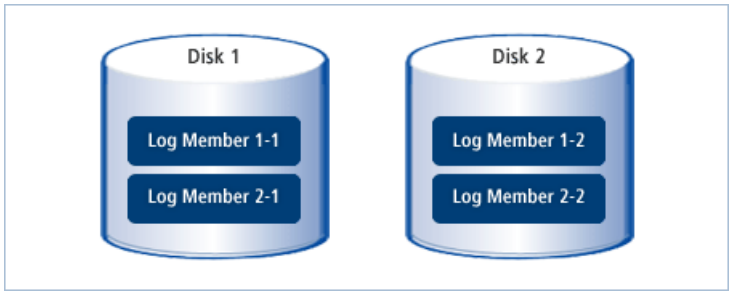
로그 멤버는 기본적으로 하나의 로그 파일이다. Redo 로그를 구성할 때 각 로그 그룹과 로그 멤버에 서로 다른 로그 파일을 할당해야 한다. 로그 파일은 데이터 파일과 서로 다른 디스크에 저장할 것을 권장한다. 로그 파일과 데이터 파일을 같은 디스크에 저장하는 경우 디스크에 장애가 발생한다면 데이터베이스를 다시 복구할 수 없게 된다. 각 로그 그룹이 여러 개의 로그 멤버로 구성된다면 최소한 로그 멤버 하나는 데이터 파일과 다른 디스크에 저장되어야 한다.

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 원인

이 된다. 따라서 서로 다른 로그 그룹의 로그 파일은 각각 다른 디스크에 저장하는 것이 시스템 성능을 높이는 데 도움이 된다.

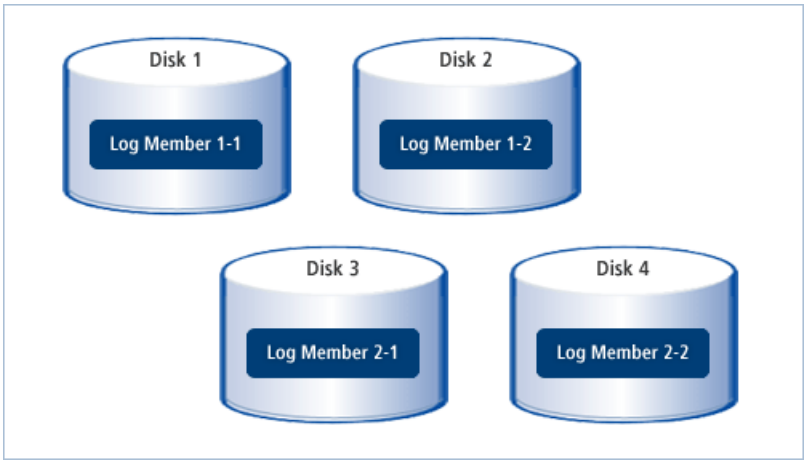
주의

단, Instance Recovery 할 때 정합성이 깨진 로그 멤버에 접근할 경우 해당 멤버를 컨트롤 파일에서 제거시킨다.

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.

아카이브 로그의 저장과 다중화

아카이브된 로그가 저장되는 위치는 초기화 파라미터 LOG_ARCHIVE_DEST로 지정한다. 필요한 아카이브 로그를 찾지 못하면 미디어 복구를 할 수 없기 때문에 아카이브 로그를 잘 보관하는 것이 중요하다.

아카이브 로그 역시 다중화하여 여러 위치에 저장할 수 있다. 아카이브 로그를 다중화할 때는 초기화 파라미터 LOG_ARCHIVE_DEST_1, LOG_ARCHIVE_DEST_2, ..., LOG_ARCHIVE_DEST_9를 사용한다. 파라미터 문자열에서 'location='으로 저장위치를 지정하고, mandatory나 optional로 다중화 아카이브 로그 저장에 실패하는 경우 동작을 지정한다(지정하지 않으면 optional로 처리된다).

mandatory로 지정된 경우 해당위치에 다중화된 아카이브 로그 저장을 성공할 때까지 계속 시도한다. 계속 실패할 경우 Redo 로그 그룹이 재사용되지 않아 데이터베이스 전체가 멈출 수 있다. optional로 지정된 위치는 아카이브 로그 저장에 실패하더라도 재시도하지 않으며, 다른 작업이 정상적으로 계속 진행된다.

[예 3.1] tip 설정 예시

```
LOG_ARCHIVE_DEST =   "/usr/tibero/log/archive"
LOG_ARCHIVE_DEST_1 = "location=/usr/tibero/archive1 mandatory"
LOG_ARCHIVE_DEST_2 = "location=/usr/tibero/archive2 mandatory"
```

```
LOG_ARCHIVE_DEST_3 = "location=/usr/tibero/archive3"
LOG_ARCHIVE_DEST_4 = "location=/usr/tibero/archive4 optional"
```

3.3.2. 로그 파일 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 파일 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. 로그 멤버의 크기의 최소값은 512KB, 최대값은 2TB이다.

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 파일 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?

  Tibero는 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.

3.3.3. 로그 파일 정보 조회

Tibero에서는 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

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

3.4. 컨트롤 파일

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

컨트롤 파일은 Tibero에 의해서만 생성과 갱신을 할 수 있으며 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.
테이블 스페이스	테이블 스페이스를 구성하는 데이터 파일 또는 생성되었거나 변경된 타임스탬프 등이 있다.
데이터 파일	데이터 파일의 이름과 위치 또는 생성되었거나 변경된 타임스탬프 등이 있다.
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.

Tibero에서는 데이터베이스를 다시 기동할 때마다 먼저 컨트롤 파일을 참조한다.

참조하는 절차는 다음과 같다.

1. 테이블 스페이스와 데이터 파일의 정보를 얻는다.

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

3. 필요한 데이터를 읽는다.

Tibero에서 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.

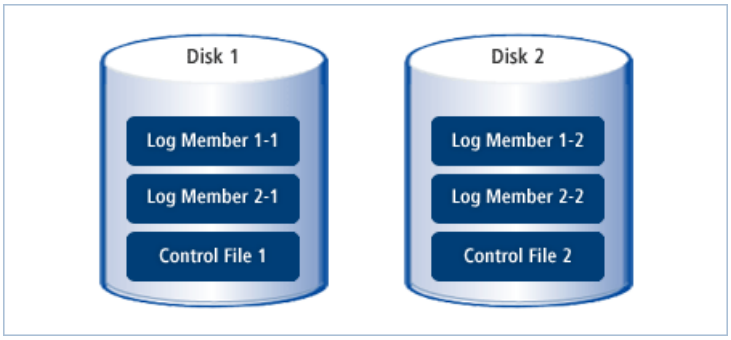
주의

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

컨트롤 파일의 다중화

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.

Tibero에서는 컨트롤 파일로부터 정보를 확인할 때 여러 복사본 중에서 하나만 읽는다. 그리고 테이블 스페이스의 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.

주의

TAC 환경에서 컨트롤 파일을 다중화하는 경우 하나의 노드라도 특정 컨트롤 파일 write에 실패하게 되면 해당 파일은 invalidate 처리되어야 한다. 따라서 모든 노드의 tip 파일에는 다중화할 컨트롤 파일의 개수와 각 컨트롤 파일 경로가 동일하게 기재되어 있어야 한다.

3.4.1. 컨트롤 파일 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.

Tibero는 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 일부 컨트롤 파일에 장애가 발생했을 경우에도 하나 이상의 컨트롤 파일이 정상이면 Tibero는 문제없이 운영된다.

컨트롤 파일의 백업은 물리적 백업과 논리적 백업을 지원한다. 하지만 물리적 백업은 수동으로 모든 데이터 파일들도 함께 백업을 해야하며 절차가 매우 중요하다. 절차를 제대로 지키지 않았을 시 추후 데이터 복구가 불가능하기 때문에, 실수할 확률이 있는 물리적 백업은 보통 추천하지 않으며 책임져주지 않는다. 따라서 논리적 백업 방법으로 컨트롤 파일을 생성하는 SQL 문장을 백업하는 것이 일반적이다. 또한 테이블 스페이스, 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. 자세한 내용은 “11.2.2. 백업 실행”을 참고한다.

다음의 SQL 문장은 컨트롤 파일을 물리적 백업하는 예이다.

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO
          '/tibero7/backup/ctrlfile1.ctl';
```

다음의 SQL 문장은 컨트롤 파일을 논리적 백업하는 예이다.

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE AS
          '/tibero7/backup/ctrlfile1.sql' REUSE NORESETLOGS;
```

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

3.4.2. 컨트롤 파일 정보 조회

Tibero에서는 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 참조 안내서"를 참고한다.

제4장 스키마 객체 관리

본 장에서는 Tibero에서 실제로 데이터베이스를 구성하는 데 필요한 논리적 저장 영역 즉, 스키마 객체를 관리하는 방법과 이를 구성하는 최소 단위인 디스크 블록을 설명한다.

4.1. 개요

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. 테이블, 인덱스와 같이 실제 물리적 공간을 가지는 객체를 **세그먼트**라고 한다.

4.2. 테이블

테이블(Table)은 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. Tibero는 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.

4.2.1. 테이블 생성, 변경, 제거

본 절에서는 테이블을 생성, 변경, 제거하는 방법을 설명한다.

테이블 생성

테이블을 생성하기 위해서는 **CREATE TABLE 문**을 사용해야 한다.

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.

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 "4.3. 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.
테이블 스페이스	– 테이블이 저장될 테이블 스페이스를 설정한다. 이 구성요소는 테이블을 생성할 때 선택적으로 사용할 수 있다. – 테이블 스페이스를 명시하지 않으면 사용자의 디폴트 테이블 스페이스로 설정된다. – 테이블의 적절한 배치가 데이터베이스 성능에 큰 영향을 미치므로 테이블의 소유자는 테이블이 저장될 테이블 스페이스를 별도로 명시하는 것이 좋다.
디스크 블록 파라미터	디스크 블록마다 테이블의 갱신에 대비하여 어느 정도 여유 공간을 남겨둘 것인가를 설정한다. 자세한 내용은 "4.4. 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, 4개의 제약조건으로 선언되어 있다. 또한 테이블 스페이스 my_space에 저장되며 디스크 블록 파라미터의 세부 항목(PCTFREE, INITRANS)을 모두 설정하였다.

테이블 변경

테이블을 변경하기 위해서는 **ALTER TABLE 문**을 사용해야 한다.

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.

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.

구성요소	설명
컬럼의 이름	– 컬럼의 이름과 정의된 컬럼의 속성을 변경한다. – 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.
테이블 스페이스	– 테이블에 할당된 테이블 스페이스는 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 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 “4.3. 제약조건”에서 자세히 설명한다. 또한 테이블을 생성하거나 변경을 위한 자세한 문법은 "Tibero SQL 참조 안내서"를 참고한다.

테이블 제거

테이블을 제거하기 위해서는 **DROP TABLE 문**을 사용해야 한다.

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.

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;
```

4.2.2. 테이블 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 “4.4. 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.

4.2.3. 테이블 정보 조회

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

정적 뷰	설명
DBA_TABLES	Tibero 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.
DBA_TBL_COLUMNS	Tibero 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 참조 안내서"를 참고한다.

4.2.4. 테이블 압축

Tibero는 테이블에 대해 중복된 컬럼 값을 압축하여 저장공간을 절약하는 압축 기능을 제공한다. 값 블록에 존재하는 중복된 컬럼 값을 한번만 저장함으로써 압축을 수행하게 된다. 이런 중복 컬럼 값들이 저장되는 공간을 심볼 테이블이라고 한다. 심볼 테이블은 해당 블록 안에 저장되기 때문에 압축된 컬럼의 원래 값을 참조하기 위해서는 해당 블록만을 참조하면 된다.

압축된 테이블에 대한 DML 지원은 일반 테이블과 동일하다. 즉, insert, update, delete 등의 DML을 지원한다. bulk가 아닌 일반적인 insert 문으로 추가된 로우는 압축이 되지 않으므로 비 압축 테이블에 대한 insert와 동일한 성능을 가진다. delete 또한 비 압축 테이블에 대한 delete와 동일한 성능을 가진다. 하지만 update는 압축을 해제해야 하는 경우가 있으므로 비 압축 테이블에 대한 update 보다 성능이 좋지 않을 수 있다.

압축을 수행하면 디스크 공간을 절약할 수 있지만 압축을 위해 CPU를 더 많이 소모한다. 테이블에 DML이 많은 경우 점점 더 압축 효율이 낮아지게 된다. 따라서 OLTP 환경 보다는 OLAP 환경에서 더 유리하다.

테이블 압축 대상

압축은 테이블과 파티션, 서브 파티션에 대해 가능하다. 파티션과 서브 파티션 각각에 대해 압축 유무를 지정할 수 있다. 즉, 한 파티션은 압축하고 한 파티션은 압축하지 않은 상태로 테이블을 생성할 수 있다.

[예 4.6] 압축이 지정된 테이블 생성

```
CREATE TABLE EMP (
       EMPNO DECIMAL(4),
       ENAME VARCHAR(10),
       JOB VARCHAR(9),
       MGR DECIMAL(4),
       HIREDATE VARCHAR(14),
       SAL NUMBER(7,2),
       COMM NUMBER(7,2),
       DEPTNO NUMBER(2))
       COMPRESS;
```

[예 4.7] 파티션별 압축을 지정하는 테이블 생성

```
CREATE TABLE EMP (
       EMPNO DECIMAL(4),
       ENAME VARCHAR(10),
       JOB VARCHAR(9),
       MGR DECIMAL(4),
       HIREDATE VARCHAR(14),
```

```
    SAL NUMBER(7,2),
    COMM NUMBER(7,2),
    DEPTNO NUMBER(2))
    COMPRESS
PARTITION BY RANGE(EMPNO)
( PARTITION EMP_PART1 VALUES LESS THAN(500),
  PARTITION EMP_PART2 VALUES LESS THAN(1000) NOCOMPRESS,
  PARTITION EMP_PART3 VALUES LESS THAN(1500),
  PARTITION EMP_PART4 VALUES LESS THAN(2000) NOCOMPRESS,
  PARTITION EMP_PART5 VALUES LESS THAN(MAXVALUE));
```

테이블 압축 상태 확인

*_TABLES, *_TBL_PARTITIONS 정적 뷰를 쿼리해 보면 테이블의 압축 상태를 알 수 있다. compression 컬럼의 값이 'YES'인 경우 추가적인 DML에 대해 압축을 수행하게 된다.

[예 4.8] 테이블의 압축 상태 확인

```
select table_name, compression from user_tables;

TABLE_NAME      COMPRESSION
--------------- -------
EMP             YES
```

테이블 압축 방법

다음의 경우 테이블에 데이터가 압축된다.

- Direct Path Loader
- Direct Path Insert(Parallel INSERT 또는 append hint로 수행되는 bulk INSERT)
- CREATE TABLE AS SELECT 문

위 경우 처럼 대량 insert하는 경우에는 테이블에 EXCLUSIVE LOCK을 잡기 때문에 다른 DML을 수행할 수 없다. 압축된 후 위 경우 이외 수행되는 일반적인 insert, update 문의 로우 값은 압축되지 않는다.

기존 테이블 압축 및 압축 해제

ALTER TABLE MOVE 문을 이용하면 기존 테이블에 대한 압축 상태 변경을 할 수 있다. 즉, 압축된 테이블을 압축 해제하거나 압축 해제된 테이블을 압축할 수 있다. 단, **ALTER TABLE MOVE** 문이 수행되는 동안에는 테이블에 EXCLUSIVE LOCK을 잡게 되므로 다른 DML을 수행할 수 없다. 파티션을 가진 테이블의 경우 테이블 자체가 아닌 파티션 단위로 MOVE를 수행해야 한다.

만약, DML을 수행하는 도중에 압축 상태를 바꾸려는 경우 Exchange DDL 기능을 이용하면 Online 중에 테이블을 압축 또는 압축 해제할 수 있다.

[예 4.9] 기존 테이블 또는 파티션을 압축하거나 압축 해제하는 예

```
ALTER TABLE TBL_COMP MOVE COMPRESS;
ALTER TABLE EMP MOVE PARTITION EMP_PART1 NOCOMPRESS;
```

ALTER TABLE 문을 이용하면 추가적인 DML에 대해 압축을 수행할지 여부를 설정할 수 있다. 압축이 지정된 테이블에 ALTER TABLE 문으로 압축을 하지 않기로 지정하면 이후 수행되는 Direct Path Loader, Parallel Insert 등에 대해 압축을 수행하지 않게 된다. 하지만 기존의 데이터의 상태는 바뀌지 않는다.

[예 4.10] 테이블의 추가적인 DML에 대한 압축 여부를 변경하는 예

```
ALTER TABLE EMP COMPRESS;
```

테이블 압축 시 제약 사항

한 번이라도 압축이 지정된 테이블에는 기본 값이 지정된 컬럼 추가, 컬럼 삭제 DDL을 수행할 수 없으며, Long 타입 컬럼을 가지고 있는 압축된 테이블에는 컬럼 추가 DDL이 허용되지 않는다.

4.2.5. OLTP COMPRESS

Tibero는 OLTP 환경에서 Direct Path Load/Insert가 아닌 일반 DML이 발생하는 테이블에 대한 압축 기능을 제공한다.

일반 compression 방식은 Direct Path Insert/Load 작업 시에만 데이터 압축이 가능하며 압축된 데이터에 대한 DML 시 압축을 해제하는 반면, OLTP compression 기능을 사용하면 일반적인 DML 작업이 발생한 테이블에 대해서도 압축을 수행하여 압축 효과를 유지할 수 있다. 테이블 데이터 블록의 공간 사용률 및 압축률이 특정 threshold 값 이하일 때 해당 데이터 블록에서 중복된 데이터를 제거하는 방식으로 압축을 수행한다.

OLTP COMPRESS 대상

OLTP COMPRESS는 테이블과 파티션, 서브 파티션에 대해 가능하다. 파티션과 서브 파티션 각각에 대해 압축 유무를 지정할 수 있다. 즉, 한 파티션은 압축하고 한 파티션은 압축하지 않은 상태로 테이블을 생성할 수 있다.

[예 4.11] OLTP COMPRESS 옵션이 지정된 테이블 생성

```
CREATE TABLE EMP (
       EMPNO DECIMAL(4),
       ENAME VARCHAR(10),
       JOB VARCHAR(9),
       MGR DECIMAL(4),
```

```
       HIREDATE VARCHAR(14),
       SAL NUMBER(7,2),
       COMM NUMBER(7,2),
       DEPTNO NUMBER(2))
       COMPRESS FOR OLTP;
```

[예 4.12] 파티션별 OLTP COMPRESS 옵션을 지정하는 테이블 생성

```
CREATE TABLE EMP (
       EMPNO DECIMAL(4),
       ENAME VARCHAR(10),
       JOB VARCHAR(9),
       MGR DECIMAL(4),
       HIREDATE VARCHAR(14),
       SAL NUMBER(7,2),
       COMM NUMBER(7,2),
       DEPTNO NUMBER(2))
       COMPRESS FOR OLTP
   PARTITION BY RANGE(EMPNO)
   ( PARTITION EMP_PART1 VALUES LESS THAN(500),
     PARTITION EMP_PART2 VALUES LESS THAN(1000) NOCOMPRESS,
     PARTITION EMP_PART3 VALUES LESS THAN(1500),
     PARTITION EMP_PART4 VALUES LESS THAN(2000) NOCOMPRESS,
     PARTITION EMP_PART5 VALUES LESS THAN(MAXVALUE));
```

테이블의 OLTP COMPRESS 옵션 확인

*_TABLES, *_TBL_PARTITIONS 정적 뷰를 쿼리해 보면 테이블의 OLTP COMPRESS 옵션 적용 유무를 알 수 있다. COMPRESS_FOR 옵션이 'ADVANCED'인 경우 OLTP COMPRESS 기능을 사용하는 테이블이다 .

[예 4.13] 테이블의 압축 상태 확인

```
select table_name, compression from user_tables;

TABLE_NAME      COMPRESSION
--------------- -------
EMP             YES
```

OLTP COMPRESS 옵션

테이블에 대해 지정할 수 있는 OLTP COMPRESS 옵션은 다음과 같다.

옵션	설명
COMPRESS FOR OLTP	Direct Path Insert/Load가 아닌 일반 DML에 대해서만 테이블을 압축한다.

옵션	설명
ROW STORE COM PRESS ADVANCED	COMPRESS FOR OLTP와 동일하며, 마찬가지로 Direct Path Insert/Load가 아닌 일반 DML에 대해서만 테이블을 압축한다.
COMPRESS FOR ALL OPERATIONS	Direct Path Insert/Load와 일반 DML에 대해서 테이블을 압축한다.

기존 테이블 압축 옵션 변경

ALTER TABLE MOVE 문을 이용하면 기존 테이블에 대한 압축을 변경할 수 있다. 파티션을 가진 테이블의 경우 테이블 자체가 아닌 파티션 단위로 MOVE를 수행해야 한다.

[예 4.14] 기존 테이블에 대한 압축을 변경하는 예

```
CREATE TABLE TBL_COMP (C1 NUMBER) COMPRESS;
ALTER TABLE TBL_COMP MOVE COMPRESS FOR OLTP;
ALTER TABLE TBL_COMP MOVE COMPRESS FOR ALL OPERATIONS;
```

ALTER TABLE 문을 이용하여 OLTP COMPRESS 옵션을 지정할 경우, 이후 수행되는 DML에 대해서 압축을 수행하도록 할 수 있다.

단, ALTER TABLE 구문을 수행한 시점에서 기존 데이터에 대한 압축이 수행되지는 않는다.

[예 4.15] 테이블의 추가적인 DML에 대한 압축을 변경하는 예

```
CREATE TABLE EMP (C1 NUMBER);
INSERT INTO EMP SELECT LEVEL FROM DUAL CONNECT BY LEVEL <= 1000;
COMMIT;
ALTER TABLE EMP COMPRESS FOR OLTP;
```

4.2.6. INDEX ORGANIZED TABLE

INDEX ORGANIZED TABLE은 인덱스의 B-TREE 구조를 이용해 데이터를 저장하는 형태의 테이블을 말한다. 일반적인 테이블에서는 데이터가 로우 단위로 무작위로 블록에 저장되지만, INDEX ORGANIZED TABLE은 인덱스와 유사한 형태로 기본 키를 기준으로 로우가 정렬되어 인덱스 리프 블록에 저장된다.

로우가 너무 크거나 지역 효율성을 위해 특정 컬럼 부터는 데이터 영역에 저장할 수도 있다. 이를 오버플로우 데이터 영역이라고 한다.

INDEX ORGANIZED TABLE은 다음과 같은 장점을 가진다.

- 기본 키를 랜덤 액세스할 때 기본 키를 기준으로 정렬되어 있으므로 일반 테이블보다 더욱 빠르다. 일반 테이블에서 인덱스가 있는 경우라도 기본 키를 인덱스에서 찾고 로우 ID로 다시 해당 로우를 찾아야 한다. 하지만 INDEX ORGANIZED TABLE에서는 인덱스 영역에서 해당 로우를 바로 찾을 수 있기 때문에 추가적인 디스크 검색이 불필요하기 때문이다.
- 인덱스와 데이터 영역에 기본 키가 중복 저장 되지 않으므로 스토리지 사용량이 줄어든다.

단, 잦은 수정이 발생할 경우 인덱스를 재구조화하는 부담이 생기므로 DML이 자주 발생하는 환경에서는 적합하지 않을수 있다. INDEX ORGANIZED TABLE은 기본 키로 전체 로우가 정렬되어 저장되므로 다른 키로 인덱스를 만들고자 하는 경우 SECONDARY INDEX를 만들 수 있다.

INDEX ORGANIZED TABLE 생성

INDEX ORGANIZED TABLE은 CREATE TABLE 문 뒤에 ORGANIZATION INDEX 구문을 덧붙여 생성할 수 있다. 이때 반드시 기본 키(primary key) 선언을 해주어야 한다.

추가로 줄 수 있는 파라미터는 다음과 같다.

파라미터	설명
OVERFLOW	INDEX ORGANIZED TABLE 로우의 크기 제한을 넘어서거나 INCLUDING 이후의 컬럼들은 OVERFLOW 데이터 영역에 저장된다. 이때 사용자가 원하는 테이블 스페이스를 줄 수 있다.
INCLUDING	INCLUDING으로 선언된 컬럼 이후부터는 무조건 오버플로우 데이터 영역에 저장된다. INCLUDING은 기본 키의 마지막 컬럼이나 기본 키가 아닌 임의의 컬럼을 지정할 수 있다.
PCTTHRESHOLD	블록 크기를 기준으로 INDEX ORGANIZED TABLE의 인덱스 영역에 들어갈 수 있는 로우의 한 최대 크기의 비율을 말한다. INCLUDING을 지정하지 않은 경우 PCTTHRESHOLD 범위를 넘는 컬럼부터 OVERFLOW 데이터 영역에 저장된다. INCLUDING을 지정한 경우라도 INCLUDING이 지정된 이전의 컬럼까지의 크기 합이 PCTTHRESHOLD 범위를 넘어서게 되면 OVERFLOW 데이터 영역에 저장된다.

[예 4.16] INDEX ORGANIZED TABLE 생성

```
CREATE TABLE TBL_IOT
  (
      COL1 NUMBER,
      COL2 VARCHAR2(20),
      COL3 VARCHAR2(10),
      COL4 VARCHAR2(10),
      PRIMARY KEY (COL1, COL2)
  )
  ORGANIZATION INDEX
  PCTTHRESHOLD 40
  OVERFLOW;
```

INDEX ORGANIZED TABLE을 생성할 때 다음의 제약조건을 유의한다.

- LOB이나 LONG은 포함할 수 없고 VIRTUAL COLUMN을 생성할 수 없다.

- PRIMARY INDEX는 UNUSABLE이 불가능하다. SECONDARY INDEX는 UNUSABLE이 가능하다.
- 컬럼의 최대 개수는 1000개이다.
- REVERSE, BITMAP, DOMAIN INDEX를 SECONDARY INDEX로 생성할 수 없다.
- 인덱스 영역에는 최대 255개의 컬럼만 저장할 수 있다. 컬럼 개수가 그 이상이거나 인덱스 영역에 다 저장 할 수 없는 경우 OVERFLOW를 지정해야 한다.
- PCTTHRESHOLD의 값은 1-50이다. 하지만 실제 인덱스 영역에 저장할 수 있는 로우의 최대 크기는 구조적인 문제로 블록의 50%보다 더 작다.
- 모든 컬럼은 PCTTHRESHOLD보다 작아야 한다.
- COMPOSITE PARTITIONING이 불가능하다.
- PARTITION MODIFY, MOVE, RENAME, SPLIT, MERGE가 불가능하다.
- TABLE COMPRESS가 불가능하다.
- TABLE을 생성한 후 ALTER TABLE PROPERTY 시에는 ADD/DROP SUPPLEMENTAL LOG만 가능하다.

INDEX ORGANIZED TABLE 삭제

DROP TABLE 구문으로 삭제할 수 있다.

[예 4.17] INDEX ORGANIZED TABLE 삭제

```
DROP TABLE TBL_IOT;
```

4.3. 제약조건

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

4.3.1. 제약조건 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.

제약조건	설명
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.

제약조건 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 4.18] 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 문**을 사용한다.

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

[예 4.19] 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
);
```

- 테이블 단위로 선언하는 경우

  모든 컬럼을 정의한 후 제약조건을 선언한다. 다음은 테이블 단위로 제약조건을 선언하는 예이다. 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.

[예 4.20] 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)
);
```

제약조건 변경

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

다음은 제약조건을 변경하는 예이다.

- 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 4.21] 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** 절을 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 4.22] 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 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.

제약조건 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 Tibero가 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 4.23] 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. ...
```

4.3.2. 제약조건 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 특성상 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.

  DISABLE은 아래와 같이 두 가지 옵션을 추가적으로 사용할 수 있다.

옵션	설명
VALIDATE	이 옵션을 사용하는 경우 제약조건에 걸려있는 인덱스를 드랍하고 해당 제약조건 컬럼에 대한 변경이 불가능하다.
NOVALIDATE	사용자가 옵션을 입력하지 않으면 기본적으로 적용되는 옵션이다.

제약조건 상태 변경

제약조건의 상태를 변경하기 위해서는 **ALTER TABLE 문**을 사용해야 한다.

다음은 제약조건의 상태를 변경하는 예이다.

- ENABLE 상태로 변경하는 경우

  [예 4.24] 제약조건의 상태 변경 - ENABLE

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

- DISABLE 상태로 변경하는 경우

[예 4.25] 제약조건의 상태 변경 - DISABLE

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

- NOVALIDATE로 추가한 제약조건을 다시 VALIDATE로 변경하는 경우

[예 4.26] 제약조건의 상태 변경 - VALIDATE

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

4.3.3. 제약조건 정보 조회

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

정적 뷰	설명
DBA_CONSTRAINTS	Tibero 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 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 참조 안내서"를 참고한다.

4.4. 디스크 블록

디스크 블록은 데이터를 저장하는 물리적인 최소 단위이며 크기가 일정하다. Tibero에서는 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.

4.4.1. 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 파라미

터에 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.

4.4.2. 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. 이때 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.

4.4.3. 파라미터 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;
```

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.

다음과 같이 파라미터는 **ALTER TABLE 문**을 이용하여 변경할 수 있다.

```
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;
```

4.5. 인덱스

인덱스(Index)는 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.

4.5.1. 인덱스 생성, 제거

Tibero는 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.

인덱스 생성

인덱스를 생성하기 위해서는 **CREATE INDEX 문**을 사용해야 한다.

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.

다음은 인덱스를 생성할 때 포함되는 구성요소이다.

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

구성요소	설명
파티션 인덱스	파티션 인덱스를 설정할 수 있다. 인덱스를 생성할 때 선택적으로 사용할 수 있다.

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

[예 4.27] 인덱스의 생성

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

인덱스 제거

인덱스를 제거하기 위해서는 **DROP INDEX 문**을 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 4.27]에서 생성한 인덱스 'EMP_FK'를 제거하는 예이다.

[예 4.28] 인덱스의 제거

```
DROP INDEX EMP_FK;
```

4.5.2. 인덱스 효율적인 관리

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

인덱스 검색 유형

인덱스를 이용하여 테이블의 로우를 검색하는 방법은 다음과 같다.

- 단일 키 검색(single key search)
  - 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(range search)
  - 검색 범위에 포함되는 키를 갖는 로우를 모두 검색하는 방법이다.

- 복합 키 검색(composite key search)
  - 서로 다른 두 개 이상의 컬럼을 하나의 키로 조합하여 검색하는 방법이다.
  - 다음은 복합 키를 이용하여 검색하는 예이다.

[예 4.29] 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 = 'Seoul'**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.

  Tibero에서는 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.

4.5.3. 인덱스 압축

Tibero는 인덱스 키를 구성하는 컬럼들을 공통부분(Prefix)과 비공통부분(Suffix)으로 나누어 공통부분을 공유하는 방식으로 압축기능을 제공한다. 공통부분에 해당하는 컬럼들을 따로 저장하고, 공통부분에 접근할 수 있는 위치정보와 비공통부분을 합쳐 하나의 키로 사용한다.

각각의 공통부분은 모든 비공통부분이 공유한다.

새로운 키가 추가될 때, 기존에 만들어진 공통부분을 사용할 수 없으면 새로운 공통부분을 만들어 저장한다. 따라서, DML이 빈번한 테이블에 속한 인덱스에 대해서 압축기능을 사용하면 성능 저하가 있을 수 있다.

인덱스 압축 시 제약 사항

유일 인덱스(Unique Index)의 경우, 한 컬럼을 제외한 모든 컬럼을 압축 대상으로 선정할 수 있다. 모든 컬럼을 압축 대상으로 포함하면 중복이 없기 때문에 압축이 불가하다. 따라서, 키를 구성하는 컬럼이 2개 이상 되어야 한다.

비유일 인덱스(Non-Unique Index)는 모든 컬럼을 압축 대상으로 선정할 수 있으므로, 키를 구성하는 컬럼이 1개만 있어도 가능하다.

압축 범위를 양수 N으로 명시한 경우, 첫번째 컬럼부터 N개의 컬럼이 압축 대상이 된다.

압축 범위를 명시하지 않는 경우, 유일 인덱스는 마지막 컬럼을 제외한 모든 컬럼에 대해서 압축을 수행한다. 반면, 비유일 인덱스는 모든 컬럼이 압축 대상이 된다.

기존의 인덱스를 압축하거나 압축을 풀려면, ALTER문을 통해서 수행할 수 없고, 기존의 인덱스를 DROP한 후에 CREATE문을 통해 설정해야한다.

[예 4.30] 인덱스 압축 방법

```
CREATE TABLE EMP
(
    EMPNO NUMBER NOT NULL,
    ENAME VARCHAR(16) NOT NULL,
    ADDR VARCHAR(24),
    SALARY NUMBER,
    DEPTNO NUMBER
);

*볼드처리된 컬럼은 정상적으로 압축된 대상을 의미한다.
```

```
CREATE UNIQUE INDEX IDX1 ON EMP (EMPNO) COMPRESS 1;
TBR-7552: Cannot use COMPRESS option for a single column key

CREATE UNIQUE INDEX IDX1 ON EMP (EMPNO, ENAME) COMPRESS 2;
TBR-7551: Invalid COMPRESS prefix length value

CREATE UNIQUE INDEX IDX1 ON EMP (EMPNO, ENAME) COMPRESS;
Index 'IDX1' created.

CREATE INDEX IDX2 ON EMP(EMPNO, ENAME, ADDR) COMPRESS;
Index 'IDX2' created.

CREATE INDEX IDX3 ON EMP(SALARY, DEPTNO, ADDR) COMPRESS 0;
TBR-7551: Invalid COMPRESS prefix length value

CREATE INDEX IDX3 ON EMP(SALARY, DEPTNO, ADDR) COMPRESS 2;
Index 'IDX3' created.
```

[예 4.31] 인덱스 압축 상태 확인

```
SELECT INDEX_NAME, UNIQUENESS, COMPRESSION FROM DBA_INDEXES WHERE TABLE_NAME = 'EMP';


INDEX_NAME UNIQUENESS COMPRESSION

---------- ---------- -----------

IDX1       UNIQUE     ENABLED
IDX2       NONUNIQUE  ENABLED

IDX3       NONUNIQUE  ENABLED
```

[예 4.32] 파티션 인덱스 압축 방법

```
CREATE TABLE SALES
(
    SALES_NO NUMBER,
    SALES_DATE DATE,
    PRICE NUMBER
)
PARTITION BY RANGE (SALES_DATE)
(
    PARTITION SALES_P1 VALUES LESS THAN (TO_DATE('2019/01/01', 'YY/MM/DD')),
    PARTITION SALES_P2 VALUES LESS THAN (TO_DATE('2020/01/01', 'YY/MM/DD')),
    PARTITION SALES_P3 VALUES LESS THAN (TO_DATE('2021/01/01', 'YY/MM/DD')),
    PARTITION SALES_P4 VALUES LESS THAN (MAXVALUE)
);
```

```
CREATE INDEX IDX1 ON SALES(SALES_DATE) LOCAL COMPRESS;
Index 'IDX1' created.
```

4.5.4. 인덱스 정보 조회

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

정적 뷰	설명
DBA_INDEXES	Tibero 내의 모든 인덱스의 정보를 조회하는 뷰이다.
USER_INDEXES	현재 사용자에 속한 인덱스의 정보를 조회하는 뷰이다.
ALL_INDEXES	사용자가 접근 가능한 인덱스의 정보를 조회하는 뷰이다.
DBA_IDX_COLUMNS	Tibero 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 참조 안내서"를 참고한다.

4.5.5. 인덱스 사용 여부 모니터링

Tibero에서는 인덱스의 사용 여부를 모니터링할 수 있는 기능을 제공하고 있다. 인덱스 모니터링의 결과는 V$OBJECT_USAGE를 조회해서 알 수 있다.

다음은 인덱스 사용 여부를 모니터링하는 예이다.

```
SQL> CREATE TABLE T (A NUMBER);
Table 'T' created.

SQL> CREATE INDEX I ON T(A);
Index 'I' created.

SQL> ALTER INDEX I MONITORING USAGE;
Index 'I' altered.

SQL> SELECT /*+ index(t i) */ * from t where a > 0;
0 rows selected.

SQL> SELECT USED FROM V$OBJECT_USAGE;
   USED
```

```
-------
      Y

1 row selected.
```

4.6. 뷰

뷰(View)는 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.

4.6.1. 뷰 생성, 변경, 제거

본 절에서는 뷰를 생성, 변경, 제거하는 방법을 설명한다.

뷰 생성

뷰를 생성하기 위해서는 **CREATE VIEW 문**을 사용해야 한다.

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.

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 4.33] 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;
```

뷰 변경

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

다음은 뷰를 변경하는 예이다.

[예 4.34] 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.

뷰 제거

뷰를 제거하기 위해서는 **DROP VIEW 문**을 사용해야 한다.

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.

다음은 뷰를 제거하는 예이다.

[예 4.35] 뷰의 제거

```
DROP VIEW EMP_DEPT;
```

4.6.2. 뷰 정보 조회

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

정적 뷰	설명
DBA_VIEWS	Tibero 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 내의 모든 뷰에 속한 컬럼의 갱신 가능성 정보를 조회하는 뷰이다.
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 참조 안내서"를 참고한다.

4.7. 시퀀스

시퀀스(Sequence)는 순차적으로 부여하는 고유번호이다. 주로 새로운 데이터에 유일한 고유번호를 자동으로 부여할 때 사용한다.

4.7.1. 시퀀스 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.

시퀀스 생성

시퀀스를 생성하기 위해서는 **CREATE SEQUENCE 문**을 사용해야 한다.

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.

다음은 시퀀스를 생성할 때 포함되는 구성요소이다.

구성요소	**설명**
시퀀스의 이름	시퀀스의 이름을 설정한다. 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. 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(최댓값)으로 순환한다.
ORDER	요청한 순서대로 시퀀스 결과가 나오도록 한다. 단 TAC 상에서만 의미가 있는 요소이다. (Single에서는 항상 순차적으로 값이 나옴)

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 Tibero가 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 4.36] 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.

시퀀스 변경

시퀀스를 변경하기 위해서는 **ALTER SEQUENCE 문**을 사용해야 한다.

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.

다음은 시퀀스를 변경하는 예이다.

[예 4.37] 시퀀스의 변경

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

```
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
```

```
----------
        14

1 selected
```

시퀀스 제거

시퀀스를 제거하기 위해서는 **DROP SEQUENCE 문**을 사용해야 한다.

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.

다음은 시퀀스를 제거하는 예이다.

[예 4.38] 시퀀스의 제거

```
DROP SEQUENCE NEW_ID;
```

4.7.2. 시퀀스 정보 조회

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

정적 뷰	설명
DBA_SEQUENCES	Tibero 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 참조 안내서"를 참고한다.

4.8. 동의어

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

4.8.1. 동의어 생성, 제거

본 절에서는 동의어를 생성, 제거하는 방법을 설명한다.

동의어 생성

동의어를 생성하기 위해서는 **CREATE SYNONYM 문**을 사용해야 한다.

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.

다음은 동의어를 생성할 때 포함되는 구성요소이다.

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

다음은 동의어를 생성하는 예이다.

[예 4.39] 동의어의 생성

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

동의어 제거

정의된 동의어를 변경하기 위해서는 먼저 동의어를 제거하고 다시 생성해야 한다. 동의를 제거하기 위해서는 **DROP SYNONYM 문**을 사용해야 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.

다음은 동의어를 제거하는 예이다.

[예 4.40] 동의어의 제거

```
DROP SYNONYM T1;
```

4.8.2. 공용 동의어 생성, 제거

본 절에서는 공용 동의어를 생성, 제거하는 방법을 설명한다.

공용 동의어 생성

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

공용 동의어를 생성하기 위해서는 **CREATE PUBLIC SYNONYM 문**을 사용해야 한다.

다음은 공용 동의어를 생성할 때 포함되는 구성요소이다. 각 구성요소는 공용 동의어를 생성할 때 반드시 포함되어야 한다.

구성요소	설명
공용 동의어의 이름	공용 동의어의 이름을 설정한다.
테이블의 이름	공용 동의어를 적용할 테이블의 이름을 설정한다.

다음은 공용 동의어를 생성하는 예이다.

[예 4.41] 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 PUB_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 동의어 제거

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

다음은 공용 동의어를 제거하는 예이다.

[예 4.42] 공용 동의어의 제거

```
DROP PUBLIC SYNONYM PUB_T1;
```

4.8.3. 동의어 정보 조회

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

정적 뷰	설명
DBA_SYNONYMS	Tibero 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 참조 안내서"를 참고한다.

4.9. 트리거

트리거(Trigger)는 테이블의 로우를 삽입, 변경, 삭제할 때 자동으로 수행되도록 미리 지정해 놓은 PSM(Persistent Store Module) 프러시저이다. 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.

4.9.1. 트리거 생성, 제거

본 절에서는 트리거의 생성, 제거하는 방법을 설명한다.

트리거 생성

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

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.

다음은 EMP 테이블에 새로운 로우를 삽입한 직후에 트리거를 수행하는 예이다.

[예 4.43] 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 SQL 참조 안내서"를 참고한다.

트리거 제거

트리거를 제거하기 위해서는 **DROP TRIGGER 문**을 사용해야 한다.

다음은 트리거를 제거하는 예이다.

[예 4.44] 트리거의 제거

```
DROP TRIGGER TRG1;
```

4.10. 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**

(Partition)은 대용량 서비스를 하는 데이터베이스에서 효율적으로 관리하고 동작하기 위해 지원하는 옵션이다. 파티션은 서로 다른 테이블 스페이스에 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.

4.10.1. 파티션 생성

파티션을 생성하기 위해서는 **CREATE TABLE 문**을 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 4.45] 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 PART4
        VALUES LESS THAN (70, 80);

ALTER TABLE PARTITIONED_TABLE1 DROP PARTITION PART1;
```

파티션 정의 시 주의사항

파티션을 정의할 때 다음과 같은 주의사항이 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);
  ```

- 현재 Tibero는 HASH 파티션 테이블에 대한 파티션 추가를 지원하지 않는다.

4.10.2. 복합 파티션 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 sold_date 컬럼을 이용해 월별로 RANGE 파티셔닝을 우선 한 뒤 각 파티션들을 다시 product_id 로 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)
);
```

두 개의 컬럼(위 예의 경우 sold_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 SQL 참조 안내서"를 참고한다.

4.10.3. 인터벌 파티셔닝

인터벌 파티션은 레인지 파티션의 확장으로 최초에 선언되지 않은 범위의 데이터가 들어왔을 때 그에 맞는 파티션을 내부적으로 생성하는 기능이다.

아래는 인터벌 파티션을 사용하여 테이블을 생성하는 예이다.

[예 4.46] 인터벌 파티션의 생성

```
CREATE TABLE PARTITIONED_TABLE_INTERVAL (C1 NUMBER, C2 CLOB, C3 NUMBER)
             PARTITION BY RANGE (C1)
             INTERVAL(10)
             (
                PARTITION PART1 VALUES LESS THAN (10),
                PARTITION PART2 VALUES LESS THAN (20),
                PARTITION PART3 VALUES LESS THAN (30)
             );
```

인터벌 파티셔닝 시 주의사항

인터벌 파티셔닝을 할 때 다음과 같은 주의사항이 있다.

- 인터벌 파티셔닝의 파티션 키는 반드시 Numerical 또는 Data range이어야 한다.
- 테이블을 생성하는 경우 최소 하나의 파티션은 지정하거나 생성해야 한다.
- Composite partitioning에서 primary 파티셔닝으로 사용 가능하지만 subpartition에서는 사용할 수 없다.

4.10.4. 인덱스 파티션 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 4.47] 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 4.48] 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 (C3, C1)
             GLOBAL PARTITION BY RANGE (C3)
             (
                PARTITION IPART1 VALUES LESS THAN (20) INITRANS 3,
                PARTITION IPART2 VALUES LESS THAN (70) PCTFREE 10,
                PARTITION IPART3 VALUES LESS THAN (MAXVALUE)
             );
```

4.10.5. 파티션 정보 조회

Tibero에서는 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 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 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 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 Tibero의 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 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. Tibero는 패스워드를 데이터 사전(Data Dictionary)에 암호화된 형태로 저장한다.

Tibero에서 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 입력한다.

```
사용자 계정.스키마 객체
```

다음은 tibero라는 사용자 계정으로 접속하여 SYS 사용자의 SYSTEM_PRIVILEGES에 접근하는 예이다.

```
$ tbsql tibero/tmax

tbSQL 7

TmaxData Corporation Copyright (c) 2008-. 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$DATABASE가 존재하지 않으면 PUBLIC 사용자가 소유한 동의어를 검색한다. 즉, PUBLIC.V$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. 사용자 생성, 변경, 제거

사용자를 새로 생성하거나 변경, 제거하기 위해서는 DBA 특권을 가진 사용자로 Tibero에 접속한다.

Tibero에서는 기본적으로 SYS라는 사용자를 제공한다. SYS 사용자는 Tibero를 설치하는 과정에서 생기는 계정으로 DBA 역할이 부여된다.

SYS 사용자로 Tibero에 접속하는 방법은 다음과 같다.

```
$ tbsql SYS/tibero

tbSQL 7

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

Connected to Tibero.

SQL>
```

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

사용자 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 Steve라는 사용자를 생성한다.

② 사용자 Steve의 패스워드를 dsjeoj134로 설정한다. CREATE USER 문을 사용할 때에는 반드시 패스워드를 설정해야 한다. 만약 패스워드에 특수문자를 사용할 경우 'dsjeoj134!'와 같이 작은따옴표로 묶어야 한다.

③ 디폴트 테이블 스페이스를 USR로 설정한다.

다음은 데이터베이스 보안 정책에 따른 사용자를 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 테이블 스페이스를 지정하지 않으면 자동으로 시스템 테이블 스페이스를 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. 자세한 내용은 “5.2. 특권”을 참고한다.

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

SQL> GRANT RESOURCE TO John;
Granted.

SQL> GRANT CONNECT TO Smith;
Granted.

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

사용자 변경

사용자에게 설정된 패스워드, 테이블 스페이스 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 테이블 스페이스를 SYSTEM 테이블 스페이스로 변경한다.

사용자 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, 프러시저, 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. 사용자 정보 조회

Tibero에서는 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.

정적 뷰	설명
USER_USERS	현재 사용자의 정보를 조회하는 뷰이다.

참고

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

5.1.3. 사용자 계정 잠금 및 해제

데이터베이스 사용자가 계정에 접속할 수 없도록 계정을 잠금 상태로 설정하거나 잠금 상태를 해제할 수 있다.

```
SQL> ALTER USER Peter ACCOUNT LOCK;

User 'PETER' altered.
```

계정 잠금 상태에서 해당 계정에 접속을 시도하면 다음과 같은 오류 코드와 함께 접속이 중단된다.

```
SQL> conn peter/abcd;
TBR-17006: Account is locked.

No longer connected to server.
```

계정 잠금을 해제하려면 다음의 SQL을 수행한다.

```
SQL> ALTER USER Peter ACCOUNT UNLOCK;

User 'PETER' altered.
```

프로파일 설정에 따라 패스워드 사용 기한이 만료되거나 패스워드 오류 횟수 초과 등으로 계정 잠금 상태가 된 경우에도 동일한 방법으로 잠금을 해제할 수 있다.

5.1.4. 운영체제(OS) 인증을 사용한 사용자 생성

사용자 생성은 보안 정책에 따라 데이터베이스 보안 정책을 따르는 사용자 생성과 운영체제(OS) 인증 정책을 따르는 사용자 생성으로 구분된다.

운영체제(OS) 인증을 사용하는 사용자 계정은 다음과 같이 생성할 수 있다.

```
SQL> CREATE USER OSA$Steve                  ... ① ...
            IDENTIFIED externally           ... ② ...
```

① CREATE USER 문을 사용하여 OS 사용자인 Steve라는 사용자를 생성하는데 OS 인증 정책 사용자를 알리는 OSA$ prefix를 붙여 생성한다. 해당 값은 OS_AUTH_PREFIX에서 변경할 수 있으며, 기본값은 OSA$이다.

② OS 인증을 사용하는 사용자 OSA$Steve의 패스워드는 데이터베이스에서 별도로 관리하지 않는다. 즉, OS의 사용자 Steve가 존재하는 경우 host에서 인증을 완료한 것으로 가정하여 데이터베이스에서 별도로 확인하지 않는다(OS의 보안이 취약한 경우에는 사용을 권하지 않는다).

OS 인증 사용자 생성 이후 로컬에서 접속하는 방법은 다음과 같다.

```
$ tbsql /

tbSQL 7

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

Connected to Tibero.

SQL>
```

참고

원격에서 OS 인증 사용자 접속은 보안상의 문제로 지원하지 않는다.

5.2. 특권

사용자가 데이터베이스의 특정 스키마 객체에 접근하려면 특권(Privilege)을 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 Peter라는 사용자 계정으로 데이터베이스에 접속한다.

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

③ Smith라는 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 생성한다. 동일한 특권을 부여해야 할 사용자가 많은 경우 역할을 이용함으로써 GRANT 명령의 사용 횟수를 줄일 수 있다. 특권과 마찬가지로 역할도 PUBLIC 사용자에게 부여하면 모든 사용자에게 역할을 부여한 것과 동일한 효과를 갖는다.

Tibero에서 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 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.
TRUNCATE	테이블에 TRUNCATE를 수행할 수 있는 권한이다. 이 권한을 사용하려면 USE_TRUNCATE_PRIVILEGE 파라미터를 'Y'로 설정해야 한다.

스키마 객체 특권 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 한다. 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 특권 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	테이블 스페이스를 생성하는 권한이다.
ALTER TABLESPACE	테이블 스페이스를 변경하는 권한이다.
DROP TABLESPACE	테이블 스페이스를 제거하는 권한이다.
SELECT ANY DICTIONARY	DICTIONARY를 조회할 수 있는 권한이다. 이 권한을 할당 받으면 SYS, SYSCAT, SYSGIS 소유의 객체들을 조회할 수 있다.
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 테이블, 뷰, 실체화 뷰를 조회할 수 있는 권한이다.
INSERT ANY TABLE	임의의 스키마에 속한 테이블 또는 동의어 테이블에 로우를 삽입하는 권한이다.
UPDATE ANY TABLE	임의의 스키마에 속한 테이블 또는 동의어 테이블의 로우를 갱신하는 권한이다.
DELETE ANY TABLE	임의의 스키마에 속한 테이블에 로우를 제거하는 권한이다.
TRUNCATE ANY TABLE	임의의 스키마에 속한 테이블에 TRUNCATE를 수행할 수 있다. 이 권한을 사용하기 위해서는 USE_TRUNCATE_PRIVILEGE 파라미터를 'Y'로 설정해야 한다.
CREATE ANY INDEX	임의의 스키마에 속한 테이블 또는 실체화 뷰의 인덱스를 생성하는 권한이다.
ALTER ANY INDEX	임의의 스키마에 속한 테이블에 인덱스를 수정하는 권한이다.

시스템 특권	설명
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.
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 MATERIALIZED VIEW	자신의 스키마에 실체화 뷰를 생성하는 권한이다.
CREATE ANY MATERIALIZED VIEW	임의의 스키마에 실체화 뷰를 생성하는 권한이다.
ALTER ANY MATERIALIZED VIEW	임의의 스키마에 속한 실체화 뷰를 변경하는 권한이다.
DROP ANY MATERIALIZED VIEW	임의의 스키마에 속한 실체화 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 시퀀스에서 시퀀스 또는 동의어 시퀀스를 조회할 수 있는 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 PSM을 생성하는 권한이다.
CREATE ANY PROCEDURE	임의의 스키마에 PSM을 생성하는 권한이다.
ALTER ANY PROCEDURE	임의의 스키마에 속한 PSM을 변경하는 권한이다.
DROP ANY PROCEDURE	임의의 스키마에 속한 PSM을 제거하는 권한이다.
EXECUTE ANY PROCEDURE	임의의 스키마에 속한 PSM을 실행하는 권한이다.
CREATE TRIGGER	자신의 스키마에 속한 트리거를 생성하는 권한이다.
CREATE ANY TRIGGER	임의의 스키마에 속한 트리거를 생성하는 권한이다.

시스템 특권	설명
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.
REFRESH ANY CACHE GROUP	임의의 스키마에 속한 캐시 그룹을 비울 수 있는 권한이다.

시스템 특권 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 특권 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. 특권 정보 조회

Tibero에서는 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.
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 또는 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 참조 안내서"를 참고한다.

5.2.4. 부가적인 특권

다음은 부가적인 특권을 설정할 수 있는 파라미터에 대한 설명이다.

파라미터	설명
USE_TRUNCATE_PRIVILEGE	TRUNCATE 수행을 위해서 TRUNCATE ANY TABLE 시스템 특권 또는 TRUNCATE 스키마 객체 특권을 사용 할 수 있다. 이 특권들을 사용하기 위해서는 USE_TRUNCATE_PRIVILEGE 파라미터를 'Y'로 설정해야 한다. – Y : TRUNCATE를 수행하기 위해서 자신의 스키마에 속한 테이블이 아닐 경우에는 TRUNCATE 특권이 부여되어야 한다. – N : TRUNCATE 수행을 위해 DROP ANY TABLE 시스템 특권이 부여되어 있어야 한다.
GRANT ALL	GRANT ALL을 수행할 때 ALL 특권의 기준은 USE_TRUNCATE_PRIVILEGE 파라미터의 여부에 따라 다르다. – Y : GRANT ALL에 TRUNCATE 특권까지 포함된다. – N : GRANT ALL에 TRUNCATE 특권이 포함되지 않는다.
REVOKE ALL	REVOKE ALL의 경우 USE_TRUNCATE_PRIVILEGE 파라미터의 여부와 상관없이 TRUNCATE 특권이 취소된다.

5.3. 프로파일

데이터베이스 사용자의 패스워드 관리 정책을 지정할 수 있다.

예를 들어 사용자 1, 2, 3은 90일 후 패스워드 사용 기간이 만료되도록 설정하고, 사용자 4, 5는 패스워드 사용 기간이 30일후 만료되며 한 번 사용한 패스워드는 재사용 못하도록 패스워드 사용 정책을 각각 다르게 설정할 필요가 있다고 가정한다. 이 경우 90일 후 패스워드 사용 기간이 만료되도록 설정한 프로파일과 30일 후 패스워드 사용 기간이 만료되며 한 번 사용한 패스워드는 재사용 못하도록 설정한 프로파일을 각각 생성하고 사용자 1, 2, 3은 첫 번째 프로파일을 사용하도록 지정하고 사용자 4, 5는 두 번째 프로파일을 사용하도록 지정한다.

이처럼 프로파일은 사용자 패스워드 관리 정책을 다양하게 생성하고 각각의 사용자에게 특정 정책을 사용하도록 지정함으로써 사용자별로 그룹화된 패스워드 정책을 관리할 수 있는 기능을 제공한다.

5.3.1. 프로파일 생성, 변경, 제거

다음은 프로파일을 생성하는 예이다.

```
SQL> CREATE PROFILE prof LIMIT
      failed_login_attempts 3
      password_lock_time 1/1440
      password_life_time 90
      password_reuse_time unlimited
      password_reuse_max 10
      password_grace_time 10
      password_verify_function verify_function;

Profile 'PROF' created.
```

프로파일 파라미터 종류

위 예제에서 보았듯이 프로파일을 생성, 변경할 때는 세부적인 파라미터를 지정할 수 있다.

각 파라미터에 대한 상세 설명은 "Tibero SQL 참조 안내서"에 기술되어 있다. 참고로 각 파라미터의 기본 값은 데이터베이스를 생성할 때 함께 생성된 DEFAULT 프로파일에 의해 결정되며, 이 기본값들은 **DBA_PROFILES 뷰**를 통해 확인할 수 있다.

다음은 프로파일을 변경하는 예이다.

```
SQL> ALTER PROFILE prof LIMIT
      password_lock_time 1
      password_reuse_time 30;

Profile 'PROF' altered.
```

위의 SQL 문장이 실행되면 PROF이라는 프로파일의 패스워드가 오류 횟수를 초과하는 경우 계정 잠금 상태가 1일간 지속된다. 또한 한 번 사용한 패스워드는 30일 동안 동일한 패스워드로 다시 변경할 수 없다.

다음은 프로파일을 삭제하는 예이다.

```
SQL> DROP PROFILE prof CASCADE;

Profile 'PROF' dropped.
```

삭제하려는 프로파일을 이미 사용자가 지정한 경우 반드시 cascade 옵션을 붙여야 한다. casade 옵션을 사용하면 해당 프로파일을 지정한 모든 사용자의 프로파일 지정 정보를 일괄 삭제한다. 단, 사용자 정보는 삭제하지 않는다.

5.3.2. 프로파일 지정

다음은 사용자 계정을 생성할 때 프로파일을 지정하는 예이다.

```
SQL> CREATE USER Peter IDENTIFIED BY abcd PROFILE prof;

User 'PETER' created.
```

다음은 prof1 프로파일에서 Default 프로파일로 변경하는 예이다. Default 프로파일은 데이터베이스를 생성할 때 기본으로 함께 생성되는 프로파일이다.

```
SQL> ALTER USER Peter PROFILE default;

User 'PETER' altered.
```

5.3.3. 프로파일 정보 조회

프로파일 관련 정보는 다음과 같이 확인할 수 있다.

```
SQL> select * from dba_profiles;

PROFILE   RESOURCE_NAME             RESOURCE_TYPE LIMIT
--------- ------------------------- ------------- ---------------
DEFAULT   FAILED_LOGIN_ATTEMPTS     PASSWORD      UNLIMITED
DEFAULT   PASSWORD_LIFE_TIME        PASSWORD      UNLIMITED
DEFAULT   PASSWORD_REUSE_TIME       PASSWORD      UNLIMITED
DEFAULT   PASSWORD_REUSE_MAX        PASSWORD      UNLIMITED
DEFAULT   PASSWORD_VERIFY_FUNCTION  PASSWORD      NULL_VERIFY_FUNCTION
DEFAULT   PASSWORD_LOCK_TIME        PASSWORD      1
DEFAULT   PASSWORD_GRACE_TIME       PASSWORD      UNLIMITED
DEFAULT   LOGIN_PERIOD              PASSWORD      UNLIMITED

16 rows selected.
```

다음은 사용자별로 적용된 프로파일 정보를 조회하는 예이다. PETER라는 사용자에 T_PROF라는 프로파일이 할당된 상황을 확인할 수 있다.

```
SQL> select username, profile from dba_users;

USERNAME   PROFILE
---------- ----------
USER1
PETER      T_PROF
OUTLN
SYSGIS
SYSCAT
SYS

6 rows selected.
```

5.3.4. VERIFY_FUNCTION

Tibero에서는 Password를 설정할 때 보안성을 위해 **Verify _function**으로 Password 설정에 제한을 둘 수 있다.

다음은 Default Verify_function을 사용하는 경우 발생하는 에러에 대한 설명이다.

- `-20001: Password same as user.`

  유저 이름과 비밀번호가 달라야한다.

- `-20002: Password length less than 4.`

  비밀번호 길이가 4 이상이어야 한다.

- `-20003: Password too simple.`

  비밀번호가 예상하기 쉬운 단어이면 안된다. Default에서 설정된 단어(welcome, database, account, 'user', 'password', 'tibero', 'computer', 'abcd')는 설정할 수 없다.

- `-20004: Password should contain at least one digit, one character and one punctuation.`

  비밀번호가 숫자, 문자, 특수문자가 적어도 1개씩 포함해야 한다.

- `-20005: Password should differ by at least 3characters.`

  바로 이전의 비밀번호와 적어도 3개 이상 달라야 한다.

5.4. 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 사용자"로 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.

5.4.1. 역할 생성, 부여, 회수

다음은 역할을 생성하거나 변경, 특권을 부여하는 예이다. 사용자 SYS로 접속하여 사용자 Peter에게 CREATE ROLE, ALTER ANY ROLE, GRANT ANY ROLE 특권을 부여하고 있다.

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

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

역할 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 역할에 대한 설명이다.

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.

역할 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.

역할 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.

5.4.2. 미리 정의된 역할

Tibero에서는 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.
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.
HS_ADMIN_ROLE	-	Heterogeneous Service를 이용하는 DBA에게 필요한 시스템 특권을 포함하고 있는 특권이다.

5.4.3. 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 비활성화한다.

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.

예를 들면 다음과 같다.

```
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;
```

5.4.4. 역할 정보 조회

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

정적 뷰	설명
DBA_ROLES	Tibero 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.
ROLE_SYS_PRIVS	현재 사용자가 접근할 수 있는 역할에 부여된 시스템 특권 정보를 조회하는 뷰이다.
ROLE_TAB_PRIVS	현재 사용자가 접근할 수 있는 역할들에 부여된 스키마 객체 특권들을 조회하는 뷰이다.
ROLE_ROLE_PRIVS	현재 사용자가 접근할 수 있는 역할들에 부여된 다른 역할들의 정보를 조회하는 뷰이다.

참고

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

5.5. 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. Tibero는 이러한 기술을 채택함으로써 기업 내 IT 자산을 보다 효과적으로 보호할 수 있다.

Tibero에서 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

  리스너 또는 클라이언트의 IP 주소에 따라 네트워크 접속을 차단하거나 허용한다.

5.5.1. 전체 네트워크 접속 제어

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

다음은 클라이언트 전체의 네트워크 접속을 허용하는 방법으로 Tibero 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.

5.5.2. IP 주소 기반 네트워크 접속 제어

5.5.2.1. 리스너의 IP를 통한 접속 제어

초기화 파라미터에 설정된 IP 주소에 따라 리스너가 사용할 IP 주소를 설정해 해당 주소로만 네트워크 접속을 허용하는 방법이다.

- LISTENER_IP

  이 방법은 특정 IP 주소로 접속하는 클라이언트의 네트워크 접속만 허용할 때 사용한다.

  – 리스너가 사용할 IP를 지정한다.

  – 위 파라미터 설정 시 ssl 및 special port도 이 주소로 바인딩 된다.

  – 운영 중 변경 및 삭제할 수 없다.

참고

IPv6를 사용하는 경우에는 위 초기화 파라미터 대신 **LISTENER_IP_6**를 사용해야 한다.

- EXTRA_LISTENER_IPS

  이 방법은 LISTENER_IP 외 추가로 접속을 허용할 IP 주소를 추가할 때 사용한다.

  – LISTENER_IP로 지정된 기본 IP 주소 외에 리스너가 사용할 추가 IP 주소이다. 따라서 LISTENER_IP가 설정된 경우에만 유효하다.

  – IPv4 형식만 지원한다.

– 운영 중 추가 및 삭제가 가능하다(아래 "동적 리스너 IP 추가 및 삭제" 참고).

– 위 파라미터 설정 시 입력한 IP 주소에 대해 기본 default port 및 추가 등록한 모든 extra port의 접속이 허용된다. 이는 동적 추가시에도 마찬가지이다.

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

<<$TB_SID.tip>>

```
EXTRA_LISTENER_IPS=192.168.1.1;192.168.2.0;
```

참고

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

참고

Windows 운영체제에서 EXTRA_LISTENER_IPS 기능은 추후 지원 예정이다.

주의

VIP 기능을 사용 중이더라도 자동으로 해당 ip가 EXTRA_LISTENER_IPS로 추가되지 않는다. 따라서 부팅 및 failover 시 항상 동적으로 추가해 주어야 한다.

동적 리스너 IP 추가 및 삭제

- LISTENER_IP 이외의 데이터베이스의 IP를 추가하고 싶은 경우 다음과 같은 설정을 통해 추가 가능하다.

```
alter system listener add ip "192.168.3.0";
```

- 추가한 리스너 IP를 삭제하는 경우 다음과 같은 명령어를 통해 삭제한다.

```
alter system listener delete ip "192.168.3.0";
```

5.5.2.2. 클라이언트의 IP를 통한 접속 제어

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.

```
예: 192.168.1.1;192.168.2.0/24
```

위의 예제에서 192.168.1.1은 192.168.1.1/32와 같은 의미이다. 192.168.2.0/24는 서브넷 마스크의 bit 수가 24bits이므로, 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_INVITED_IP의 최대 길이는 255자이다. 256 이상의 IP 주소를 설정할 경우에는 LSNR_INVITED_IP_FILE을 사용한다.

- LSNR_INVITED_IP_FILE

이 방법은 특정 파일에 접속을 서용하는 IP 주소 목록을 기재한 후 해당 파일의 절대 경로를 적어주면 그 파일을 읽어서 INVITED_IP를 설정한다.

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

– 여러 개의 IP 주소를 설정하는 경우에는 한 라인에 1개의 IP 주소를 설정한다.

– 설정할 수 있는 파일의 최대 크기는 8MB이다.

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

<</home/tibero/invited_ip.txt>>

```
192.168.1.1
                    192.168.2.0/24
                    192.1.0.0/16
```

<<$TB_SID.tip>>

```
LSNR_INVITED_IP_FILE=/home/tibero/invited_ip.txt
```

- 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
```

- LSNR_DENIED_IP_FILE

  이 방법은 특정 파일에 접속을 허용하지 않는 IP 주소 목록을 기재한 후 해당 파일의 절대 경로를 적어 주면 그 파일을 읽어서 DENIED_IP를 설정한다.

  – LSNR_INVITED_IP_FILE 초기화 파라미터의 설정 방법과 동일하다.

LSNR_INVITED_IP와 LSND_DENIED_IP 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 서버를 운영하는 중에 서버를 다시 기동하지 않고 LSNR_INVITED_IP 또는 LSNR_DENIED_IP의 설정을 변경하려는 경우 우선 $TB_SID.tip 파일에 LSNR_INVITED_IP 또는 LSNR_DENIED_IP의 설정을 변경한 후 파일을 저장하고 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.

5.5.3. 동적 리스너 포트 추가 및 삭제

LISTENER_PORT 이외의 데이터베이스의 접속 포트를 추가하고 싶은 경우 다음과 같은 설정을 통해 추가 가능하다.

```
alter system listener add port 8799;
```

추가한 리스너 포트를 삭제하는 경우 다음과 같은 명령어를 통해 삭제한다.

```
alter system listener delete port 8799;
```

EXTRA_LISTENER_PORTS 파라미터를 설정하는 경우 끝부분에 세미콜론(;) 구분자를 넣어야 한다.

<<$TB_SID.tip>>

```
EXTRA_LISTENER_PORTS=8799;8800;
```

참고

Windows 운영체제에서 동적 리스너 포트 추가 및 삭제는 추후 지원 예정이다.

5.6. 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.

5.6.1. 감사 설정과 해제

감사를 설정하거나 해제하려면 **AUDIT** 또는 **NOAUDIT** 명령어을 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 문이 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 그것이 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 “5.6.3. SYS 사용자에 대한 감사”를 참고한다.

5.6.2. 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_FILE_SIZE=10M
```

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

다음은 저장된 감사 기록의 항목에 대한 설명이다.

항목	설명
SERIAL_NO	특정 세션을 식별할 수 있는 보조 키의 개념이다. 동시에 Active할 수 있는 세션의 개수는 한정적이며 그런 Active한 세션을 식별하는 것이 Session ID이다. Session ID는 세션을 식별하는데 한계가 있다. 예를 들어 1번 세션이 수행하다가 모든 수행을 마치고 Close된 다음 다시 Open되어 수행한다고 했을 때 단순히 Session ID만 가지고는 1번 세션이 수행한 두 개의 작업을 식별할 수 없게 된다. 즉, 1번 세션이 Close하기 전에 수행한 세션과 다시 Open하여 수행한 이후 세션을 식별할 수 없다. 따라서 이를 위해 부가적인 정보가 필요한데 그것이 Serial Number(SERIAL_NO)이다. 이 숫자는 세션이 다시 열리거나 재사용될 때 증가되어 Session ID와 붙여서 특정 세션을 식별할 수 있게 된다.
AUD_NO	세션이 열린 다음 기록된 Audit 엔트리의 순차적인 번호이다.
STMT_ID	Audit을 남기게 한 Statement를 파싱하여 나온 Physical Plan의 내부적인 PPID이다.
CLIENT_ID	Tibero에 접속하여 명령을 수행하도록 한 클라이언트의 이름이다(Client Name으로 명명하는 것이 더 정확할 수도 있다).

항목	설명
PRIV_NO	해당 Statement를 수행하는 데에 필요한 Privilege 번호이다. 내부적으로 음수는 System privileges(SELECT_ANY_TABLE 등의 ANY 시리즈), 양수는 Object Privilege(select on t1에서 select)를 의미한다.
ACTION	해당 Statement가 결국 성공했는지 여부를 나타내는 컬럼이다. – S : 성공을 의미한다. – F : 실패를 의미한다.
USGMT_ID	현재 트랜잭션을 위한 Undo를 기록한 Undo Segment ID이다.
SLOTNO	Undo Segment에 위치한 Slot들 중 어떤 Slot인지를 식별할 수 있는 값이다. 트랜잭션이 시작하면 우선 Undo segment에서 Slot 하나를 할당받아야 한다. 따라서 Undo Segment와 Slot Number를 이용하여 현재 Active한 트랜잭션들을 식별할 수 있다.
WRAPNO	SERIAL_NO가 세션을 위한 부가적인 정보였다면 WRAPNO는 트랜잭션을 위한 부가적인 정보이다. 트랜잭션이 커밋되고, 또 다른 트랜잭션이 해당 Slot을 재사용하게 되면 이 값이 증가하게 된다. 따라서 3개의 정보(USGMT_ID, SLOTNO, WRAPNO)를 이용하면 과거에 수행되었던 모든 트랜잭션을 포함하여 하나의 트랜잭션을 식별할 수 있게 된다.

AUDIT_TRAIL이 OS일 경우 로그의 형태로 Audit를 남기고, DB 또는 DB_EXTENDED일 경우에는 SYS._DD_AUD 테이블에 Insert를 하여 View 형태로 보여준다.

참고

1. $TB_SID.tip 파일 설정에 관한 자세한 내용은 "Tibero 참조 안내서"를 참고한다.
2. SYS 사용자에 대한 감사 기록은 데이터베이스에 저장되지 않는다. 자세한 내용은 “5.6.3. SYS 사용자에 대한 감사”를 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 참조 안내서"를 참고한다.

5.6.3. 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 'Y'로 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.

5.6.4. Fine-Grained Auditing

Fine-Grained Auditing(FGA)는 일반적인 감사(Auditing)보다 상세한 감사를 하려는 기능이다. AUDIT_TRAIL 파라미터를 설정하지 않아도 DBMS_FGA 패키지를 이용하여 감사를 할 수 있으며, 특정 스키마의 컬럼 또는 데이터에 대한 감사를 가능하게 한다.

감사 기록

DBMS_FGA 패키지를 사용하여 사용자가 감사할 조건(policy)을 등록한다. 사용자가 등록한 조건이 맞을 때 SYS.FGA_LOG$ 테이블에 기록이 된다.

다음은 policy를 등록 후 감사 기록이 저장되고 확인하는 예이다.

```
BEGIN
DBMS_FGA.ADD_POLICY(
  OBJECT_SCHEMA => 'TIBERO',
  OBJECT_NAME => 'T',
  POLICY_NAME => 'FGA_POLICY',
  AUDIT_CONDITION => null,
  AUDIT_COLUMN => 'C1',
```

```
  STATEMENT_TYPES => 'SELECT',
  AUDIT_TRAIL => DBMS_FGA.DB + DBMS_FGA.EXTENDED);
END;
/

SQL> SELECT C1 FROM T;

        C1
----------
       123

1 row selected.

SQL>  SELECT OBJ$SCHEMA, OBJ$NAME, POLICYNAME, LSQLTEXT FROM SYS.FGA_LOG$;

OBJ$SCHEMA OBJ$NAME   POLICYNAME LSQLTEXT
---------- ---------- ---------- --------------------
TIBERO     T          FGA_POLICY SELECT C1 FROM T
```

위의 예제는 DBMS_FGA 패키지의 ADD_POLICY 프러시저를 사용하여 policy를 등록한다. 등록 된 policy의 이름은 FGA_POLICY이며, TIBERO.T 테이블 C1 컬럼이 SELECT 문으로 조회될 때 감사한다. C1 컬럼을 조회한 뒤 SYS.FGA_LOG$에 감사 기록이 저장이 된다.

참고

1. Policy 등록과 제거 및 활성화, 비활성화에 대한 자세한 내용은 "Tibero tbPSM 참조 안내서"의 "제 13장 DBMS_FGA"를 참고한다.

2. 감사 기록 조회에 대한 자세한 내용은 “5.6.2. 감사 기록”을 참고한다.

제6장 데이터 암호화

본 장에서는 Tibero에서 데이터 암호화(Data Encryption) 기능을 사용하고 관리하기 위한 방법을 설명한다.

6.1. 개요

Tibero에서는 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 Tibero는 데이터를 암호화하여 디스크에 저장하는 기능을 제공하고 있다.

DBA가 암호화할 데이터(테이블의 컬럼 또는 테이블 스페이스)를 지정하면 Tibero는 데이터를 저장할 때 내부적으로 암호화하여 저장하고 검색할 때 복호화해서 보여준다. 이때 사용자나 애플리케이션 프로그램은 데이터의 암호화 여부를 고려할 필요가 없다.

Tibero는 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
SEED	SEED 128 bits key
ARIA128	ARIA 128 bits key
ARIA192	ARIA 192 bits key
ARIA256	ARIA 256 bits key
GOST	GOST 256 bits key

6.2. 환경설정

데이터 암호화 기능을 사용하기 위해서는 우선 먼저 보안 지갑을 생성해야 한다. 보안 지갑은 암호화에 사용할 마스터 키를 보관하고 있다. Tibero는 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 6.1] 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 6.2] 보안 지갑의 위치 설정 : <<$TB_SID.tip>>

```
WALLET_FILE=/path/to/TBWALLET
```

3. **보안 지갑의 사용**

   - 보안 지갑 열기

     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 6.3] 보안 지갑 열기

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY 'tibero';
System altered.
```

주의

TAC 환경에서 각 노드가 가지고 있는 보안 지갑은 반드시 동일해야 한다.

   - 보안 지갑 닫기

     데이터 암호화 기능을 사용하지 못하도록 보안 지갑을 닫으려면 이 역시 DBA 권한을 가진 사용자가 다음의 명령을 수행한다.

     [예 6.4] 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.

보안 지갑을 닫기 전 암호화된 테이블 스페이스에 대한 active tx는 모두 commit(또는 rollback)되어야 한다.

- 보안 지갑 패스워드 변경

  생성한 보안 지갑의 패스워드를 변경하려면 DBA 권한을 가진 사용자가 다음의 명령을 수행한다. IDENTIFIED BY 절 뒤에는 변경하고 싶은 패스워드를 입력한다.

  [예 6.5] 보안 지갑 패스워드 변경

```
SQL> ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY 'supertibero';
System altered.
```

- 보안 지갑 교체

주의

이전에 보안 지갑을 사용한 적이 있는 경우 새로운 보안 지갑으로 교체할 수 없다.

또한, 보안 지갑이 변경되는 경우 미디어 복구는 지원하지 않는다.

정말 보안 지갑 교체를 원하는 경우 안전하게 교체하는 방법은 다음과 같다.

a. 보안 지갑을 이용해 생성한 암호화 컬럼과 암호화 테이블 스페이스를 모두 일반 컬럼과 일반 스페이스로 변경 또는 삭제한다.

b. CHECK_WALLET_IN_USE 파라미터를 N으로 설정한다. (default:Y)

   만약 TAC 환경이라면 모든 노드에 대해 수행한다.

c. 위 과정을 차례로 수행한 후 새로운 보안 지갑을 열어 사용한다.

6.3. 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 때는 암호화 알고리즘과 SALT 옵션을 사용할 것인지의 여부를 명시할 수 있다. 암호화 알고리즘은 Tibero에서 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

6.3.1. 암호화 컬럼을 갖는 테이블 생성

CREATE TABLE 문에서 암호화할 컬럼을 정의할 때 ENCRYPT 절을 지정하여 테이블을 생성한다.

[예 6.6] 암호화 컬럼을 갖는 테이블 생성 - 디폴트 암호화 옵션(AES192 알고리즘, SALT)

```
SQL> CREATE TABLE customer (
       cust_id       CHAR(4) CONSTRAINT cust_id_pk PRIMARY KEY NOT NULL,
       cust_name     VARCHAR(20) NULL,
       cust_type     VARCHAR(18) NULL,
       cust_addr     VARCHAR(40) NULL,
       cust_tel      VARCHAR(15) ENCRYPT NULL,
       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 6.7] 암호화 컬럼을 갖는 테이블 생성 - AES256 알고리즘, NO SALT 옵션 설정

```
SQL> CREATE TABLE customer (
       cust_id       CHAR(4) CONSTRAINT cust_id_pk PRIMARY KEY NOT NULL,
       cust_name     VARCHAR(20) NULL,
       cust_type     VARCHAR(18) NULL,
       cust_addr     VARCHAR(40) ENCRYPT USING 'AES256' NULL,
       cust_tel      VARCHAR(15) ENCRYPT NO SALT NULL,
       reg_date      DATE NULL
);
Table 'CUSTOMER' created.
```

6.3.2. 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 6.8] 암호화 컬럼 추가

```
SQL> ALTER TABLE customer ADD (cust_password VARCHAR(12) ENCRYPT NO SALT);
Table 'CUSTOMER' altered.
```

6.3.3. 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 6.9] 일반 컬럼을 암호화 컬럼으로 변경

```
SQL> ALTER TABLE customer MODIFY (reg_date ENCRYPT NO SALT);
Table 'CUSTOMER' altered.
```

6.3.4. 암호화 컬럼을 일반 컬럼으로 변경

ALTER TABLE 문에서 컬럼을 변경할 때 DECRYPT 절을 지정하면 일반 컬럼이 된다.

[예 6.10] 암호화 컬럼을 일반 컬럼으로 변경

```
SQL> ALTER TABLE customer MODIFY (reg_date DECRYPT);
Table 'CUSTOMER' altered.
```

6.3.5. 모든 암호화 컬럼의 알고리즘 변경

ALTER TABLE 문에서 REKEY 명령을 지정하면 해당 테이블의 모든 암호화 컬럼의 암호화 알고리즘이 변경된다.

[예 6.11] 모든 암호화 컬럼의 암호화 알고리즘 변경

```
SQL> ALTER TABLE customer REKEY USING '3DES168';
Table 'CUSTOMER' altered.
```

6.3.6. 암호화 컬럼에 대한 인덱스

일반 인덱스처럼 생성이 가능하지만 다음과 같은 제약사항이 있다. SALT 옵션이 지정안된 암호화 컬럼에 대해서만 인덱스를 생성할 수 있고 해당 인덱스에 대해서는 EQUAL('=') 조건만 사용하여 INDEX RANGE SCAN이 가능하다.

[예 6.12] 암호화 컬럼에 대한 인덱스

```
SQL> CREATE TABLE customer (
       cust_id       CHAR(4) CONSTRAINT cust_id_pk PRIMARY KEY NOT NULL,
       cust_name     VARCHAR(20) NULL,
       cust_type     VARCHAR(18) NULL,
       cust_addr     VARCHAR(40) ENCRYPT USING 'AES256' NULL,
       cust_tel      VARCHAR(15) ENCRYPT NO SALT NULL,
       reg_date      DATE NULL
);
Table 'CUSTOMER' created.

SQL> CREATE INDEX XI_CUSTOMER ON CUSTOMER (CUST_ADDR);
TBR-7288: Unable to encrypt index key column(s) with SALT.

SQL> CREATE INDEX XI_CUSTOMER ON CUSTOMER (CUST_TEL);
Index 'XI_CUSTOMER' created.

SQL> SELECT COUNT (*) FROM CUSTOMER WHERE CUST_TEL = '010-2233-9999';
Execution Plan
------------------------------------------------------------------------------
   1  COLUMN PROJECTION (Cost:1, %%CPU:0, Rows:1)
   2    SORT AGGR (Cost:1, %%CPU:0, Rows:1)
   3      INDEX (RANGE SCAN): XI_CUSTOMER (Cost:1, %%CPU:0, Rows:1)
```

```
SQL> SELECT COUNT (*) FROM CUSTOMER WHERE CUST_TEL <= '010-2233-XXXX';
Execution Plan
----------------------------------------------------------------------------
   1  COLUMN PROJECTION (Cost:1, %%CPU:0, Rows:1)
   2    SORT AGGR (Cost:1, %%CPU:0, Rows:1)
   3      FILTER (Cost:1, %%CPU:0, Rows:10)
   4        INDEX (FULL): XI_CUSTOMER (Cost:1, %%CPU:0, Rows:100)
```

6.4. 테이블 스페이스 암호화

테이블 스페이스 암호화(Tablespace Encryption)는 컬럼 암호화와 마찬가지로 데이터베이스의 데이터를 보호하는 또 다른 방법이다. 컬럼 암호화와 다른 점은 테이블 또는 테이블의 컬럼 단위가 아닌 테이블 스페이스 전체에 대해 암호화 여부와 암호화 알고리즘을 지정한다는 것이다.

테이블 스페이스 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 테이블 스페이스의 모든 데이터 블록에 암호화와 복호화가 발생하기 때문에 테이블 스페이스의 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.

6.4.1. 암호화된 테이블 스페이스 생성

테이블 스페이스를 생성하는 **CREATE TABLESPACE 문**에서 암호화 여부를 지정하는 ENCRYPT 절을 추가하면 그 테이블 스페이스는 암호화된 테이블 스페이스가 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 6.13] 암호화된 테이블 스페이스 생성 - 3DES168 알고리즘 지정

```
SQL> CREATE TABLESPACE encrypted_space
     DATAFILE '/usr/tibero/data/encrypted001.dtf' SIZE 50M
     AUTOEXTEND ON NEXT 1M
     EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K
     ENCRYPTION USING '3DES168'
     DEFAULT STORAGE (ENCRYPT);
Tablespace 'ENCRYPTED_SPACE' created.
```

암호화된 테이블 스페이스를 생성할 때에는 다음과 같은 사항을 주의한다.

- 암호화 알고리즘 지정

  테이블 스페이스 암호화에 사용할 수 있는 암호화 알고리즘의 종류가 컬럼 암호화에 비해 적다.

3DES168, AES128, AES192, AES256 이외의 암호화 알고리즘을 지정하게 되면 암호화된 테이블 스페이스는 생성할 수 없다.

- 보안 지갑 열기

  암호화된 테이블 스페이스를 생성하기 전에 반드시 보안 지갑이 열려 있어야 한다. 보안 지갑이 열리지 않은 상태에서 CREATE TABLESPACE 문을 실행하게 되면 다음의 예처럼 에러가 발생하게 되고 테이블 스페이스와 데이터 파일이 생성되지 않게 된다. 따라서 이를 해결하기 위해서는 보안 지갑을 열고 다시 시도하면 된다.

[예 6.14] 암호화된 테이블 스페이스 - 생성 실패

```
SQL> CREATE TABLESPACE encrypted_space
     DATAFILE '/usr/tibero/data/encrypted001.dtf' SIZE 50M
     AUTOEXTEND ON NEXT 1M
     EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K
     ENCRYPTION USING '3DES168'
     DEFAULT STORAGE (ENCRYPT);
TBR-12073: wallet not opened.
```

6.4.2. 암호화된 테이블 스페이스 변경

암호화된 테이블 스페이스의 저장 공간이 더 필요한 경우 **ALTER TABLESPACE 문**의 **ADD DATAFILE 절**을 이용하여 새로운 데이터 파일을 테이블 스페이스에 추가할 수 있다. 새로 추가된 데이터 파일은 처음 테이블 스페이스를 생성했을 때 지정한 암호화 여부와 암호화 알고리즘의 속성을 그대로 가지게 된다.

[예 6.15] 암호화된 테이블 스페이스 - 데이터 파일 추가

```
SQL> ALTER TABLESPACE encrypted_space
     ADD DATAFILE '/usr/tibero/data/encrypted002.dtf' SIZE 50M;
Tablespace 'ENCRYPTED_SPACE' altered.
```

일반 테이블 스페이스를 암호화된 테이블 스페이스로 바꾸거나 암호화된 테이블 스페이스를 일반 테이블 스페이스로 바꾸는 것은 불가능하다. 또한 한 테이블 스페이스에 포함된 데이터 파일에 대해서도 암호화 여부와 암호와 알고리즘을 다르게 지정하는 것도 불가능하다. 따라서 테이블 스페이스를 생성할 때에는 이러한 사항을 고려하고 신중히 결정해야 한다. 이러한 사항에도 일반 테이블 스페이스를 암호화된 테이블 스페이스로 바꾸기 위해서는 새로 암호화된 테이블 스페이스를 만든 다음 일반 테이블 스페이스에 포함된 데이터를 모두 암호화된 테이블 스페이스로 옮겨야 한다.

6.4.3. 암호화된 테이블 스페이스 사용

테이블 또는 인덱스를 생성할 때 암호화된 테이블 스페이스를 지정하면 해당 세그먼트는 물리적으로 그 테이블 스페이스에 저장되고 데이터 블록을 읽고 쓰는 과정에서 자동으로 데이터 암호화와 복호화가 발생한다.

[예 6.16] 암호화된 테이블 스페이스 - 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 테이블 스페이스에 테이블을 생성하고 SQL 문장을 통해 데이터를 조회하고 갱신하는 과정 동안에는 보안 지갑은 항상 열려 있어야 한다. 보안 지갑이 열리지 않은 상태에서 암호화된 테이블 스페이스가 포함된 동작을 수행하게 되면 [예 6.14]와 같은 에러가 발생한다.

테이블 스페이스 중에서 SYSTEM, UNDO, TEMP 테이블 스페이스는 암호화 여부를 지정할 수 없다. 즉, 모든 테이블 스페이스를 암호화할 수 있는 것은 아니다. 또한 Redo 로그 파일에 대해서도 암호화 여부를 지정할 수 없다. 하지만 암호화된 테이블 스페이스에 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 테이블 스페이스에 임시로 저장되는 데이터가 암호화된 테이블 스페이스에 속한 것이라면 TEMP 영역도 자동으로 암호화하고 복호화된다. 따라서 데이터베이스의 다른 파일(Undo, Redo, TEMP 등)을 통해 암호화된 테이블 스페이스의 데이터가 일부라도 노출되지 않도록 보호할 수 있다.

6.4.4. 암호화된 테이블 스페이스 정보 조회

Tibero에서는 암호화된 테이블 스페이스의 정보를 관리하기 위해 다음 표에 나열된 뷰를 제공하고 있다.

뷰	설명
DBA_TABLESPACES	테이블 스페이스의 전체 정보를 조회하는 뷰이다. ENCRYPTED 컬럼을 통해 암호화 여부를 조회할 수 있다.
V$ENCRYPTED_TABLESPACES	암호화된 테이블 스페이스의 정보만을 조회하는 뷰이다. 테이블 스페이스 ID와 암호화 알고리즘을 조회할 수 있다.

참고

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

6.4.5. 암호화된 테이블 스페이스의 암호화 컬럼에 대한 인덱스

암호화된 테이블 스페이스를 이용한 암호화 컬럼에 대하여 일반 인덱스 생성은 NO_SALT 암호화 컬럼에만 가능하고 범위 조건에 대한 INDEX RANGE SCAN이 안되는 제약사항이 있다. 이런 제약사항이 없으면서 데이타 보안은 제공하기 위해 암호화된 테이블 스페이스를 이용한 암호화된 컬럼에 대한 인덱스를 제공한다.

[예 6.17] 암호화된 테이블 스페이스를 이용한 암호화 컬럼에 대한 인덱스

```
SQL> CREATE TABLE customer (
     cust_id       CHAR(4) CONSTRAINT cust_id_pk PRIMARY KEY NOT NULL,
     cust_name     VARCHAR(20) NULL,
     cust_type     VARCHAR(18) NULL,
     cust_addr     VARCHAR(40) ENCRYPT USING 'AES256' NULL,
     cust_tel      VARCHAR(15) ENCRYPT NO SALT NULL,
     reg_date      DATE NULL
     );
     Table 'CUSTOMER' created.

SQL> CREATE TABLESPACE encrypted_space
     DATAFILE '/usr/data/encrypted001.dtf' SIZE 50
     EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K
     ENCRYPTION USING '3DES168'
     DEFAULT STORAGE (ENCRYPT);
     Tablespace 'ENCRYPTED_SPACE' created.

SQL> CREATE INDEX XI_CUSTOMER_01 ON CUSTOMER (CUST_ADDR) ENABLE RANGE;
     TBR-7002: Unsupported DDL

SQL> CREATE INDEX XI_CUSTOMER_01 ON CUSTOMER (CUST_ADDR)
     TABLESPACE ENCRYPTED_SPACE ENABLE RANGE;
     Index 'XI_CUSTOMER_01' created.

SQL> CREATE INDEX XI_CUSTOMER_02 ON CUSTOMER (CUST_TEL)
     TABLESPACE ENCRYPTED_SPACE ENABLE RANGE;
     Index 'XI_CUSTOMER' created.

SQL> SELECT COUNT (*) FROM CUSTOMER WHERE CUST_TEL <= '010-2233-XXXX';
     Execution Plan
   ----------------------------------------------------------------------------


     1  COLUMN PROJECTION (Cost:1, %%CPU:0, Rows:1)
     2    SORT AGGR (Cost:1, %%CPU:0, Rows:1)
    3      INDEX (RANGE SCAN): XI_CUSTOMER_02 (Cost:1, %%CPU:0, Rows:1)
```

6.5. HSM 연동을 통한 키 분리

HSM(Hardware Security Module)은 데이터 암호화 및 복호화에 사용되는 키를 안전하게 저장/관리할 수 있도록 제작된 전용 하드웨어 장비이다. HSM 장비와의 연동을 통하여 Tibero에서 테이블 및 테이블 스페이스 암호화를 위해 사용되는 마스터키를 외부 장비로 분리할 수 있다. 해당 기능을 사용 시 데이터 암호화 키가 DB와는 독립적으로 보관되어 관리되기 때문에 더 높은 수준의 보안을 기대할 수 있다.

현재는 Linux와 AIX만 지원하며, HSM 장비 지원 목록은 다음과 같다.

- D'Amo KMS (펜타시큐리티)
- Vormetric Data Security Manager (탈레스)

6.5.1. 환경설정

데이터 암호화 기능을 사용하기 위해서는 우선 먼저 보안 지갑을 생성해야 한다. 보안 지갑의 생성 및 사용 방법에 대해서는 “6.2. 환경설정”에 서술되어 있다.

HSM 장비와의 연동을 위해서 필요한 설정은 다음과 같다.

1. **라이브러리 세팅**

   연동하고자 하는 HSM 장비에 따라서 필요한 라이브러리 파일이 다를 수 있다. 해당 파일들은 HSM 장비를 관리하는 각 업체를 통해서 제공받아야하며 제공 받은 파일을 $TB_HOME/lib에 배치하면 된다. 각각의 장비에 따라 필요한 라이브러리 파일은 다음과 같다.

HSM 장비	라이브러리 파일명
D'Amo KMS	libsgkms_crytoki.so
Vormetric Data Security Manager	libvorpkcs11.so

주의

현 문서는 HSM 장비를 이미 정상적으로 설치 및 구축해두었다는 가정 하에 서술 되어있다. 따라서 각 HSM 장비의 설치 및 라이브러리에 대한 추가적인 문의 사항이 있다면 이는 해당 HSM 장비의 담당 업체에 따로 문의하여야 한다.

2. **파라미터 설정**

$TB_SID.tip 파일에 HSM 연동 관련 초기화 파라미터들을 추가 작성하여 보안 지갑의 위치를 설정한다.

초기화 파라미터	설명
USE_HSM	HSM 장비 연동을 통한 키 분리 기능의 사용 여부를 결정한다.
HSM_LIB_FILE	HSM 키 분리 기능을 사용하는 경우 사용하는 장비에 맞는 라이브러리 파일의 위치를 설정한다. 해당 파라미터는 반드시 절대경로로 작성되어야 한다.
HSM_NOT_USE_WRAPPING	HSM 키 분리 기능을 사용하는 경우 키를 wrapping할 것인지 여부를 결정한다. (Vormetric Data Security Manager 사용 시에만 작성)
HSM_TTE_KEY_BACKUP_FILE	만약 기존에 테이블스페이스 암호화 기능(TTE)를 사용 중이었던 경우, HSM 키 분리 기능을 활성/비활성화 하기 위해서는 테이블스페이스를 새로운 키로 다시 암호화하기 위한 작업이 필요하다. 작업 중 문제가 생길 때를 대비하여, 기존 TTE 키를 백업하기 위한 경로를 설정한다. 해당 파라미터는 반드시 절대경로로 작성되어야 한다.

[예 6.18] D'Amo KMS 장비와 연동하는 경우: <<$TB_SID.tip>>

```
WALLET_FILE=/path/to/TBWALLET
USE_HSM=Y

#절대경로로 라이브러리 파일명까지 포함하여 작성
HSM_LIB_FILE="/path/to/hsm_library/libsgkms_crytoki.so"

#절대경로로 백업 파일명까지 포함하여 작성
HSM_TTE_KEY_BACKUP_FILE="/path/to/tte_key_backup/tte_key.backup"
```

[예 6.19] Vormetric Data Security Manager 장비와 연동하는 경우: <<$TB_SID.tip>>

```
WALLET_FILE=/path/to/TBWALLET
USE_HSM=Y
HSM_NOT_USE_WRAPPING=Y

#절대경로로 라이브러리 파일명까지 포함하여 작성
HSM_LIB_FILE="/path/to/hsm_library/libvorpkcs11.so"

#절대경로로 백업 파일명까지 포함하여 작성
HSM_TTE_KEY_BACKUP_FILE="/path/to/tte_key_backup/tte_key.backup"
```

6.5.2. 사용 방법

HSM 장비를 통해 데이터 암호화 키를 분리한 뒤, 이를 이용한 데이터 암호화 기능을 활성화하고자 하면 DBA 권한을 가진 사용자가 다음의 명령을 수행해야 한다. 이 때, HSM 장비에 따라 IDENTIFIED 절 뒤에

입력하는 아이디와 USING 절 뒤에 입력하는 패스워드로 입력해야 하는 값이 다르기 때문에 이 점 유의해야한다. 각 장비에 따른 명령은 다음과 같다.

[예 6.20] Vormetric Data Security Manager를 이용한 키 분리 이후 데이터 암호화 기능 활성화

```
SQL> alter system set encryption wallet open identified by '[Vormetric_PIN값]'
migrate using '[WALLET password]';
System altered.
```

[예 6.21] D'Amo KMS를 이용한 키 분리 이후 데이터 암호화 기능 활성화

```
SQL> alter system set encryption wallet open identified by 'NULL' migrate using
'[WALLET password]';
System altered.
```

데이터 암호화 기능을 사용하지 못하도록 비활성화하려면, DBA 권한을 가진 사용자가 다음의 명령을 수행해야 한다.

[예 6.22] HSM 키 분리 이후 데이터 암호화 기능 비활성화

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;
System altered.
```

주의

보안 지갑과 마찬가지로 데이터베이스 인스턴스를 종료하면 데이터 암호화 기능이 비활성화되므로 다시 기동할 때마다 활성화 해줘야한다.

주의

마스터키가 변경되는 경우 미디어 복구는 지원하지 않는다.

제7장 통신 암호화

본 장에서는 Tibero에서 통신 암호화 기능을 사용하고 관리하기 위한 방법을 설명한다.

7.1. 개요

Tibero는 서버와 클라이언트 간에 송수신되는 메시지에 대한 기밀성을 보장하기 위하여 통신 암호화 기능을 제공한다. Tibero의 통신 암호화 기능은 Netscape사의 SSL 통신 프로토콜을 채택하였으며 Openssl Project에서 제공하는 라이브러리를 사용하여 개발되었다. 참고로 Windows 운영체제의 경우 현재 통신 암호화 기능을 제공하지 않는다.

7.2. 환경설정

통신 암호화 기능을 사용하기 위해서는 통신 양자 간에 보안 통신을 위한 설정이 필요하다. 서버에서는 보안 통신에 사용될 인증서와 개인 키를 미리 소지하고 있거나 없으면 새로 생성해야 한다. 그 다음 인증서와 개인 키의 위치를 환경설정 파일에 지정해주게 되면 보안 통신 전용 포트가 열리게 된다.

클라이언트에서는 보안 통신 사용 여부를 환경설정 파일에 지정해준다. 이에 따라 일반 또는 보안 접속이 이루어지게 된다.

7.2.1. 개인 키 및 인증서 생성

개인 키 및 인증서 생성은 X.509 v3(PKI ITU-T 표준)을 따른다. 개인키 및 인증서를 생성할 때 입력하는 주요 정보는 다음과 같다.

- password

  해당 암호는 개인키 및 인증서를 생성하는 데에만 쓰인다.

- basename to make a certificate and a private key

  개인키 및 인증서 파일의 이름을 결정하는데 쓰이며, 이외의 용도는 없다.

- the number of days to make a certificate valid for

  해당 기간 만료 후에는 개인키 및 인증서를 사용할 수 없기 때문에 재생성해야 하며 재생성은 온라인 중에도 가능하다. 세션이 새로 로그인할 때 하나의 ssl 단위가 새로 생성되기 때문에, 온라인 중에 인증서 및 개인키를 갱신할 경우 기존과 동일한 파일명으로 갱신한 후 ssl 옵션으로 세션을 새로 붙이면 된다.

실제 개인 키 및 인증서를 생성하는 방법은 다음과 같다.

[예 7.1] 개인 키 및 인증서의 생성

```
$ cd $TB_HOME/bin
$ ./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 bits 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. 만약 원하는 폴더에 개인 키 및 인증서를 생성하고 싶다면 다음과 같이 -p나 -P 옵션을 통해 생성 가능하다.

```
./tb_cert_manager -p $TB_HOME/config/[생성을 원하는 폴더이름]
```

이때, 경로는 절대경로를 입력한다.

```
공개 키 기반 개인 키: [$TB_SID].key
인증서 생성 요청서: [$TB_SID].csr
인증서: [$TB_SID].crt
```

추가로 TAC인 경우는 노드별로 각각 개인키 및 인증서를 생성할 필요가 없고 한번 만든 개인키 및 인증서를 공유해서 사용할 수 있다. 혹은 -p 또는 -P 옵션으로 개인키 및 인증서를 생성하여 다중 인스턴스에서 사용 가능하다.

7.2.2. 개인 키 및 인증서 위치 설정

$TB_SID.tip 환경설정 파일에 CERTIFICATE_FILE과 PRIVKEY_FILE 초기화 파라미터를 추가하고 인증서와 개인 키의 위치를 절대경로로 설정한다.

[예 7.2] 개인 키 및 인증서의 위치설정

<<$TB_SID.tip>>

```
# 아래의 $TB_HOME을 절대경로로 입력한다.
CERTIFICATE_FILE=$TB_HOME/config/tb_wallet/[$TB_SID].crt
PRIVKEY_FILE=$TB_HOME/config/tb_wallet/[$TB_SID].key
```

7.2.3. 클라이언트 설정

tbdsn.tbr 파일에서 통신 암호화 사용을 설정하는 방법은 다음과 같다.

[예 7.3] 클라이언트 설정

<<tbdsn.tbr>>

```
TB=(
    (INSTANCE=(HOST=서버 IP)
              (PORT=서버 포트)
              (DB_NAME=데이터베이스 이름)
              (USE_SSL=Y)
    )
)
```

SID 내의 USE_SSL값을 'Y'로 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 하게 된다.

제8장 Separation of Duties

본 장에서는 Tibero의 관리자 권한 분할을 위한 특수 기능인 Separation of Duties에 대해서 설명한다.

8.1. 개요

Tibero DBMS는 SYS 하나의 관리자가 데이터베이스에 대한 관리를 총체적으로 하고 있다. 이로 인해, SYS 관리자의 권한이 남용된다거나, 계정이 탈취되는 등 보안상의 문제가 발생하였을 경우 보안 체계가 무너지며 데이터 유출, 조작, 손상 등이 초래될 수 있다.

Tibero Separation of Duties(이하 SOD)는 이러한 보안상의 취약점을 보완하기 위해 기존 SYS 관리자의 권한을 3개로 나누어서 관리자의 역할을 분리한 체계이다. 관리자를 나누어 각 관리자의 관리 업무에 필요한 권한만 할당 되었으며, 분리된 관리자 단독으로 보안 체계를 붕괴할 수 없는 시스템이다.

SOD 시스템을 설치할 때 다음의 3개의 관리자가 생성된다.

- 시스템 관리자(SYSADM)

  데이터베이스 내 테이블 스페이스, 데이터 파일, 스키마 오브젝트 등에 대한 관리를 담당한다.

- 보안 관리자(SYSSEC)

  데이터베이스 내 보안을 관리하며 권한 관리를 담당한다.

- 감사 관리자(SYSAUD)

  데이터베이스 내의 이력을 관리하며 감사를 담당한다.

8.1.1. 시스템 관리자

데이터베이스의 일상 관리 및 사용을 담당하는 관리자이다. 일반적으로 DBA가 하는 역할을 담당하는 관리자이다. 보안, 감사를 제외한 데이터베이스의 유지보수, 정책 및 절차 확립, 백업 및 복구를 담당하며, 담당하는 부분에 대한 설명은 “2.1.1. DBA”를 참고한다.

일반적인 사용자 권한 부여에 대한 권한은 없지만, 기본 권한인 connect(접속 권한), resource(사용자 자원 사용 권한)에 대한 부여는 가능하다. 감사 관련 뷰를 제외한 모든 static view에 조회 가능하다.

8.1.2. 보안 관리자

데이터베이스 내 사용자의 접근 관리 및 권한에 대한 관리를 담당하는 관리자이다. SOD에서는 사용자가 특정 행동을 할 수 있는지에 대한 관리를 보안의 기준으로 삼고 있으며, 특권 및 역할에 대한 권한 관리를

통해서 보안을 관리하게 된다. 특권과 역할에 대한 설명은 "5.2. 특권"과 "5.4. 역할"을 참고한다. **SOD에서는 권한관리를 오직 보안 관리자만이 할 수 있다.**

기존 DBA 역할이 가지는 권한이 보안상의 문제로 축소가 되었으며, 아래의 권한이 철회되었다.

- 철회된 특권 항목
  - CREATE ROLE
  - DROP/GRANT/ALTER ANY ROLE
  - GRANT ANY PRIVILEGE
  - GRANT ANY OBJECT PRIVILEGE
  - AUDIT SYSTEM/ANY

보안 관련된 아래의 정적 뷰들을 조회하여 관리할 수 있다.

뷰	설명
ROLE_ROLE_PRIVS	Role과 관련된 권한을 조회한다.
ROLE_SYS_PRIVS	System authorization과 관련된 권한을 조회한다.
ROLE_TAB_PRIVS	Object authorization과 관련된 권한을 조회한다.

8.1.3. 감사 관리자

데이터베이스의 감사를 담당하는 관리자이다. SOD에서는 감사 관리자와 일반적으로 DBA가 하는 역할을 담당하는 관리자이다. 해당 관리자가 하는 담당하는 부분에 대해서는 "5.6. 감사"를 참고한다.

SOD의 경우 감사 관리자를 제외한 다른 사용자 관리자들은 감사 기능을 사용할 수 없게 된다. 또한, 감사 관리자만이 감사 관련 뷰(DBA_AUDIT_TRAIL)에 대한 조회가 가능하다.

8.2. 설치 방법

다음은 SOD의 설치 방법에 대한 설명이다.

1. SOD 기능을 사용하려면 우선 TIP 파일에 'SEPARATION_OF_DUTIES=Y'를 추가 설정해주어야 한다. 해당 파라미터는 SOD를 사용할 경우에 설치 전에 반드시 추가해 주어야 한다. 해당 파라미터는 동적으로 바꿀 수 없으며, 변경할 때에는 데이터베이스를 내린 후 다시 올려야 한다. 패치와 같이 부득이한 상황이 아닌 이상 해당 파라미터를 변경하지 않는 것을 권장한다.
2. TIP 설정 후 데이터베이스를 mount할 때 옵션 -sod Y를 추가한다. 수동 설치하는 경우에는 데이터베이스를 생성한 이후 system.sh을 수행할 때 옵션 -sod Y를 추가한다.
3. 각 관리자별 권한에 따라 ID와 비밀번호를 설정한다.

   SYS 관리자는 아래의 관리자로 분리되었으며, 각 관리자의 ID와 초기 비밀번호는 다음과 같다.

관리자명	관리자 아이디	기본 암호
시스템 관리자	SYSADM	SYSADM
보안 관리자	SYSSEC	SYSSEC
감사 관리자	SYSAUD	SYSAUD

참고

해당 암호는 추후 제품 설치 후에 각 관리자가 변경하는 것을 권장한다.

8.3. 변경 및 주의 사항

다음은 SOD를 사용하는데 주의사항이다.

- SOD는 Tibero에서 별도로 제공하고 있는 기능이고, 특성 상 제품 설치할 때에만 기능을 활성화 할 수 있다. 즉, 이미 설치되어 있는 데이터베이스에 해당 기능을 사용할 수 없다. 이는 제품을 설치할 때에 3 분류의 관리자로 설정해야 하는 특성 때문이다. 해당 기능을 효율적으로 사용하기 위해서는 각각의 관리자를 세 명이서 나누어야 한다.
- 해당 기능을 사용하면 기존의 SYS는 사용 불가능하게 된다(계정이 잠겨있으며, 잠금을 풀 수 없도록 설정되어 있음). SYS 관련 정보는 static view에서도 보여지지 않는다.

제9장 Virtual Private Database

본 장에서는 Tibero의 보안상 혹은 특정 서비스를 목적으로 필요한 유저에 해당하는 데이터만 다룰 수 있게 하는 기술인 Virtual Private Database에 대해서 설명한다.

9.1. 개요

Virtual Private Database는 행과 열 수준에서 데이터베이스 접근을 제어하는 보안 정책을 생성한다.

기본적으로 Virtual Private Database는 보안 정책이 적용된 테이블, 뷰, 또는 동의어에 대해 실행되는 SQL 문에 동적 WHERE 절을 추가한다. Virtual Private Database는 테이블, 뷰, 동의어에 대해 직접 세부 수준까지 보안을 강화한다. 보안 정책을 이러한 데이터베이스 객체에 직접 연결하고 사용자가 데이터에 접근할 때 마다 정책이 자동으로 적용되기 때문에 보안을 우회할 방법이 없다.

사용자가 Virtual Private Database 정책을 보호되는 테이블, 뷰, 동의어를 직접 또는 간접적으로 접근할 때 데이터베이스는 사용자의 SQL 문을 동적으로 수정한다. 이 수정은 보안 정책을 구현하는 함수가 반환하는 WHERE 문을 생성한다. 이러한 Virtual Private Database 정책은 SELECT, INSERT, UPDATE, DELETE 문에 적용할 수 있다.

9.2. Virtual Private Database의 이점

Virtual Private Database는 보안성, 단순성 및 유연성 면에서 이점을 제공한다.

모든 애플리케이션에서 접근 제어를 구현하는 대신 Virtual Private Database 보안 정책을 활용하면 다음과 같은 이점이 있다.

- 보안성

  Virtual Private Database는 심각한 애플리케이션 보안 문제를 해결할 수 있다. 세분화된 접근 제어는 어떻게 사용자가 데이터에 접근하든 똑같은 보안 정책을 적용시킬 수 있다. 데이터에 대한 원천적인 보안 설정이 가능하다.

- 단순성

  특정 테이블, 뷰, 동의어를 위한 보안 설정을 할 필요 없이 보안 정책을 한번에 설정할 수 있다.

- 유연성

  SELECT, INSERT, UPDATE, DELETE에 따라 다른 보안 정책을 만들 수 있다. 또한 테이블, 뷰, 동의어에 여러 정책을 생성할 수 있다.

9.3. Virtual Private Database의 구성 요소

Virtual Private Database를 만들려면 동적 WHERE 문을 생성하는 정책 함수와 함수를 객체로 연결하는 보안 정책이 필요하다.

- 정책 함수

  정책 함수는 다음과 같은 조건을 만족해야 한다.

  – 스키마 이름과 객체 이름을 인수로 입력해야 한다.

  – WHERE 문의 조건문을 항상 반환해야 한다. 항상 반환 값은 VARCHAR 타입이여야 한다.

  – 유효한 WHERE 문을 생성해야 한다.

  – 연관된 정책 함수 내의 테이블을 선택하면 안된다.

  – 연관 되지 않는 함수이여야 한다. 함수는 패키지 변수에 종속되지 않아야 한다.

  – 예제

```
CREATE OR REPLACE FUNCTION F_vpd_select(schema varchar2, object varchar2)
RETURN varchar2
IS
BEGIN
    RETURN 'mod( c1, 2) != 0';
END;
/
```

- 보안 정책

  DBMS_RLS 패키지를 통해서 보안 정책을 만들 수 있다. 사용자는 SYS 계정이거나 DBMS_RLS 패키지를 EXECUTE 할 수 있는 권한이 있어야 한다. DBMS_RLS.ADD_POLICY로 정책을 만들 수 있다.

  – 예제

```
BEGIN
    DBMS_RLS.ADD_POLICY(
        object_name => 'T_vpd_select',
        policy_name => 'policy_select',
        policy_function => 'F_vpd_select',
        statement_types => 'select'
    );
END;
/
```

9.4. Virtual Private Database의 구성

Virtual Private Database를 제대로 구성하기 위해서는 Application Context와 Policy를 알아야 한다.

9.4.1. Application Context

Application Context는 사용자 식별 정보를 저장해준다. 이 사용자 식별 정보를 이용해서 데이터 접근 권한을 관리할 수 있다. Virtual Private Database는 정책 함수에 Application Context를 적용하여 사용자에 따라서 다른 결과를 보여줄 수 있다.

Application Context 구성

- namespace : Application Context의 이름 (예: orders_ctx)
- name : 속성의 이름 (예: cust_no)
- value : 속성의 값 (예: 1)

CREATE CONTEXT

문맥 네임스페이스를 생성(혹은 수정)하고 해당 네임스페이스의 문맥을 조작할 수 있는 패키지를 지정한다. 지정된 패키지에서 DBMS_SESSION.SET_CONTEXT 프러시저를 사용해 해당 네임스페이스의 속성 값들을 설정할 수 있다.

- 예제

  orders_ctx란 Application Context를 생성한다. orders_ctx_proc란 프러시저로 orders_ctx를 보안 정책을 구현할 수 있다.

```
CREATE CONTEXT orders_ctx USING orders_ctx_proc;
```

참고

CREATE CONTEXT에 대한 자세한 내용은 "Tibero SQL 참조 안내서"의 "7.26 CREATE CONTEXT"를 참고한다.

DBMS_SESSION.SET_CONTEXT

이 프러시저는 CREATE CONTEXT DDL을 통해 지정된 패키지 내에서만 호출할 수 있다. 설정된 속성 값은 세션이 유지되는 동안 보존된다. 만일 해당 문맥의 속성이 이미 설정되어 있을 경우 SET_CONTEXT 호출은 해당 속성값을 덮어쓴다.

- 예제

  orders_ctx_proc 프러시저를 정의한다. 패키지 프러시저 안에 DBMS_SESSION.SET_CONTEXT를 활용해서 orders_ctx의 Application Context를 설정한다.

```
create or replace procedure orders_ctx_proc(
val in VARCHAR2)
as
begin
    dbms_session.set_context ('orders_ctx', 'cust_no', val);
end;
/

EXEC orders_ctx_proc('1');
```

참고

DBMS_SESSION.SET_CONTEXT에 대한 자세한 내용은 "Tibero tbPSM 참조 안내서"의 "36.2.5 SET_CONTEXT"를 참고한다.

SYS_CONTEXT

SYS_CONTEXT는 문맥 네임스페이스(CONTEXT NAMESPACE)와 관련된 파라미터의 값을 반환하는 함수이다. 문맥 네임스페이스와 파라미터는 문자열이나 표현식으로 정의할 수 있으며, 함수의 반환값은 VARCHAR 타입이다. Tibero에서 디폴트로 제공하고 있는 문맥 네임스페이스는 USERENV이다.

- 예제

  orders_ctx의 cust_no 속성의 값을 반환한다.

```
SQL> SELECT SYS_CONTEXT('orders_ctx', 'cust_no') cust_no
     FROM DUAL;

CUST_NO
-------
1

1 row selected.
```

참고

SYS_CONTEXT에 대한 자세한 내용은 "Tibero SQL 참조 안내서"의 "4.2.159 SYS_CONTEXT"를 참고한다.

9.4.2. Policy

Virtual Private Database의 정책은 정책 함수에 따라 WHERE 문을 바꿔주며 이 정책을 구성하기 위해선 다음 DBMS_RLS 패키지를 활용한다.

Procedure	설명
DBMS_RLS.ADD_POLICY	테이블, 뷰, 동의어에 정책을 추가한다.
DBMS_RLS.DROP_POLICY	테이블, 뷰, 동의어에 걸려있는 보안 정책을 제거한다.
DBMS_RLS.ENABLE_POLICY	테이블, 뷰, 동의어에 걸려있는 보안 정책을 활성화 또는 비활성화한다.
DBMS_RLS.REFRESH_POLICY	해당 보안 정책으로 인해 영향을 받았던 실행 계획들과 메모리에 저장되어 있던 정책 함수의 결과들을 모두 무효화 시킨다.

참고

DBMS_RLS에 대한 자세한 내용은 "Tibero tbPSM 참조 안내서"의 "제33장 DBMS_RLS"를 참고한다.

9.5. Virtual Private Database 생성 예제

본 절에서는 Application Context 없는 간단한 VPD와 Application Context 있는 VPD를 생성하는 간단한 예제를 설명한다.

9.5.1. Application Context 없는 VPD

다음은 Application Context 없는 VPD를 생성하는 과정에 대한 설명이다.

1. User 생성 및 권한 부여

```
CONN sys/tibero;
CREATE USER user1 IDENTIFIED BY tibero;
GRANT CREATE SESSION TO user1;
```

2. Table 생성

```
CREATE TABLE orders
(
order_no number(4),
item varchar(256)
);
INSERT INTO orders VALUES(1, 'apple');
INSERT INTO orders VALUES(1, 'banana');
INSERT INTO orders VALUES(2, 'candy');
```

```
INSERT INTO orders VALUES(2, 'dog');
INSERT INTO orders VALUES(3, 'elephant');
INSERT INTO orders VALUES(3, 'frog');
INSERT INTO orders VALUES(4, 'giraffe');
INSERT INTO orders VALUES(4, 'hawk');
GRANT SELECT on orders TO user1;
```

3. 정책 함수 생성

```
CREATE OR REPLACE FUNCTION policy_func1(
  schema_var IN VARCHAR2,
  table_var  IN VARCHAR2
 )
 RETURN VARCHAR2
 IS
  return_val VARCHAR2 (400);
 BEGIN
  return_val := 'order_no = 1';
  RETURN return_val;
 END policy_func1;
/
```

4. VPD 정책 생성

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'sys',
    object_name      => 'orders',
    policy_name      => 'cust_policy',
    function_schema  => 'sys',
    policy_function  => 'policy_func1',
    statement_types  => 'select'
   );
 END;
/
```

5. 정책 확인

```
CONN user1/tibero;
SELECT * FROM sys.orders;

  ORDER_NO ITEM
---------- --------------------
         1 apple
         1 banana

2 rows selected.
```

6. 예제 객체 제거

```
CONN sys/tibero;
DROP FUNCTION policy_func1;
EXEC DBMS_RLS.DROP_POLICY('SYS','ORDERS','CUST_POLICY');
DROP TABLE orders;
DROP USER user1;
```

9.5.2. Application Context 있는 VPD

다음은 Application Context 있는 VPD를 생성하는 과정에 대한 설명이다.

1. User 생성 및 권한 부여

```
CONN SYS/TIBERO;
CREATE USER sys_vpd IDENTIFIED BY tibero;
GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE,
CREATE TRIGGER, ADMINISTER DATABASE TRIGGER TO sys_vpd;
GRANT EXECUTE ON DBMS_SESSION TO sys_vpd;
GRANT EXECUTE ON DBMS_RLS TO sys_vpd;
CREATE USER user1 IDENTIFIED BY tibero;
CREATE USER user2 IDENTIFIED BY tibero;
CREATE USER user3 IDENTIFIED BY tibero;
CREATE USER manager IDENTIFIED BY tibero;
GRANT CREATE SESSION TO user1;
GRANT CREATE SESSION TO user2;
GRANT CREATE SESSION TO user3;
GRANT CREATE SESSION TO manager;
GRANT CONNECT, RESOURCE TO manager;
```

2. Table 생성

```
CONN manager/tibero;
CREATE TABLE customers
(
cust_no number(4),
cust_id varchar(20),
cust_name varchar(20)
);
INSERT INTO customers VALUES (1, 'USER1', 'Park');
INSERT INTO customers VALUES (2, 'USER2', 'Kim');
INSERT INTO customers VALUES (3, 'USER3', 'Lee');
GRANT SELECT ON customers TO sys_vpd;

CREATE TABLE orders
(
```

```
order_no number(4),
user_id varchar(256),
item varchar(256)
);
INSERT INTO orders VALUES(1, 'USER1', 'apple');
INSERT INTO orders VALUES(1, 'USER1', 'banana');
INSERT INTO orders VALUES(2, 'USER2', 'candy');
INSERT INTO orders VALUES(2, 'USER2', 'dog');
INSERT INTO orders VALUES(3, 'USER3', 'elephant');
INSERT INTO orders VALUES(3, 'USER3', 'frog');
INSERT INTO orders VALUES(4, 'USER4', 'giraffe');
INSERT INTO orders VALUES(4, 'USER4', 'hawk');
GRANT SELECT ON orders TO user1;
GRANT SELECT ON orders TO user2;
GRANT SELECT ON orders TO user3;
```

3. Application Context 생성

```
CONN sys_vpd/tibero;
CREATE OR REPLACE CONTEXT orders_ctx USING orders_ctx_pkg;
```

4. Application Context 생성을 위한 PL/SQL Package 생성

```
CREATE OR REPLACE PACKAGE orders_ctx_pkg IS
  PROCEDURE set_custnum;
 END;
/
CREATE OR REPLACE PACKAGE BODY orders_ctx_pkg IS
  PROCEDURE set_custnum
  AS
    custnum NUMBER;
  BEGIN
     SELECT cust_no INTO custnum FROM MANAGER.CUSTOMERS
        WHERE cust_id = SYS_CONTEXT('USERENV', 'SESSION_USER');
     DBMS_SESSION.SET_CONTEXT('orders_ctx', 'cust_no', custnum);
  EXCEPTION
   WHEN NO_DATA_FOUND THEN NULL;
  END set_custnum;
END;
/
```

5. Logon Trigger 생성

```
CREATE TRIGGER set_custno_ctx_trig AFTER LOGON ON DATABASE
 BEGIN
  sys_vpd.orders_ctx_pkg.set_custnum;
```

```
 END;
/
```

6. Logon Trigger 테스트

```
CONN user1/tibero;
SELECT SYS_CONTEXT('orders_ctx', 'cust_no') custnum FROM DUAL;

CUSTNUM
----------------------------------------------------------------
1

1 row selected.

CONN user2/tibero;
SELECT SYS_CONTEXT('orders_ctx', 'cust_no') custnum FROM DUAL;

CUSTNUM
----------------------------------------------------------------
2

1 row selected.

CONN user3/tibero;
SELECT SYS_CONTEXT('orders_ctx', 'cust_no') custnum FROM DUAL;

CUSTNUM
----------------------------------------------------------------
3

1 row selected.
```

7. 정책 함수 생성

```
CONN sys_vpd/tibero;
CREATE OR REPLACE FUNCTION get_user_orders(
  schema_p   IN VARCHAR2,
  table_p    IN VARCHAR2)
 RETURN VARCHAR2
 AS
  orders_pred VARCHAR2 (400);
 BEGIN
  orders_pred := 'order_no = SYS_CONTEXT(''orders_ctx'', ''cust_no'')';
 RETURN orders_pred;
END;
/
```

8. VPD 정책 생성

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'manager',
    object_name      => 'orders',
    policy_name      => 'cust_policy',
    function_schema  => 'sys_vpd',
    policy_function  => 'get_user_orders',
    statement_types  => 'select');
 END;
/
```

9. 정책 확인

```
CONN user1/tibero;
SELECT * FROM manager.orders;
  ORDER_NO USER_ID              ITEM
---------- -------------------- --------------------
         1 USER1                apple
         1 USER1                banana

2 rows selected.

CONN user2/tibero;
SELECT * FROM manager.orders;

  ORDER_NO USER_ID              ITEM
---------- -------------------- --------------------
         2 USER2                candy
         2 USER2                dog

2 rows selected.

CONN user3/tibero;
SELECT * FROM manager.orders;

  ORDER_NO USER_ID              ITEM
---------- -------------------- --------------------
         3 USER3                elephant
         3 USER3                frog

2 rows selected.
```

10. 예제 객체 제거

```
CONN sys_vpd/tibero;
DROP FUNCTION get_user_orders;
EXEC DBMS_RLS.DROP_POLICY('MANAGER','ORDERS','CUST_POLICY');
DROP CONTEXT orders_ctx;

CONN manager/tibero;
DROP TABLE orders;
DROP TABLE customers;

CONN sys/tibero;
DROP USER user1;
DROP USER user2;
DROP USER user3;
DROP USER sys_vpd;
DROP USER manager;
```

제10장 Tibero Label Security

본 장에서는 레이블을 통하여 사용자의 접근을 제어하는 기능인 Label Security에 대해서 설명한다.

10.1. 개요

Label Security는 특정 테이블의 열 및 사용자에게 할당 된 레이블을 이용하여 대상 테이블의 행에 대한 접근을 제어하는 기능이다.

기본적으로 Label Security는 행 레이블과 사용자 레이블을 비교하여 사용자의 접근을 제어하는 방식으로 동작하며, 민감한 정보를 승인된 사용에게만 접근 가능하도록 설정할 수 있다.

10.2. Label Security의 구성 요소

Label Security 기능을 사용하기 위해선 접근 제어 지표인 레이블과 접근 제어 관련 정보를 관리하는 정책이 필요하다.

- 레이블

  레이블은 다음과 같은 조건을 만족해야 한다.

구분	설명
Level	데이터의 민감한 정도를 수직적으로 나타낸다. (예: 공개, 대외비, 기밀)
Compartment	데이터의 구성을 수평적으로 나타낸다. 개별 프로젝트에 대한 접근을 제어하는 데 도움이 된다. (예: 알파, 베타, 감마)
Group	데이터의 구성을 tree 형태로 나타낸다. 계층형 구조의 데이터에 대한 접근을 제어하는 데 도움이 된다. (예: 티맥스 – 티맥스데이터, 티맥스소프트)

- 정책

  접근 제어 관련 정보를 관리하는 메인 컨테이너이다.

예를 들어 데이터의 민감도가 공개자료, 대외비, 기밀 3 단계의 level로 이루어져 있을 때 특정 사용자에게 대외비까지 접근할 수 있는 level이 부여되었다면 그 사용자가 기밀 데이터에 접근하는 것은 불가능하다.

10.3. 레이블에 대한 이해

Label Security 기능을 사용하기 위해서는 레이블 기반 보안 방식에 및 레이블 구성 요소에 대한 이해가 필요하다.

10.3.1. 레이블 기반 보안 방식

레이블은 크게 데이터 레이블과 사용자 레이블로 나누어 생각할 수 있다.

- 데이터 레이블

  행 데이터의 민감한 정도와 특성을 나타내며, 사용자가 해당 행에 접근하기 위해 충족해야 하는 기준이 된다.

- 사용자 레이블

  사용자의 민감한 정도와 레이블이 지정된 데이터에 대한 사용자의 접근을 제한하는 기준이 된다. 각 사용자에게는 레벨, 구획, 그룹의 범위가 할당되며 해당 범위 내의 레이블이 있는 데이터에 접근 가능하다.

10.3.2. 레이블 구성 요소

레이블을 구성하는 요소는 레벨, 구획, 그룹으로 나뉜다. 해당 요소들을 적절하게 결합하여 레이블을 생성할 수 있다.

레벨 구성 요소

레벨은 레이블 정보의 민감도를 등급화 한 지표이다. 큰 레벨 값은 정보의 민감도가 높다는 것을 의미한다.

다음과 같이 레벨을 정의할 수 있다.

숫자 형식	긴 형식	짧은 형식
30	기밀 (SECRET)	S
20	대외비 (CONFIDENTIAL)	C
10	공개 (PUBLIC)	P

구획 구성 요소

구획은 레이블이 지정된 데이터 정보의 민감도를 더 세분화하는 영역이다. 특정 프로젝트와 관련 있는 모든 데이터는 동일한 구획으로 레이블을 지정할 수 있다.

다음과 같이 구획을 정의할 수 있다.

숫자 형식	긴 형식	짧은 형식
80	알파 (ALPHA)	A
50	베타 (BETA)	B

숫자 형식	긴 형식	짧은 형식
20	감마 (GAMMA)	G

그룹 구성 요소

그룹은 계층적 구조로 레이블이 지정된 데이터 정보의 민감도를 더 세분화하는 영역이다. 조직 구성을 기반으로 데이터에 레이블을 지정할 수 있으며, 특정 부서와 관련있는 모든 데이터는 레이블에 해당 부서의 그룹을 포함할 수 있다.

다음과 같이 그룹을 정의할 수 있다.

숫자 형식	긴 형식	짧은 형식	부모 그룹
1000	TMAX	T	
1100	TMAX_DATA	T_DATA	T
1200	TMAX_SOFT	T_SOFT	T

10.3.3. 레이블 구문 유형

레이블 구성 요소를 정의한 후 특정 레벨, 구획, 그룹 집합을 결합하여 데이터 레이블을 생성할 수 있다.

다음과 같은 구문 형태로 레이블 문자열 표현이 가능하다.

```
LEVEL:COMPARTMENT1,…,COMPARTMENTn:GROUP1,…,GROUPn
```

```
SECRET:ALPHA,BETA:TMAX_DATA
CONFIDENTIAL:GAMMA:TMAX
PUBLIC
SECRET::TMAX_SOFT
```

레벨 구성 요소

레벨은 레이블 정보의 민감도를 등급화한 지표이다. 큰 레벨 값은 정보의 민감도가 높다는 것을 의미한다.

다음과 같이 레벨을 정의할 수 있다.

숫자 형식	긴 형식	짧은 형식
30	기밀 (SECRET)	S
20	대외비 (CONFIDENTIAL)	C
10	공개 (PUBLIC)	P

구획 구성 요소

구획은 레이블이 지정된 데이터 정보의 민감도를 더 세분화하는 영역이다. 특정 프로젝트와 관련 있는 모든 데이터는 동일한 구획으로 레이블을 지정할 수 있다.

다음과 같이 구획을 정의할 수 있다.

숫자 형식	긴 형식	짧은 형식
80	알파 (ALPHA)	A
50	베타 (BETA)	B
20	감마 (GAMMA)	G

그룹 구성 요소

그룹은 계층적 구조로 레이블이 지정된 데이터 정보의 민감도를 더 세분화하는 영역이다. 조직 구성을 기반으로 데이터에 레이블을 지정할 수 있으며, 특정 부서와 관련있는 모든 데이터는 레이블에 해당 부서의 그룹을 포함할 수 있다.

다음과 같이 그룹을 정의할 수 있다.

숫자 형식	긴 형식	짧은 형식	부모 그룹
1000	TMAX	T	
1100	TMAX_DATA	T_DATA	T
1200	TMAX_SOFT	T_SOFT	T

10.4. 관리자 권한 작동 방식

Label Security 관리자가 설정 한 권한에 따라 사용자의 접근을 제어한다.

10.4.1. 권한 부여 레벨

관리자는 사용자에게 Label Security 정책에 대한 레벨 권한을 명시적으로 설정할 수 있다.

권한 부여	설명
사용자 최대 레벨	사용자가 읽기/쓰기 작업 중 접근 가능한 최대 레벨
사용자 최소 레벨	사용자가 쓰기 작업 중 접근 가능한 최소 레벨(사용자 최대 레벨은 사용자 최소 레벨보다 크거나 같아야 한다.)
사용자 기본 레벨	기본적으로 설정되는 레벨
사용자 기본 행 레벨	데이터를 삽입할 때 기본적으로 사용되는 레벨

10.4.2. 권한 부여 구획

관리자는 사용자가 접근할 수 있는 구획을 명시적으로 지정할 수 있다. 쓰기 권한이 없는 구획이 포함된 행을 직접 삽입, 업데이트 또는 삭제할 수 없다.

다음은 사용자에게 지정한 구획에 대한 예이다.

짧은 이름	긴 이름	쓰기	기본	행
A	알파 (ALPHA)	YES	YES	YES
B	베타(BETA)	YES	YES	NO
C	감마(GAMMA)	YES	YES	NO

10.4.3. 권한 부여 그룹

관리자는 사용자가 접근할 수 있는 그룹을 명시적으로 지정할 수 있다.

다음은 사용자에게 지정한 그룹에 대한 예이다.

짧은 이름	긴 이름	쓰기	기본	행	부모
T	TMAX	YES	YES	YES	
T_DATA	TMAX_DATA	YES	YES	NO	T
T_SOFT	TMAX_SOFT	YES	YES	NO	T

10.5. Label Security 정책 생성

본 절에서는 Label Security의 정책을 생성하는 방법과 그 과정을 설명한다.

10.5.1. Label Security 정책 컨테이너 생성

Label Security 정책 컨테이너는 정책에 대한 메타 데이터를 저장한다. 이 컨테이너는 정책의 이름, 정책을 부여할 테이블에 추가되는 열의 이름, 정책에 대한 기본 옵션을 담고있다.

다음은 정책을 생성하는 구문에 대한 예이다.

```
BEGIN
  SA_SYSDBA.CREATE_POLICY (
    policy_name => 'tls_pol',
    column_name => 'lb_col',
    default_options => 'read_control');
  END;
/
```

Default option은 정책을 특정 테이블에 적용할 때, 테이블 옵션을 부여하지 않을 경우 설정되는 기본 옵션이다.

10.5.2. Label Security 정책에 대한 데이터 레이블 생성

데이터 레이블은 테이블 행 의 민감도를 나타낸다. 데이터 레이블의 구성요소는 레벨, 구획, 그룹이 있다.

구성요소	설명
레벨	등급에 따른 구분 (예: 기밀(SECRET), 대외비(CONFIDENTIAL), 공개(PUBLIC))
구획	영역에 따른 구분 (예: 알파(ALPHA), 베타(BETA), 감마(GAMMA))
그룹	계층에 따른 구분 (예: TMAX, TMAX_DATA, TMAX_SOFT)

정책에 대한 레벨 생성

레벨은 등급에 따른 구분이다. 숫자가 클수록 높은 보안 등급을 의미한다.

숫자 형식	긴 형식	짧은 형식
30	기밀 (SECRET)	S
20	대외비 (CONFIDENTIAL)	C
10	공개 (PUBLIC)	P

다음은 정책에 대한 레벨을 생성하는 예이다.

```
BEGIN
  SA_COMPONENTS.CREATE_LEVEL(
    policy_name => 'tls_pol',
    level_num   => 30,
    short_name => 'S',
    long_name   => 'SECRET'
  );
  SA_COMPONENTS.CREATE_LEVEL(
    policy_name => 'tls_pol',
    level_num   => 20,
    short_name => 'C',
    long_name   => 'CONFIDENTIAL'
  );
  SA_COMPONENTS.CREATE_LEVEL(
    policy_name => 'tls_pol',
    level_num   => 10,
    short_name => 'P',
    long_name   => 'PUBLIC'
  );
```

```
END;
/
```

정책에 대한 구획 생성

구획은 영역에 따른 구분이다. 사용자는 자신이 속한 영역의 데이터를 읽을 수 있다.

숫자 형식	긴 형식	짧은 형식
80	알파 (ALPHA)	A
50	베타 (BETA)	B
20	감마 (GAMMA)	G

다음은 정책에 대한 구획을 생성하는 예이다.

```
BEGIN
  SA_COMPONENTS.CREATE_COMPARTMENT (
    policy_name     => 'tls_pol',
    comp_num      => '80',
    short_name      => 'A',
    long_name       => 'ALPHA'
  );
  SA_COMPONENTS.CREATE_COMPARTMENT (
    policy_name     => 'tls_pol',
    comp_num      => '50',
    short_name      => 'B',
    long_name       => 'BETA')
  ;
  SA_COMPONENTS.CREATE_COMPARTMENT (
    policy_name     => 'tls_pol',
    comp_num       => '20',
    short_name      => 'G',
    long_name       => 'GAMMA')
  ;
END;
/
```

정책에 대한 그룹 생성

그룹은 계층에 따른 구분이다. 사용자는 자신이 속한 조직의 데이터를 읽을 수 있다.

숫자 형식	긴 형식	짧은 형식	부모 그룹
1000	TMAX	T	
1100	TMAX_DATA	T_DATA	T

숫자 형식	긴 형식	짧은 형식	부모 그룹
1200	TMAX_SOFT	T_SOFT	T

다음은 정책에 대한 그룹을 생성하는 예이다.

```
BEGIN
  SA_COMPONENTS.CREATE_GROUP (
    policy_name     => 'tls_pol',
    group_num       => 1000,
    short_name      => 'T’,
    long_name       => 'TMAX')
  ;
  SA_COMPONENTS.CREATE_GROUP (
    policy_name     => 'tls_pol',
    group_num       => 1100,
    short_name      => 'T_DATA',
    long_name       => 'TMAX_DATA’,
    parent_name     => 'T')
  ;
  SA_COMPONENTS.CREATE_GROUP (
    policy_name     => 'tls_pol',
    group_num       => 1200,
    short_name      => 'T_SOFT',
    long_name       => 'TMAX_SOFT',
    parent_name      => 'T')
  ;
END;
/
```

데이터 레이블 생성

SA_LABEL_ADMIN.CREATE_LABEL을 통해 데이터 레이블을 생성할 수 있다.

다음은 데이터 레이블을 생성하는 예이다.

```
BEGIN
  SA_LABEL_ADMIN.CREATE_LABEL(
    policy_name => 'tls_pol',
    label_tag   => 300000,
    label_value => 'S',
    data_label  => TRUE
);
  SA_LABEL_ADMIN.CREATE_LABEL(
    policy_name => 'tls_pol',
    label_tag   => 200000,
    label_value => 'C',
```

```
    data_label  => TRUE
);
  SA_LABEL_ADMIN.CREATE_LABEL(
    policy_name => 'tls_pol',
    label_tag   => 100000,
    label_value => 'P',
    data_label  => TRUE
);
END;
/
```

참고

모든 정책에 대해 레이블 테그(label_tag)는 유일해야 하며, 같은 정책 내에서 같은 레이블 값(label_value)에 서로 다른 레이블 테그를 설정하는 것은 불가능하다. 하지만 다른 정책에 대해서는 레이블 값이 동일 하더라도 서로 다른 레이블 테그를 설정할 수 있다.

10.5.3. Label Security 정책에 대한 사용자 권한 부여

사용자 레이블은 SA_USER_ADMIN.SET_LEVELS, SA_USER_ADMIN.SET_COMPARTMENTS, SA_USER_ADMIN.SET_GROUPS 프러시저를 통해 부여할 수 있다.

정책에 대한 사용자 레벨 부여

레벨은 SA_USER_ADMIN.SET_LEVELS 프러시저를 통해 부여할 수 있다.

다음은 정책에 대한 사용자 레벨을 부여하는 예이다.

```
BEGIN
  SA_USER_ADMIN.SET_LEVELS(
    policy_name => 'tls_pol',
    user_name   => 'tibero',
    max_level   => 'S',
    min_level   => 'C',
    def_level   => 'C',
    row_level   => 'C')
  ;
END;
/
```

매개 변수	설명
policy_name	정책 이름
user_name	정책이 적용될 사용자 이름

매개 변수	설명
max_level	사용자가 읽기/쓰기 작업 중 접근 가능한 최대 레벨
min_level	사용자가 쓰기 잡업 중 접근 가능한 최소 레벨(사용자 최대 레벨은 사용자 최소 레벨보다 크거나 같아야 한다.)
def_level	기본적으로 설정되는 레벨
row_level	데이터를 삽입할 때 기본적으로 사용되는 레벨

정책에 대한 사용자 구획 부여

레벨을 설정한 후 선택적으로 구획을 사용자에게 부여할 수 있다. SA_USER_ADMIN.SET_COMPARTMENTS 프러시저를 통해 부여할 수 있다.

다음은 정책에 대한 사용자 구획을 부여하는 예이다.

```
BEGIN
  SA_USER_ADMIN.SET_COMPARTMENTS (
    policy_name   => 'tls_pol',
    user_name     => 'tibero',
    read_comps    => 'A, B, G',
    write_comps   => 'A, B',
    def_comps     => 'A, B',
    row_comps     => 'A, B')
  ;
END;
/
```

매개 변수	설명
policy_name	정책 이름
user_name	정책이 적용될 사용자 이름
read_comps	읽기 가능한 구획
write_comps	쓰기 가능한 구획
def_comps	기본적으로 설정되는 구획
row_comps	데이터를 삽입할 때 기본적으로 사용되는 구획

정책에 대한 사용자 그룹 부여

레벨을 설정한 후 선택적으로 그룹을 사용자에게 부여할 수 있다. SA_USER_ADMIN.SET_GROUPS 프러시저를 통해 부여할 수 있다.

다음은 정책에 대한 사용자 그룹을 부여하는 예이다.

```
BEGIN
  SA_USER_ADMIN.SET_GROUPS (
    policy_name    => 'tls_pol',
    user_name      => 'tibero',
    read_groups    => 'T',
    write_groups   => 'T',
    def_groups     => 'T',
    row_groups     => 'T')
  ;
END;
/
```

매개 변수	설명
policy_name	정책 이름
user_name	정책이 적용될 사용자 이름
read_groups	읽기 가능한 그룹
write_groups	쓰기 가능한 그룹
def_groups	기본적으로 설정되는 그룹
row_groups	데이터를 삽입할 때 기본적으로 사용되는 그룹

10.5.4. 데이터베이스 테이블에 정책 적용

테이블에 대한 정책 적용은 SA_POLICY_ADMIN.APPLY_TABLE_POLICY 프러시저를 통해 할 수 있다.

다음은 데이터베이스 테이블에 정책을 적용하는 예이다.

```
BEGIN
  SA_POLICY_ADMIN.APPLY_TABLE_POLICY(
    policy_name       => 'tls_pol',
    schema_name       => 'tibero',
    table_name        => 'test_table',
    table_options     =>‘read_control’)
  ;
END;
/
```

매개 변수	설명
policy_name	정책 이름
schema_name	스키마 이름
table_name	정책이 적용될 테이블 이름
table_options	정책 시행 옵션

매개 변수	설명
label_function	Label 함수의 문자열
predicate	Read control과 함께 쓰이는 조건

제11장 백업과 복구

Tibero는 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.

11.1. Tibero 구성 파일

Tibero의 파일은 컨트롤 파일(Control file), 데이터 파일(Data file), 임시 파일(Temp file), 로그 파일(Log file)로 구성된다.

플래시백 데이터베이스(Flashback Database) 기능 사용 시에는 플래시백 로그 파일(Flashback Log File)이 추가된다. 플래시백 데이터베이스 기능에 대해서는 "11.6. 플래시백 데이터베이스"를 참고한다.

각 파일의 특징과 역할은 다음과 같다.

- **컨트롤 파일(Control file)**

  컨트롤 파일은 Tibero를 구성하는 모든 파일의 위치와 데이터베이스의 이름 등 데이터베이스의 구조를 저장하는 파일이다. 특히, Tibero가 사용하는 데이터 파일, 로그 파일 등의 상태 정보가 기록된다. 컨트롤 파일은 Tibero를 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 생성문을 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(Data file)**

  데이터 파일은 사용자의 데이터를 저장하는 파일로써 로그 파일과 함께 데이터베이스를 구성하는 가장 중요한 파일이다.

  Permanent 테이블 스페이스와 Undo 테이블 스페이스에서 정의한 파일로 테이블, 인덱스 등의 데이터베이스 객체를 저장한다. 테이블 스페이스는 한 개 이상의 데이터 파일로 이루어지며, 한 데이터 파일은

하나의 테이블 스페이스에 속한다. 데이터 파일은 사용자의 데이터가 물리적으로 저장되는 공간이므로 반드시 백업해야 한다.

- **임시 파일(Temp file)**

  임시 파일은 데이터베이스가 메모리에서 처리할 수 없는 방대한 양의 데이터를 다루는 경우 임시로 사용하기 위한 공간이다.

  Tibero는 사용자의 질의를 처리하기 위해 정렬 등의 연산을 수행할 때와 임시 테이블(Temporary Table)의 데이터를 저장할 때 데이터 파일을 사용한다. 데이터 파일은 임시 테이블 스페이스(Temporary Tablespace)에서만 정의할 수 있고, 임시 테이블 스페이스는 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(Log file)**

  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.

- **플래시백 로그 파일(Flashback Log File)**

  플래시백 로그 파일은 플래시백 로그를 저장하는 파일이다. 데이터 파일이 변경되기 직전 내용들을 블록 단위로 기록하는 파일로써 플래시백 데이터베이스 기능으로 데이터베이스를 가까운 과거로 되돌릴 때 사용한다. 일반적인 사용 방법과 특징은 로그 파일과 같다.

  ARCHIVELOG 모드로 운영 중이어야만 플래시백 로그를 기록할 수 있다. 플래시백 아카이브 로그 파일을 오래 보관할수록 플래시백 데이터베이스 기능으로 돌아갈 수 있는 과거가 더 오래된다. 하지만, 백업 파일 사용하여 복구 시 플래시백 로그 파일은 모두 무용지물이 되기 때문에 일반 아카이브 로그 파일과 다르게 백업 대상이 아니다. 일반 로그 파일과 다르게 Tibero Standby Cluster의 동기화 대상이 아니다.

11.2. 백업

백업(Backup)은 여러 가지 유형의 장애로부터 데이터베이스를 보호하는 것을 뜻한다. 즉, 시스템 장애가 발생했을 때 복구를 하거나 시스템 작동을 유지하기 위한 절차 또는 기법이다.

Tibero는 데이터베이스를 백업하는 방법을 크게 두 가지로 나누어 수행할 수 있다.

- 논리적 백업

  논리적 백업이란 테이블, 인덱스, 시퀀스와 같은 데이터베이스의 논리적 단위를 백업하는 것을 뜻한다. Tibero에서는 이를 위해 tbExport와 tbImport 유틸리티를 제공하고 있다. tbExport와 tbImport에 대한 자세한 내용은 "Tibero 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 주의한다.

11.2.1. 백업 종류

다음은 Tibero가 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 백업된 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.

모드	설명
	데이터베이스를 구성하는 전체 파일은 반드시 Tibero가 정상적으로 종료된 상태에서 백업한다. 백업 때문에 서비스가 중지되면 안 된다. 복구는 데이터베이스를 백업받은 시점으로부터 복구할 수 있다.

- Consistent 백업

  Tibero를 정상적으로 종료한 상태에서 백업하는 방법이다. 실행 예는 “11.2.2. 백업 실행”의 "Consistent 백업"을 참고한다.

- Inconsistent 백업

  Tibero의 데이터베이스가 운영 중일 때 백업하거나 정상적으로 종료되지 않은 상태에서 백업하는 방법이다. 실행 예는 “11.2.2. 백업 실행”의 "Inconsistent 백업"을 참고한다. **단, NOARCHIVELOG 모드에서는 정합성 문제가 발생할 수 있으므로 이 방법은 허용되지 않는다.**

11.2.2. 백업 실행

본 절에서는 Tibero가 제공하는 물리적 및 논리적인 백업 방법에 따라 백업을 실행하는 예를 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.

컨트롤 파일 백업

컨트롤 파일은 물리적인 백업과 논리적인 백업을 모두 지원하지만, Tibero에서는 컨트롤 파일의 논리적인 백업을 추천한다. 물리적 백업은 제약사항과 사용법이 복잡하기 때문에 실수를 방지하기 위해 보통 사용하지 않는 것이 좋다. 물리적 백업시 데이터의 정합성을 맞춰주기 위해 모든 데이터 파일을 백업한 후 마지막에 컨트롤 파일을 백업해 주어야 한다. 데이터베이스의 구조에 변화가 일어난 경우에는 컨트롤 파일의 생성문을 백업하는 논리적 백업을 사용하는 것을 권장한다.

다음은 컨트롤 파일을 물리적 백업으로 tibero7/backup 디렉터리에 있는 ctrlfile1.ctl 파일에, 논리적 백업으로 컨트롤 파일의 생성문을 ctrlfile1.sql 파일에 백업하는 예이다.

- 컨트롤 파일의 물리적 백업

  [예 11.1] 컨트롤 파일의 물리적 백업

```
SQL> alter database backup controlfile to
       '/tibero7/backup/ctrlfile1.ctl';

Altered.
```

- 컨트롤 파일의 논리적 백업

[예 11.2] 컨트롤 파일의 논리적 백업

```
SQL> alter database backup controlfile to trace as
       '/tibero7/backup/ctrlfile1.sql' reuse NORESETLOGS;

Altered.
```

생성된 ctrlfile1.sql 파일은 다음과 같은 내용을 포함한다.

[예 11.3] 백업된 컨트롤 파일 생성문

```
CREATE CONTROLFILE REUSE DATABASE "inventory"
 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.dtf',
 '/disk1/undo001.dtf'
 NOARCHIVELOG
 MAXLOGFILES 255
 MAXLOGMEMBERS 8
 MAXDATAFILES 100
 MAXBACKUPSET 500
 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 Tibero를 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 11.4] 컨트롤 파일 경로 설정

```
CONTROL_FILES="/home/tibero/tibero7/database/TB7/c1.ctl"
```

다음과 같이 MOUNT나 OPEN 상태에서 컨트롤 파일의 목록을 조회하려면 동적 뷰 V$CONTROLFILE를 조회한다.

[예 11.5] 컨트롤 파일 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 11.5]의 질의 결과와 동일하게 설정한 후 CREATE CONTROLFILE 문을 실행해야 한다.

Consistent 백업

본 절에서는 Tibero가 정상적으로 종료한 후에 백업하는 방법을 설명한다.

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 Tibero의 인스턴스가 시작된 상태이며, OPEN은 컨트롤 파일에 정의한 모든 파일이 오픈된 상태를 의미한다.

[예 11.6] 데이터 파일의 조회

```
SQL> SELECT NAME FROM V$DATAFILE;

NAME
------------------------------------------------------------
/disk1/system001.dtf
/disk2/undo001.dtf
/disk3/user001.dtf

3 selected.
```

다음은 온라인 로그 파일을 MOUNT나 OPEN 상태에서 조회하는 방법이다.

[예 11.7] 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.

데이터베이스는 다음과 같이 NORMAL 모드로 종료해야 한다.

```
SQL> tbdown NORMAL;
Tibero instance was terminated.
```

데이터베이스가 정상적으로 종료되면 운영체제별로 제공하는 파일 복사 명령을 사용하여 백업한다.

Inconsistent 백업

본 절에서는 Tibero가 운영 중일 때 백업하는 방법을 설명한다.

데이터베이스가 운영 중이면 운영체제의 명령어를 사용해 데이터 파일을 복사하는 것은 안전하지 않다. 따라서 다음과 같은 문장을 실행하여 Tibero에 백업의 시작과 종료를 통보해야 한다.

```
alter tablespace {tablespace name} begin backup
...
alter tablespace {tablespace name} end backup
```

begin backup과 end backup 문장 사이에는 해당 테이블 스페이스의 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 테이블 스페이스를 선정한다.

[예 11.8] Inconsistent 백업 - 테이블 스페이스의 선정

```
SQL> select name,type from v$tablespace;

NAME                           TYPE
------------------------------ ----
SYSTEM                         DATA
UNDO                           UNDO
USER                           DATA
TEMP                           TEMP

3 selected.
```

2. 백업할 테이블 스페이스에 속한 데이터 파일을 조회한 후 begin backup, end backup 명령어를 사용하여 백업을 수행한다. 예를 들어 USER 테이블 스페이스를 백업할 경우를 가정하고 수행한다.

[예 11.9] 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.dtf

1 selected
```

```
SQL> alter tablespace SYSTEM begin backup;

Altered.

SQL> !cp /disk3/user001.dtf /backup/
SQL> alter tablespace  SYSTEM end backup;

Altered.
```

11.3. 복구

Tibero를 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. Tibero는 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.

11.3.1. 부트 모드별 복구

Tibero는 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 Tibero를 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

  Tibero의 데이터 파일, 온라인 로그 파일, 컨트롤 파일이 일관성을 유지할 때에만 OPEN 모드로 동작할 수 있다. OPEN 모드에서 Tibero는 세 파일을 열고 정상으로 동작한다. 일반 사용자는 데이터베이스를 이용할 수 있다. 오프라인 상태인 테이블 스페이스에 사용자가 접근하려면 우선 해당 테이블 스페이스를 온라인 상태로 전환해야 한다. 이때 다른 파일들과 일관성을 유지하기 위해 해당 테이블 스페이스에 온라인 미디어 복구를 수행해야 한다.

11.3.2. 파손 복구

파손 복구(Crash Recovery)는 Tibero를 운영하는 중에 정전, 시스템 이상, 강제 종료 등으로 데이터베이스가 비정상적으로 종료되었을 때 사용자의 명령 없이 자동으로 복구되는 것을 의미한다. 복구가 완료되면 Tibero가 정상적으로 동작한다.

파손 복구는 온라인 로그 파일의 내용 중 아직 데이터 파일에 반영되지 않은 부분을 기록하여 Tibero가 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 없으면 DBA의 도움 없이 자동으로 이루어진다.

- 평균 파손 복구 시간 설정

  Tibero는 평균 파손 복구 시간을 설정 할수 있는 기능(Mean Crash Recovery Time)을 제공하고 있다. Mean Crash Recovery Time(이하 MCRT)은 파손 복구시 필요한 I/O 횟수를 통제함으로써 평균 파손 복구 시간을 조절한다.

  MCRT는 다음 파라미터를 설정해서 조절할 수 있다.

파라미터	설명
_MCRT_TARGET	평균 파손 복구 시간을 설정한다. (기본값: 1800, 단위: 초)

11.3.3. 미디어 복구

Tibero를 구성하는 파일이 물리적인 손상이나 정상적으로 동작할 수 없는 경우가 발생할 수 있다. 이러한 경우 데이터베이스가 정상적으로 동작할 수 있도록 복구하는 과정이 **미디어 복구**(Media Recovery)이다.

미디어 복구 과정은 자동으로 이루어지지 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 Tibero를 기동해야 한다. RESETLOGS는 온라인 로그 파일을 초기화하는 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 11.10] 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 또는 특정 시점까지 반영되지 않은 변화를 로그 파일을 사용하여 복구하는 과정으로 나눌 수 있다. 단순한 복원 과정만으로는 Tibero의 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.

11.3.4. 온라인 미디어 복구

Tibero를 운영하는 도중에 일부 데이터 파일이 물리적으로 손상되거나 정상적으로 동작할 수 없는 경우가 발생할 수 있다. 이러한 경우에 OPEN 모드에서 해당 데이터 파일이 포함된 테이블 스페이스만 미디어 복구를 수행할 수 있다. 이것이 **온라인 미디어 복구**(Online Media Recovery)이다. 온라인 미디어 복구는 완전 복구만 가능하다.

11.3.5. 스냅샷 데이터베이스 복구

Tibero에서 공식적으로 제공하는 백업 방법이 아닌 스토리지 스냅샷 솔루션을 이용하여 데이터베이스를 백업한 경우 이를 복구하기 위해서는 아래와 같이 스냅샷 데이터베이스 전용 복구 DDL을 이용하여 복구를 수행해야 한다.

```
SQL> ALTER DATABASE RECOVER AUTOMATIC SNAPSHOT DATABASE;
```

스냅샷 데이터베이스 복구의 경우 복구 관리자를 통한 복구는 현재 지원하지 않는다.

11.3.6. Clone DB 생성 및 복구

테스트 및 여러 목적을 위해 Tibero 백업을 이용하여 Clone DB를 구축할 수 있다. Full Backup을 통해서 운영 DB와 동일하게 구축할 수 있으며, 일부 Tablespace에 대한 Backup을 통해 부분적으로도 구축할 수 있다.

Clone DB 구축 시, 다음과 같이 진행한다.

1. 운영 DB에서 Backup 진행

   Clone DB 구축하기 위한 Backup에는 아래 항목이 포함되어야 한다.

   - Tablespace에 대한 Backup
   - Archive Logfile
   - Control File Trace Backup

2. Clone DB를 생성할 위치에 Backup을 restore
3. Control File 생성문 수정

Clone DB 생성에 필요 없는 Tablespace을 Control File 생성문의 DATAFILE 항목에서 제외한다. 일반적으로 Backup에 온라인 로그파일이 포함되지 않으므로 RESETLOGS 옵션으로 수정한다.

4. NOMOUNT mode로 기동 후, Control File 재생성

```
$ tbboot -t NOMOUNT

SQL> @ctrl_file_backup.sql
```

5. 미디어 복구 진행

   V$ARCHIVE_DEST_FILES를 통해 restore 된 Archive Logfile 들을 조회하여 복구 목표 시점 확인 후, 불완전 복구 진행한다.

```
SQL> alter database recover automatic database until change [target TSN];
```

6. Control File과 Data Dictionary 간 정합성 확인 DDL 수행

```
SQL> alter database check controlfile from data dictionary;
```

7. Clone DB 생성에 필요 없는 Tablespace에 대해 offline immediate DDL 수행

```
SQL> alter tablespace excluded_ts offline immediate;
```

8. DB down 및 RESETLOGS 기동

```
$ tbdown
$ tbboot -t RESETLOGS
```

9. Clone DB 생성에 필요없는 Tablespace에 대해 drop DDL 수행

```
SQL> drop tablespace excluded_ts;
```

11.4. 복구 관리자

Tibero는 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 Tibero는 복구 관리자(이하 RMGR)를 제공한다.

11.4.1. 기본 기능

RMGR은 다양한 백업/복구 시나리오를 지원하도록 구성되어 있다. Tibero에서 제공되는 RMGR의 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 테이블 스페이스를 Hot Backup 상태로 만들고 백업을 진행한다.

백업을 완료하면 데이터베이스의 End Backup 기능을 사용하여 모든 테이블 스페이스를 Hot Backup 상태로부터 해제한다. 명령과 옵션들을 통해 백업이 진행/완료되어 하나의 Backup Set(백업 셋)이 생성되면, V$BACKUP_SET을 통해 진행 상황 및 해당 Backup Set에 사용한 옵션들을 확인할 수 있다.

- Incremental 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 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.

- Block Change Tracking

  Incremental Backup은 이전 백업과 현재 운영 중인 데이터 파일 간의 변경된 사항만을 기록하여야 하기 때문에 전체 데이터 파일을 스캔하여 백업본과의 차이점을 구하는 동작을 수행한다. 이러한 동작은 데이터 파일이 큰 경우에는 데이터 파일을 모두 스캔하는 오버헤드가 크기 때문에 백업되는 파일의 용량은 적지만 시간이 오래 걸리는 단점이 있다. 이런 단점을 보완하기 위해 Tibero 서버가 마지막 백업시점 이후의 데이터 파일의 변화 내역을 기록하고 백업할 때 활용한다. 이 기능은 어떤 블록만 백업하면 될지 추적하기가 쉽기 때문에 Incremental Backup의 수행속도가 비약적으로 빨라질 수 있다. 이를 위해 RMGR 및 Tibero 서버는 Block Change Tracking(BCT)기능을 제공한다.

  BCT 기능을 사용하기 위해서는 Tibero 서버 tip 파일에 BCT 관련 파라미터를 설정하고 BCT 기능을 컨트롤하는 DDL을 수행해야 하며, LGWR AIO 기능과는 호환되지 않는다.

[예 11.11] Tibero 서버 tip 파일에 BCT 설정

```
BLOCK_CHANGE_TRACKING="/database/emp/emp_change.bct"
LGWR_USE_AIO=N
```

[예 11.12] BCT 기능 사용

```
SQL> ALTER SYSTEM ENABLE BLOCK CHANGE TRACKING;

System altered.
```

RMGR은 Incremental Backup을 수행할 때 이 BCT 파일을 사용하여 백업할 블록들의 리스트를 빠른 시간내에 탐색하여 백업을 효과적으로 수행할 수 있다. 해당 파일이 운영 중에 삭제되거나 장애가 생길 때에 Tibero 서버는 자동적으로 BCT 기능을 중지하며 사용자는 다시 BCT 기능을 DDL로 수행해야 한다.

수동으로 BCT 기능을 중지하기 위해서는 다음의 DDL을 수행한다.

[예 11.13] BCT 기능 해제

```
SQL> ALTER SYSTEM DISABLE BLOCK CHANGE TRACKING;

System altered.
```

RMGR은 Tibero 서버에 현재 BCT 기능이 켜져있는지를 질의하여 이전 백업부터의 블록 변화가 BCT 파일에 온전히 기록이 되었는지 확인한 후 유효한 BCT 파일이 존재할 때만 BCT를 이용한 Incremental Backup을 수행한다. Tibero 서버는 Incremental Backup이 끝나면 백업이 끝난 시점부터 변경된 사항들을 BCT 파일에 기록하게 된다.

- Automatic Recovery

  RMGR을 이용하여 만들어진 백업본을 이용하여 자동 복구를 진행한다. 백업 정보는 컨트롤 파일에 저장이 되어있다. 컨트롤 파일에 저장된 정보를 기반으로 Online Full Backup/Incremental Backup 정보를 분석하여 자동으로 Merge 후 복구를 진행한다. 컨트롤 파일이 접근 불가능할때에는 백업된 컨트롤 파일을 이용하여 복구를 진행한다. 단, 이때에는 특정 옵션을 사용하여 백업이 존재하는 백업 경로를 명시해주어야 한다.

주의

TAC 환경에서 RMGR을 이용한 복구를 진행하기 위해서는 구성 노드 중 한 개의 노드만 떠있는 상황에서 복구를 진행해야 한다.

- Datafile/Tablespace 단위 백업 및 복구

  전체 데이터베이스를 백업/복구하는 대신에 필요한 데이터 파일 또는 테이블 스페이스만 대상으로 백업/복구 작업을 수행할 수 있다. 복구할 때에는 데이터 파일 또는 테이블 스페이스는 각 백업 파일들을 가져온 후 복구 자체는 전체 복구로 MOUNT 모드에서 진행된다.

- Delete Backup Set

  RMGR을 이용하여 컨트롤 파일에 등록된 Backup Set(백업 셋)을 삭제할 수 있다. 삭제 대상이 되는 Target Backup Set은 Backup Set ID를 직접 지정할 수 있으며(--bakcup-set 옵션), 특정 시간을 명시하여 특정 시간 이전에 생성된 Backup Set들을 모두 지정할 수 있다(--beforetime 옵션).

  삭제하고자 하는 Backup Set이 존재하는 백업 경로는 컨트롤 파일을 참조하여 결정하므로, 백업이 이루어진 이후에 백업 경로가 변경된 경우에는 삭제하고자 하는 Backup Set이 존재하는 백업 경로(-o 옵션)를 명시해주어야 한다. 만약 컨트롤 파일에 등록된 Backup Set을 사용자가 수동으로 삭제하였거나, 잘못된 백업 경로를 명시하여 백업 경로에서 삭제 대상으로 지정된 Backup Set을 찾을 수 없는 경우에는 컨트롤 파일에 등록된 Backup Set Entry만을 삭제하고 종료할 수 있다(--cf-only 옵션).

- Standby Backup

  RMGR을 이용하여 Tibero Standby Cluster 환경의 standby에서 백업을 진행할 수 있다. Standby의 백업 파일을 이용하여 standby, primary 모두 복구가 가능하다. Standby 환경에서 백업을 진행할 때는 다음과 같은 제약사항이 추가된다.
  – --for-standby 옵션은 사용할 수 없다.

– -w, --with-archivelog 옵션을 사용하려면 primary 접속에 필요한 정보를 tbdsn.tbr에 설정한 후에 primary의 SID를 $TB_SID.tip 파일에 초기화 파라미터 RMGR_PRIMARY_SID로 설정해야 한다.

[예 11.14] Primary SID 설정

```
# tbdsn.tbr
primary=(
        (INSTANCE=(HOST=168.1.1.33)
                  (PORT=8629)
                  (DB_NAME=tibero)
        )
)
# $TB_SID.tip
RMGR_PRIMARY_SID=primary
```

11.4.2. 복구 관리자 옵션

RMGR은 셸 명령으로 실행되며 다양한 옵션을 지정하여 원하는 기능을 사용할 수 있다.

옵션	설명
backup	RMGR를 통해 백업을 진행하여 Backup Set을 생성한다.
recover	RMGR로 백업한 Backup Set을 원하는 경로에 복원하여 복구를 진행한다.
delete	RMGR로 받아놓은 백업본 중 사용자 입력으로 주어진 조건에 맞는 Backup Set을 삭제한다.

--userid	데이터베이스에 접속할 사용자명과 패스워드 및 SID를 아래와 같은 형식으로 지정한다.
	`--userid USERID[[/PASSWD][@SID]]`
	화면상에 비밀번호 노출을 원치 않을 때에는 비밀번호를 공백으로 입력한 후 비밀번호를 입력하라는 문구가 나오면 비공개로 번호를 입력할 수 있다.
	`--userid USERID/[@SID]`
	OS 인증을 받은 계정으로 로그인하는 경우 Userid와 Passwd를 입력하지 않아도 로그인 가능하다(단, 현재는 backup과 delete 기능만 가능).
	`--userid /`

--help	RMGR의 옵션 사용법을 출력한다.
--interval	RMGR 백업/복구 진행률을 확인하여 출력해주는 실시간 시간 간격을 초 단위로 조절한다.

옵션	설명
	기본값은 1 초이며, 최소 0.01 초까지 설정 가능하다. 0으로 설정하는 경우 실시간 진행률을 확인하지 않는다.
-v, --verbose	RMGR 백업/복구를 진행하는 경우 각 데이터 파일마다의 절대 경로를 출력한다.
-s, --silent	RMGR 백업/복구를 진행하는 경우 각 데이터 파일마다의 진행률을 출력하지 않는다.
-l, --log-level	클라이언트 측 RMGR 로그(tbrmgr_trace.log)의 기록 레벨을 설정한다. 레벨은 1부터 5까지 있으며 숫자가 커질 수록 더 자세히 기록된다. 기본 레벨은 4이다.
-L	tbrmgr_trace.log의 기록될 위치를 지정한다. 기본 경로는 $TB_HOME/client/tbrmgr_log/이다.

-o	백업, 복구 및 삭제에 사용될 디렉터리 경로를 지정한다.
	
	백업할 때 옵션을 사용하지 않을 경우 RMGR_BACKUP_DEST가 기본 경로로 설정된다. 복구의 경우 옵션을 사용하지 않으면 각 Backup Set마다 백업된 경로를 자동으로 찾아간다.
	
	옵션을 사용할 경우 모든 Full/Incremental Backup Set 및 아카이브 로그 백업 셋들이 해당 경로에 있어야 한다. 콤마(,)로 구분된 다중경로를 16개까지 지원한다. 백업을 제외한 복구 및 삭제에서 다중경로 옵션을 사용하는 경우, 명시한 디렉터리 경로는 모두 유효해야 한다.
	
	`tbrmgr  backup -o /backup/`
	`tbrmgr  backup -o /backup1/,/backup2/`
	
	아카이브 로그의 경우 다중경로 삭제 기능을 지원하지 않는다.

-n	NetBackup을 사용하는 경우 백업/복구에 사용될 NetBackup 경로를 지정한다. 백업할 때 옵션을 사용하지 않을 경우 RMGR_NBU_BACKUP_DEST가 기본 경로로 설정된다. 이외 특이 사항은 -o 옵션과 동일하며, 복구/삭제가 동작할 때 -o와 -n을 동시에 사용할 수 있다. USE_NBU_FOR_BACKUPSET 파라미터를 Y로 사용할 경우 기본 백업/경로는 로컬이 아닌 NetBackup 경로가 우선적이다.
-i, --incremental	가장 최신 백업에 대한 Incremental backup을 수행한다.
-C, --cumulative	마지막 Full backup에 대한 Incremental backup을 수행한다.
-w, --with-archivelog	백업/복구를 수행할 때 hot backup을 복구하기 위한 아카이브 로그 파일도 함께 백업/복원한다.

옵션	설명
	기본적으로 클러스터 환경에서는 모든 Instance들이 Shared disk로 모두 같은 LOG_ARCHIVE_DEST를 공유해야 정상적으로 동작한다. 이는 Active stroage를 사용하는 경우도 마찬가지이다. NetBackup을 사용하는 경우 USE_NBU_FOR_ARCHIVELOG=Y인 경우엔 옵션 사용 불가하다. 백업된 아카이브 로그 파일은 다음의 형식으로 관리된다. `bkl_<BACKUPSET#>_t<THREAD#>-r<RESETLOGS_TSN>-s<SEQUENCE#>.arc` 예를 들어 BACKUPSET#는 1, THREAD#는 0, RESETLOGS TSN는 0, SEQEUNCE#는 1인 경우 파일명은 'bkl_1_t0-r0-s1.arc'이 된다.

-a, --archive-only	가장 최근에 백업한 아카이브 로그 백업 셋 이후에 생성된 아카이브 로그 파일들을 모두 백업한다. 복구할 때 필요한 경우 자동으로 함께 사용된다.
	
	`tbrmgr backup --archive-only`
	
	특정 Redo 스레드의 아카이브 로그의 특정 시퀀스를 지정하여 해당 시퀀스 이상부터 최신까지의 아카이브 로그 파일들을 백업할 수도 있다(--thread, --from-seq 옵션).
	
	`tbrmgr backup --archive-only --thread <THREAD#> --from-seq <SEQUENCE#>`
	
	NetBackup을 사용하는 경우 USE_NBU_FOR_ARCHIVELOG=Y인 경우엔 옵션 사용 불가하다.

-p, --parallel	복구 전용 Process의 스레드들은 각자 하나의 데이터 파일을 담당하는데, 스레드 개수를 명시하여 여러 스레드를 할당하여 병렬적으로 백업/복구를 수행한다. 기본값은 1이며 최대값은 16이다. 이는 (RECO_PROC_WTHR_CNT - _CACHE_RECO_DOP -3)/2로 조정할 수 있다. `tbrmgr backup --parallel <THREAD_COUNT>`
-u, --skip-unused	아직 사용되지 않은 깨끗한 블록은 백업하지 않는다. 보통 백업에 소용되는 시간은 늘어나고 생성되는 파일 크기를 줄일 수 있다.
-c, --compress	백업을 수행할 때 데이터를 압축하여 저장한다. LOW, MEDIUM, BASIC, HIGH로 옵션을 추가하여 압축 정도를 지정할 수 있다. 옵션을 지정하지 않으면 BASIC 옵션으로 압축이 진행된다. LOW > MEDIUM > BASIC > HIGH 순으로 압축속도가 빠르며, HIGH > BASIC > MEDIUM > LOW 순으로 압축률이 높다. `tbrmgr backup --compress [LOW|MEDIUM|BASIC|HIGH]`

옵션	설명
-d, --datafile	백업/복구할 대상 데이터 파일을 지정한다. 지정할 때에는 데이터 파일 번호를 명시해야 하며, 콤마(,)를 사용하여 여러 개를 명시할 수 있다. `tbrmgr backup --datafile <DATAFILE#1,DATAFILE#2,...>`
-t, --tablespace	백업/복구할 대상 테이블 스페이스를 지정한다. 지정할 때에는테이블 스페이스 이름을 명시해야 하며, 콤마(,)를 사용하여 여러 개를 명시할 수 있다. `tbrmgr backup --tablespace <TABLESPACE_NAME1,TABLESPCE_NAME2,...>`
-T, --skip-tablespace	백업/복구 대상에서 제외할 테이블 스페이스를 지정한다. 지정할 때에는테이블 스페이스 이름을 명시해야 하며, 콤마(,)를 사용하여 여러 개를 명시할 수 있다. `tbrmgr backup --skip-tablespace <TABLESPACE_NAME1,TABLESPACE_NAME2,...>`
--skip-readonly	백업/복구 대상에서 Read only 테이블 스페이스들은 제외한다.
--skip-offline	백업/복구 대상에서 Offline 테이블 스페이스들은 제외한다.
--untiltime	시간기반 불완전 복구를 수행한다. 옵션에서 지정한 시간까지 변경된 내용만 복구된다. 시간은 YYYYMMDDHH24MISS의 형식이다. `tbrmgr recover --untiltime <YYYYMMDDHH24MISS>`
--untilchange	변경 기반 불완전 복구를 수행한다. 옵션에서 지정한 TSN까지 변경된 내용만 복구된다. `tbrmgr recover --untilchange <TSN>`
--from-seq	해당 옵션 사용하여 아카이브 로그의 특정 시퀀스를 지정할 수 있고, 해당 시퀀스 부터 마지막 시퀀스, 또는 특정 시퀀스까지 백업/복구한다(--to-seq 옵션). 무조건 Redo 스레드를 지정해주어야 한다(--thread 옵션). `tbrmgr backup --thread <THREAD#> --from-seq <SEQUENCE#>`
--to-seq	해당 옵션 사용하여 아카이브 로그의 특정 시퀀스를 지정할 수 있고, 해당 시퀀스까지 모두 복구한다. 무조건 Redo 스레드를 지정해주고(--thread 옵션), 구간을 정해주어야 한다(--from-seq 옵션). `tbrmgr recover --thread <THREAD#> --from-seq <SEQUENCE#> --to-seq <SEQUENCE#>`

옵션	설명
--thread	아카이브 로그 백업/복구할 때 Redo 스레드를 지정한다.
--arc-dest-force	아카이브 로그 백업/복구할 때 대상 구간에 임의의 아카이브 로그 파일을 찾을 수 없어도 실패하지 않고 진행되게 한다.
--delete-original	아카이브 로그 백업/복구 후 백업 본이 아닌 원본 파일들을 삭제한다.
--with-password-file	MOUNT 모드에서 SYS 계정 로그인에 필요한 password file을 함께 백업/복구한다.
--no-rollback	백업 도중 취소/실패하는 경우 기존에 백업한 파일들을 롤백하지 않고 보존한다.
--continue	백업 파일을 가져오지 않고 복구만 진행한다. 주로 아카이브 로그 파일들을 소량씩 가져오며 불완전 복구를 단계적으로 하려고할 때 사용할 수 있다. MOUNT 모드에서만 사용 가능하다.
--for-standby	Standby 구축을 위한 백업/복구를 진행한다. 모든 데이터 파일들과 복구에 필요한 아카이브 로그 파일, 그리고 온라인 Redo 로그 파일들까지 백업/복구한다. 클러스터 환경에 대한 제약 사항이 --with-archivelog와 동일하다. NetBackup 연동을 지원하지 않는다.
--at-standby	Standby에서의 복구를 진행한다. --for-standby와 다르게 standby에서 별개의 Media recovery를 진행해야할 때 사용된다.
--recover-to	Backup Set을 지정한 특정 경로에 가져온 후 복구를 수행한다. 데이터베이스와 컨트롤 파일이 알고 있는 파일들의 경로도 함께 변경된다. Active storage 경로도 가능하다. `tbrmgr recover --recover-to <NEW_DIRECTORY>`
--restore-only	대상 백업 데이터 파일들을 가져온 후 복구는 수행하지 않는다. MOUNT 또는 NORMAL 모드에서 사용 가능하며, NORMAL 모드에서는 무조건 Offline 테이블 스페이스들을 지정해야 한다. `tbrmgr recover --restore-only`
--restore-archive-only	대상 백업 아카이브 로그 파일들을 가져온 후 복구는 수행하지 않는다. 무조건 --from-seq, --to-seq, --thread 옵션을 함께 사용해야 한다. MOUNT 또는 NORMAL 모드에서 사용 가능하다. NetBackup을 사용하는 경우 USE_NBU_FOR_ARCHIVELOG=Y인 경우엔 옵션 사용이 불가하다.

옵션	설명
--restore-incremental-only	대상 incremental backup 파일들을 가져와서 대응되는 데이터파일과 병합한다. 단, incremental backup이 백업될 당시 기준으로 한 full backup 데이터파일 상태여야한다. 이 기능은 주로 테스트 장비에서 full backup set으로 resetlogs 부팅하여 테스트 후, flashback database로 full backup set 상태로 다시 돌아오고 나서 사용된다. 즉, 해당 상태에서 운영기의 incremental backup을 병합하여 테스트 장비가 운영기를 간헐적 동기화 될 수 있게 한다. MOUNT 또는 NORMAL 모드에서 사용 가능하다.
--incrementally-updated	존재하는 full backup set을 해당 backup set을 base로 하는 incremental/cumulative backup set로 incremental update한다. 즉, 존재하는 full backup set과 incremental/cumulative backup set을 병합하여 하나의 full backup set으로 만드는 기능이다. 이를 통해 저장 공간 확보 및 추후 복구 시간 축소의 장점을 갖는다.
--wallet	암호화된 테이블 스페이스에 접근하기 위한 인증작업을 수행한다. 사용자가 명시한 PASSWORD를 통해 WALLET을 열고 복구를 시작한다. `tbrmgr recover --wallet <WALLET_PASSWORD>`
-b, --backup-set	지정한 Backup Set을 삭제하거나 복구 시에는 지정한 Backup Set부터 탐색하여 가져온다.
--archivelog	컨트롤 파일에 기록된 아카이브 로그들을 삭제하고, 사용자가 경로를 지정한 경우에는 해당 경로에 있는 아카이브 로그들도 삭제한다. --beforetime, --beforechange 혹은 --sent-to-standby 옵션과 같이 사용되어야 한다.
--beforetime	명시한 시점 이전의 Backup Set들 혹은 아카이브 로그들을 삭제한다. 시간은 YYYYMMDDHH24MISS의 형식이다. `tbrmgr delete --beforetime <YYYYMMDDHH24MISS>`
--beforechange	명시한 TSN 이전의 아카이브 로그들을 삭제한다.
--redundancy	명시한 숫자만큼의 full backup set 개수보다 오래된 backup set들을 삭제한다.
--recovery-window	명시한 숫자만큼의 일 수보다 오래된 backup set들을 삭제한다.
--cf-only	실제 Backup Set 혹은 아카이브 로그의 물리적 파일은 삭제하지 않고 컨트롤 파일에서만 정보를 삭제한다. 백업 파일을 사용자가 수동으로 삭제하였거나, 삭제 대상 Backup Set을 물리적으로 찾지 못할 때 사용 가능하다.
--sent-to-standby	Standby로 전송이 완료된 아카이브 로그들을 삭제한다. --archivelog 옵션과 같이 사용되어야 하며, --beforetime 또는 --beforechange와 같이 사용될 수 있다.

옵션	설명
--switch	데이터베이스의 파일이름을 Backup Set의 백업 파일의 이름으로 바꿔준다. 테이블 스페이스를 지정할 수 있다. NetBackup을 사용한 Backup Set에 대해서는 지원하지 않는다. `tbrmgr recover --switch`
--no-image-logging	백업을 수행할 때 기존의 image logging 방식이 아닌 block consistency check 방식을 이용한다. standby에서 백업을 수행할 경우 옵션 사용 여부 상관없이 block consistency check 방식으로 진행한다. 백업에 소요되는 시간은 늘어나지만 백업하는 동안 발생하는 온라인 redo 로그의 크기는 줄어든다.

11.4.3. 복구 관리자를 이용한 백업 및 복구 예제

본 절에서는 다음의 백업/복구 시나리오를 통해서 RMGR을 이용한 백업/복구를 설명한다.

- Online Full Backup 시나리오
- Compress 옵션과 Skip Unused 옵션을 적용한 Online Full Backup 시나리오
- With Archive Log 옵션을 적용한 Online Full Backup 시나리오
- With Archive Log 옵션을 적용한 Incremental Backup 시나리오
- Online Full Backup을 이용한 복구 시나리오
- Online Full Backup과 Archive Log Backup을 이용한 복구 시나리오
- Online Full Backup과 Incremental Backup을 이용한 복구 시나리오
- Tablespace 기반 복구 시나리오

주의

RMGR의 Online(Full / Incremental) Backup을 통해 생성된 Backup Set을 이용하여 복구를 수행하기 위해서는 Archive Log File이 반드시 필요하다. 따라서 Archive Log File이 유실될 경우에 대비하여 Online(Full / Incremental) Backup의 경우 Archive Log File도 함께 Backup (--with-archivelog / --archive-only 옵션) 해두는 것이 바람직하다. 또한, TAC 환경에서는 RMGR을 이용한 Archive Log Backup을 위해서는 모든 노드들의 Archive Log File에 접근할 수 있도록, 모든 노드들은 공유 디스크의 동일한 LOG_ARCHIVE_DEST에 Archive Log File을 저장해야한다.

RMGR은 대부분의 과정을 자동으로 진행하기 때문에 사용자가 실행 명령을 통해 작업을 명시해준 이후에는 특별히 관리해야 할 사항이 없다. RMGR이 작업을 진행하는 동안에는 작업의 진행 과정을 살펴볼 수 있다.

RMGR을 통해 Backup을 진행한 이후에는 V$BACKUP_SET을 통해 Backup Set 정보를 조회할 수 있으며, V$BACKUP_ARCHIVED_LOG를 통해 Archive Log Backup 정보를 조회할 수 있다. V$BACKUP_SET_TABLESPACE를 조회하면 각 Backup Set에 대한 테이블 스페이스 정보를 확인할 수 있다.

참고

데이터 파일이 Raw Device 파일 및 Tibero Active Storage인 경우에도 RMGR을 이용한 백업 및 복구가 가능하다. 단, Active Storage를 사용할 때 미리 Active Storage Instance를 설정 및 부팅을 해야 한다.

Online Full Backup 시나리오

RMGR을 이용하여 임의의 백업 경로에서 Online Full Backup을 수행할 수 있으며(-o 옵션), 백업 경로를 명시하지 않은 경우에는 RMGR_BACKUP_DEST가 기본 dest로 설정된다.

[예 11.15] Online Full Backup 시나리오

```
$ tbrmgr backup -o /home/tbrdb/work/7/backup/
================================================================
= Recovery Manager(RMGR) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
==========================================
  RMGR - ONLINE backup
==========================================
DB connected
archive log check succeeded
 100.00% |=====================================>|  12800/12800  blks 0.08s
Synchronizing...
 100.00% |=====================================>|  25600/25600  blks 0.18s
Synchronizing...
 100.00% |=====================================>|  12800/12800  blks 0.10s
Synchronizing...
 100.00% |=====================================>|   1280/1280   blks 0.02s
Synchronizing...
Database full backup succeeded
DB disconnected
RMGR backup ends

$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;
```

```
    SET_ID START_TIME
---------- ------------------------------------------------------------
FINISH_TIME                                                       START_TSN
------------------------------------------------------------ ----------
FINISH_TSN RESETLOGS_TSN   BASE_SET   SIZE(KB) IS_PARTIAL IS_INCREMENTAL
---------- ------------ ---------- ---------- ---------- --------------
WITH_ARCHIVELOG
---------------
         1 2018/06/11
2018/06/11                                                         36321
     36338             0          0     453588 NO         NO
NO

1 row selected.

SQL> select * from V$BACKUP_ARCHIVED_LOG;

0 row selected.
```

Compress 옵션과 Skip Unused 옵션을 적용한 Online Full Backup 시나리오

데이터를 압축하여 Backup Set을 생성하는 Compress(-c) 옵션과 실제로 사용되지 않은 블록을 백업 대상에서 제외하는 Skip Unused(-u) 옵션을 함께 적용하여 Online Full Backup을 수행할 수 있다.

[예 11.16] Compress 옵션과 Skip Unused 옵션을 적용한 Online Full Backup 시나리오

```
$ tbrmgr backup -c -u -o /home/tbrdb/work/7/backup/
================================================================
= Recovery Manager(RMGR) starts                               =
=                                                             =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
==========================================
  RMGR - ONLINE backup
==========================================
DB connected
archive log check succeeded
 100.00% |=======================================>|  12800/12800  blks 1.00s
Synchronizing...
 100.00% |=======================================>|  25600/25600  blks 1.85s
Synchronizing...
 100.00% |=======================================>|  12800/12800  blks 0.00s
Synchronizing...
 100.00% |=======================================>|   1280/1280   blks 0.06s
Synchronizing...
```

```
Database full backup succeeded
DB disconnected
RMGR backup ends
```

With Archive Log 옵션을 적용한 Online Full Backup 시나리오

RMGR을 이용한 Backup을 수행하는 경우 with Archive Log(--with-archivelog) 옵션을 적용함으로써, 백업 경로에 데이터 파일에 대한 백업과 함께 Archive Log Backup을 생성할 수 있다. Online Backup을 이용하여 복구를 수행하기 위해서는 Archive Log가 반드시 필요하므로, Archive Log Backup을 생성함으로써 원본 Archive Log File이 유실되는 경우에도 정상적으로 복구를 진행할 수 있다.

백업된 아카이브 로그 파일은 다음의 형식으로 관리된다.

```
bkl_<BACKUPSET#>_t<THREAD#>-r<RESETLOGSTSN>-s<SEQUENCE#>.arc
```

예를 들어 BACKUPSET#는 1, THREAD#는 0, RESETLOGS TSN는 0, SEQEUNCE#는 1인 경우 파일명은 'bkl_1_t0-r0-s1.arc'이 된다.

V$BACKUP_SET을 조회함으로써 각 Backup Set에 Archive Log Backup이 존재하는지 확인할 수 있으며, V$BACKUP_ARCHIVED_LOG를 조회함으로써 Archive Log Backup의 정보를 확인할 수 있다.

[예 11.17] With Archive Log 옵션을 적용한 Online Full Backup 시나리오

```
$ tbrmgr backup --with-archivelog -o /home/tbrdb/work/7/backup/
================================================================
= Recovery Manager(RMGR) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
============================================
  RMGR - ONLINE backup
============================================
DB connected
archive log check succeeded
 100.00% |=======================================>|  12800/12800  blks 0.08s
Synchronizing...
 100.00% |=======================================>|  25600/25600  blks 0.18s
Synchronizing...
 100.00% |=======================================>|  12800/12800  blks 0.08s
Synchronizing...
 100.00% |=======================================>|   1280/1280   blks 0.02s
Synchronizing...
Database full backup succeeded
DB disconnected
RMGR backup ends
```

```
$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;

    SET_ID START_TIME
---------- --------------------------------------------------------------------
FINISH_TIME                                                          START_TSN
---------- --------------------------------------------------------------------
FINISH_TSN RESETLOGS_TSN   BASE_SET   SIZE(KB) IS_PARTIAL IS_INCREMENTAL
---------- ------------- ---------- ---------- ---------- --------------
WITH_ARCHIVELOG
---------------
         1 2016/06/16
2016/06/16                                                             34386
     34441             0          0     453588 NO         NO
YES


1 row selected.

SQL> set line 200
SQL> col MIN_LOG_TIME for a20
SQL> col MAX_LOG_TIME for a20
SQL> col RESETLOG_TIME for a20
SQL> select * from V$BACKUP_ARCHIVED_LOG a;

    SET_ID MIN_LOG_TSN MAX_LOG_TSN MIN_LOG_TIME         MAX_LOG_TIME
---------- ----------- ----------- -------------------- --------------------
         1       34386       34441 2016/06/16           2016/06/16

MIN_LOG_SEQUENCE MAX_LOG_SEQUENCE RESETLOG_TSN RESETLOG_TIME
---------------- ---------------- ------------ --------------------
               2                2            0

1 row selected.
```

With Archive Log 옵션을 적용한 Incremental Backup 시나리오

앞서 Online Full Backup을 통해 생성된 Full Backup Set이 최소 1개 이상 존재하는 경우에는 가장 최신의 Backup Set과 현재 상태를 비교하여 변경사항만을 백업하는 Incremental Backup을 수행할 수 있다. 변경사항만을 백업하므로 Full Backup Set에 비해 Backup Set의 Size를 획기적으로 줄일 수 있지만, 앞선

Backup Set이 유실되어 비교 대상이 사라지게 되면 Backup Set을 데이터베이스 복구에 사용할 수 없는 위험성이 존재한다.

V$BACKUP_SET을 조회함으로써 각 Backup Set이 Incremental Backup Set인지 확인할 수 있으며, Incremental Backup의 경우 비교 대상이 되는 Backup Set(Base Set)의 ID를 확인할 수 있다. Full Backup Set의 경우 Base Set이 존재하지 않으므로 Base Set ID가 0으로 표기된다.

[예 11.18] With Archive Log 옵션을 적용한 Incremental Backup 시나리오

```
$ tbrmgr backup -i --with-archivelog -o /home/tbrdb/work/7/backup/
================================================================
= Recovery Manager(RMGR) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
==========================================
  RMGR - INCREMENTAL backup
==========================================
DB connected
archive log check succeeded
 100.00% |=======================================>|  12800/12800  blks 0.04s
Synchronizing...
 100.00% |=======================================>|  25600/25600  blks 0.04s
Synchronizing...
 100.00% |=======================================>|  12800/12800  blks 0.02s
Synchronizing...
 100.00% |=======================================>|   1280/1280   blks 0.02s
Synchronizing...
Database incremental backup succeeded
DB disconnected
RMGR backup ends

$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;

    SET_ID START_TIME
---------- --------------------------------------------------------------
FINISH_TIME                                                    START_TSN
-------------------------------------------------------------- ----------
FINISH_TSN RESETLOGS_TSN   BASE_SET   SIZE(KB) IS_PARTIAL IS_INCREMENTAL
---------- ------------- ---------- ---------- ---------- --------------
WITH_ARCHIVELOG
---------------
         1 2016/06/16
```

```
2018/06/11                                                          34386
     34441             0          0     453588 NO         NO
YES

         2 2016/06/16
2018/06/11                                                          34448
     35234             0          1      23730 NO         YES
YES


2 rows selected.

SQL> set line 200
SQL> col MIN_LOG_TIME for a20
SQL> col MAX_LOG_TIME for a20
SQL> col RESETLOG_TIME for a20
SQL> select * from V$BACKUP_ARCHIVED_LOG a;

    SET_ID MIN_LOG_TSN MAX_LOG_TSN MIN_LOG_TIME         MAX_LOG_TIME
---------- ----------- ----------- -------------------- --------------------
         1       34386       34441 2016/06/16           2016/06/16
         2       34448       35234 2016/06/16           2016/06/16


MIN_LOG_SEQUENCE MAX_LOG_SEQUENCE RESETLOG_TSN RESETLOG_TIME
---------------- ---------------- ------------ --------------------
               2                2            0
               6                6            0

2 row selected.
```

Online Full Backup을 이용한 복구 시나리오

RMGR은 Online Full Backup을 통해 생성된 Backup Set을 이용하여 복구를 수행할 수 있다. 아래 예제는 Backup Set에 Archive Log Backup이 존재하지 않는 경우로, 이 경우에는 원본 Archive Log File을 가지고 있어야 복구를 진행할 수 있다.

[예 11.19] Online Full Backup을 이용한 복구 시나리오

```
$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;
```

```
    SET_ID START_TIME
---------- ------------------------------------------------------------
FINISH_TIME                                                    START_TSN
---------- ------------------------------------------------------------ ----------
FINISH_TSN RESETLOGS_TSN   BASE_SET   SIZE(KB) IS_PARTIAL IS_INCREMENTAL
---------- ------------ ---------- ---------- ---------- --------------
WITH_ARCHIVELOG
---------------
         1 2016/06/16
2016/06/16                                                       34386
     34441             0          0     453588 NO         NO
NO

1 row selected.

SQL> quit
Disconnected.

$ tbrmgr recover -o /home/tbrdb/work/7/backup/
================================================================
= Recovery Manager(RMGR) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
============================================
  RMGR - recovery
============================================
Tibero instance terminated (ABNORMAL mode).

Control file #0 (/home/tbrdb/work/7/database/TB7/c1.ctl) is accessible
Listener port = 45648

Tibero 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.
Tibero instance started up (MOUNT mode).
DB Connected

RMGR BEGIN RESTORE
 full backup set_id: 1
 last incremental backup set_id: 1

Applying FULL BACKUP (set_id:1, ts_id:0, df_id:0)
 100.00% |=======================================>|  12800/12800  blks 0.00s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:1, df_id:1)
 100.00% |=======================================>|  25600/25600  blks 0.20s
```

```
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:3, df_id:2)
 100.00% |=======================================>|  12800/12800  blks 0.00s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:4, df_id:3)
 100.00% |=======================================>|   1280/1280   blks 0.00s
Synchronizing...
Database restore succeeded
recoverSQL: ALTER DATABASE RECOVER AUTOMATIC
Database automatic recovery succeeded
DB disconnected

Tibero instance terminated (NORMAL mode).

Listener port = 45648

Tibero 7

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

Online Full Backup과 Archive Log Backup을 이용한 복구 시나리오

RMGR은 Online Full Backup을 통해 생성된 Backup Set을 이용하여 데이터베이스 복구를 수행할 수 있다. 아래 예제는 원본 Archive Log File이 유실된 경우로, with Archive Log(--with-archivelog) 옵션을 통해 Archive Log Backup을 Restore하는 경우에는 정상적으로 복구가 진행되지만, with Archive Log(--with-archivelog) 옵션을 적용하지 않고 복구를 수행하는 경우에는 복구에 실패하는 것을 확인할 수 있다.

[예 11.20] Online Full Backup과 Archive Log Backup을 이용한 복구 시나리오

```
$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;

    SET_ID START_TIME
---------- --------------------------------------------------------------------
FINISH_TIME                                                          START_TSN
-------------------------------------------------------------------- ----------
FINISH_TSN RESETLOGS_TSN   BASE_SET   SIZE(KB) IS_PARTIAL IS_INCREMENTAL
---------- ------------- ---------- ---------- ---------- --------------
WITH_ARCHIVELOG
---------------
```

```
         1 2016/06/16
2016/06/16                                                        34386
     34441             0          0     453588 NO         NO
YES

1 row selected.

SQL> set line 200
SQL> col MIN_LOG_TIME for a20
SQL> col MAX_LOG_TIME for a20
SQL> col RESETLOG_TIME for a20
SQL> select * from V$BACKUP_ARCHIVED_LOG a;

    SET_ID MIN_LOG_TSN MAX_LOG_TSN MIN_LOG_TIME         MAX_LOG_TIME
---------- ----------- ----------- -------------------- --------------------
         1       34386       34441 2016/06/15           2016/06/16

MIN_LOG_SEQUENCE MAX_LOG_SEQUENCE RESETLOG_TSN RESETLOG_TIME
---------------- ---------------- ------------ --------------------
               2                2            0

1 row selected.

SQL> quit
Disconnected.

$ tbrmgr recover -o /home/tbrdb/work/7/backup/
================================================================
= Recovery Manager(RMGR) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
============================================
  RMGR - recovery
============================================
Tibero instance terminated (ABNORMAL mode).

Control file #0 (/home/tbrdb/work/7/database/TB7/c1.ctl) is accessible
Listener port = 45648

Tibero 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.
Tibero instance started up (MOUNT mode).
DB Connected

RMGR BEGIN RESTORE
```

```
 full backup set_id: 1
 last incremental backup set_id: 1

Applying FULL BACKUP (set_id:1, ts_id:0, df_id:0)
 100.00% |=======================================>|  12800/12800  blks 0.00s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:1, df_id:1)
 100.00% |=======================================>|  25600/25600  blks 0.20s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:3, df_id:2)
 100.00% |=======================================>|  12800/12800  blks 0.00s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:4, df_id:3)
 100.00% |=======================================>|   1280/1280   blks 0.00s
Synchronizing...
Database restore succeeded
recoverSQL: ALTER DATABASE RECOVER AUTOMATIC
RMGR Error: recovery failed (automatic recovery failed)
SVR Error:  Unable to find archive log file for thread 0 from change 34428.


$ tbrmgr recover --with-archivelog -o /home/tbrdb/work/7/backup/
================================================================
= Recovery Manager(RMGR) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
==========================================
  RMGR - recovery
==========================================
Tibero instance terminated (ABNORMAL mode).

Control file #0 (/home/tbrdb/work/7/database/TB7/c1.ctl) is accessible
Listener port = 45648

Tibero 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.
Tibero instance started up (MOUNT mode).
DB Connected

RMGR BEGIN RESTORE
 full backup set_id: 1
 last incremental backup set_id: 1

Applying FULL BACKUP (set_id:1, ts_id:0, df_id:0)
 100.00% |=======================================>|  12800/12800  blks 0.00s
```

```
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:1, df_id:1)
 100.00% |=======================================>|  25600/25600  blks 0.20s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:3, df_id:2)
 100.00% |=======================================>|  12800/12800  blks 0.00s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:4, df_id:3)
 100.00% |=======================================>|   1280/1280   blks 0.00s
Synchronizing...
Database restore succeeded
recoverSQL: ALTER DATABASE RECOVER AUTOMATIC
Database automatic recovery succeeded
DB disconnected

Tibero instance terminated (NORMAL mode).

Listener port = 45648

Tibero 7

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

Online Full Backup과 Incremental Backup을 이용한 복구 시나리오

RMGR은 Online Full Backup을 통해 생성된 Backup Set과 Incremental Backup을 통해 생성된 Backup Set을 자동으로 Merge하여 복구를 진행한다.

[예 11.21] Online Full Backup과 Incremental Backup을 이용한 복구 시나리오

```
$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;

    SET_ID START_TIME
---------- --------------------------------------------------------------------
FINISH_TIME                                                          START_TSN
---------------------------------------------------------------- ----------
FINISH_TSN RESETLOGS_TSN   BASE_SET   SIZE(KB) IS_PARTIAL IS_INCREMENTAL
---------- ------------- ---------- ---------- ---------- --------------
WITH_ARCHIVELOG
```

```
---------------
         1 2016/06/16
2018/06/11                                                          34386
     34441             0          0     453588 NO         NO
YES

         2 2016/06/16
2018/06/11                                                          34448
     35234             0          1      23730 NO         YES
YES


2 rows selected.

SQL> set line 200
SQL> col MIN_LOG_TIME for a20
SQL> col MAX_LOG_TIME for a20
SQL> col RESETLOG_TIME for a20
SQL> select * from V$BACKUP_ARCHIVED_LOG a;

    SET_ID MIN_LOG_TSN MAX_LOG_TSN MIN_LOG_TIME         MAX_LOG_TIME
---------- ----------- ----------- -------------------- --------------------
         1       34386       34441 2016/06/16           2016/06/16
         2       34448       35234 2016/06/16           2016/06/16

MIN_LOG_SEQUENCE MAX_LOG_SEQUENCE RESETLOG_TSN RESETLOG_TIME
---------------- ---------------- ------------ --------------------
               2                2            0
               6                6            0

2 row selected.

SQL> quit
Disconnected.

$ tbrmgr recover --with-archivelog -o /home/tbrdb/work/7/backup
=================================================================
= Recovery Manager(RMGR) starts                                 =
=                                                               =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
=================================================================
===========================================
  RMGR - recovery
===========================================
Tibero instance terminated (ABNORMAL mode).

Control file #0 (/home/tbrdb/work/7/database/TB7/c1.ctl  ) is accessible
```

```
Listener port = 45648

Tibero 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.
Tibero instance started up (MOUNT mode).
DB Connected

RMGR BEGIN RESTORE
 full backup set_id: 1
 last incremental backup set_id: 2

Applying FULL BACKUP (set_id:1, ts_id:0, df_id:0)
 100.00% |=======================================>|  12800/12800  blks 0.00s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:1, df_id:1)
 100.00% |=======================================>|  25600/25600  blks 0.20s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:3, df_id:2)
 100.00% |=======================================>|  12800/12800  blks 0.00s
Synchronizing...
Applying FULL BACKUP (set_id:1, ts_id:4, df_id:3)
 100.00% |=======================================>|   1280/1280   blks 0.00s
Synchronizing...
Applying INCREMENTAL BACKUP (set_id:2, ts_id:0, df_id:0)
 100.00% |=======================================>|  12800/12800  blks 0.60s
Synchronizing...
Applying INCREMENTAL BACKUP (set_id:2, ts_id:1, df_id:1)
 100.00% |=======================================>|  25600/25600  blks 1.20s
Synchronizing...
Applying INCREMENTAL BACKUP (set_id:2, ts_id:3, df_id:2)
 100.00% |=======================================>|  12800/12800  blks 0.80s
Synchronizing...
Applying INCREMENTAL BACKUP (set_id:2, ts_id:4, df_id:3)
 100.00% |=======================================>|   1280/1280   blks 0.00s
Synchronizing...
Database restore succeeded
recoverSQL: ALTER DATABASE RECOVER AUTOMATIC
Database automatic recovery succeeded
DB disconnected

Tibero instance terminated (NORMAL mode).

Listener port = 45648

Tibero 7
```

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

Tablespace 기반 복구 시나리오

RMGR은 사용자가 Tablespace Name으로 지정한(--tablespace 옵션) 특정 테이블 스페이스에 대해서만 복구를 수행할 수 있다. V$BACKUP_SET_TABLESPACE을 조회함으로써 각 Backup Set에 포함된 테이블 스페이스를 확인할 수 있다.

[예 11.22] Tablespace 기반 복구 시나리오

```
$ tbsql sys/tibero

SQL> set line 200
SQL> col NAME for a20
SQL> select * from V$TABLESPACE a;

       TS# NAME                 TYPE BIGFILE FLASHBACK_ON
---------- -------------------- ---- ------- ------------
         0 SYSTEM               DATA NO      NO
         1 UNDO                 UNDO NO      NO
         2 TEMP                 TEMP NO      NO
         3 USR                  DATA NO      NO
         4 SYSSUB               DATA NO      NO

5 rows selected.

SQL> select * from V$BACKUP_SET_TABLESPACE;

    SET_ID        TS#
---------- ----------
         1          0
         1          1
         1          3
         1          4

4 rows selected.

SQL> quit
Disconnected.

$ tbrmgr recover --tablespace USR -o /home/tbrdb/work/7/backup/
================================================================
= Recovery Manager(RMGR) starts                                =
=                                                              =
```

```
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
=====================================================================
==========================================
RMGR - recovery
==========================================
Tibero instance terminated (ABNORMAL mode).

Control file #0 (/home/tbrdb/work/7/database/TB7/c1.ctl) is accessible
Listener port = 45648

Tibero 7

TmaxData Corporation Copyright (c) 2008-. All rights reserved.
Tibero instance started up (MOUNT mode).
DB Connected

RMGR BEGIN RESTORE
 full backup set_id: 1
 last incremental backup set_id: 1

Applying FULL BACKUP (set_id:1, ts_id:3, df_id:2)
 100.00% |=======================================>|  12800/12800  blks 0.00s
Synchronizing...
Database restore succeeded
recoverSQL: ALTER DATABASE RECOVER AUTOMATIC
Database automatic recovery succeeded
DB disconnected

Tibero instance terminated (NORMAL mode).

Listener port = 45648

Tibero 7

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

11.4.4. 복구 관리자를 이용한 백업 삭제 예제

본 절에서는 다음의 백업 삭제 시나리오를 통해서 RMGR을 이용한 백업 삭제를 설명한다.

- Backup Set ID에 기반한 백업 삭제 시나리오
- Backup Date에 기반한 백업 삭제 시나리오

RMGR을 이용하여 Backup Set을 삭제하는 경우, 삭제 대상이 되는 Target Backup Set은 두 가지 방식으로 지정할 수 있다. 첫 번째로 Backup Set ID(--backup_set 옵션)를 명시하여 해당 Backup Set만을 삭제할 수 있으며, 두 번째로 Backup Date(--beforetime 옵션)를 명시하여 명시한 Backup Date 이전에 백업이 이루어진 모든 Backup Set들을 삭제할 수 있다.

RMGR은 컨트롤 파일을 참조하여 삭제 대상으로 지정된 Backup Set의 존재 유무와 백업 경로를 확인한 후 삭제를 진행한다. 따라서 컨트롤 파일에서 참조할 수 없는 Backup Set은 RMGR을 이용하여 삭제할 수 없으며, Backup Set이 존재하는 백업 경로가 백업할 때 컨트롤 파일에 등록된 백업 경로와 다른 경우에는 Backup Dest(-o 옵션)를 직접 명시해주어야 변경된 백업 경로에서 실제 백업된 파일에 대한 삭제를 진행할 수 있다.

컨트롤 파일에 등록되어 있는 Backup Set 정보는 V$BACKUP_SET을 통해 조회할 수 있으며, 각 Backup Set에 속하는 Archive Log 정보는 V$BACKUP_ARCHIVED_LOG를 통해 조회할 수 있다.

참고

컨트롤 파일에 등록된 Backup Set을 사용자가 수동으로 삭제하였거나, 잘못된 백업 경로를 명시하여 백업 경로에서 삭제 대상으로 지정된 Backup Set을 찾을 수 없는 경우에는 컨트롤 파일에 등록된 Backup Set Entry만을 삭제하고 종료한다. 또한 테이프 장비에 저장된 경우 역시 컨트롤 파일에 등록된 Backup Set Entry만을 삭제하고 종료한다.

Backup Set ID에 기반한 백업 삭제 시나리오

RMGR은 사용자가 Backup Set ID로 지정한(--backup_set 옵션) 특정 Backup Set만을 선택적으로 삭제할 수 있다.

다음 예제에서는 RMGR을 이용하여 Backup Set ID = 1에 해당하는 Backup Set을 삭제한다.

[예 11.23] Backup Set ID에 기반한 백업 삭제 시나리오

```
$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;

    SET_ID START_TIME
---------- --------------------------------------------------------------------
FINISH_TIME                                                         START_TSN
-------------------------------------------------------------------- ----------
FINISH_TSN RESETLOGS_TSN   BASE_SET   SIZE(KB) IS_PARTIAL IS_INCREMENTAL
---------- ------------- ---------- ---------- ---------- --------------
WITH_ARCHIVELOG
---------------
         1 2018/06/11
```

```
2018/06/11                                                          37093
     37109             0          0     453588 NO         NO
YES

         2 2018/06/11
2018/06/11                                                          37361
     37377             0          0     453588 NO         NO
YES

         3 2018/06/11
2018/06/11                                                          37390
     37406             0          0     453588 NO         NO
YES


3 rows selected.

SQL> quit
Disconnected.

$ tbrmgr delete --backup_set 1 -o /home/tbrdb/work/7/backup
================================================================
= Recovery Manager(RMGR) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
============================================
  RMGR - delete
============================================
DB connected
 #1 of #3 backup sets erased
RMGR delete ends

$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;

    SET_ID START_TIME
---------- --------------------------------------------------------------
FINISH_TIME                                                    START_TSN
-------------------------------------------------------------- ----------
FINISH_TSN RESETLOGS_TSN   BASE_SET   SIZE(KB) IS_PARTIAL IS_INCREMENTAL
---------- ------------ ---------- ---------- ---------- --------------
WITH_ARCHIVELOG
```

```
---------------
         2 2018/06/11
2018/06/11                                                          37361
     37377             0          0     453588 NO         NO
YES

         3 2018/06/11
2018/06/11                                                          37390
     37406             0          0     453588 NO         NO
YES


2 rows selected.
```

Backup Date에 기반한 백업 삭제 시나리오

RMGR은 사용자가 Backup Date로 지정한(--beforetime 옵션) 시간 이전에 생성된 모든 Backup Set을 삭제할 수 있다.

다음 예제에서는 RMGR을 이용하여 "2016년 06월 17일 12시" 이전에 생성된 모든 Backup Set을 삭제한다. Backup Date는 YYYYMMDDHH24MISS 형태로 명시한다.

[예 11.24] Backup Date에 기반한 백업 삭제 시나리오

```
$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;

    SET_ID START_TIME
---------- --------------------------------------------------------------------
FINISH_TIME                                                        START_TSN
-------------------------------------------------------------------- ----------
FINISH_TSN RESETLOGS_TSN   BASE_SET   SIZE(KB) IS_PARTIAL IS_INCREMENTAL
---------- ------------- ---------- ---------- ---------- --------------
WITH_ARCHIVELOG
---------------
         2 2018/06/11
2018/06/11                                                          37361
     37377             0          0     453588 NO         NO
YES

         3 2018/06/11
2018/06/11                                                          37390
```

```
     37406             0          0     453588 NO         NO
YES

2 rows selected.

SQL> quit
Disconnected.

$ tbrmgr delete --beforetime 20180612120000 -o /home/tbrdb/work/7/backup/
================================================================
= Recovery Manager(RMGR) starts                                =
=                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved. =
================================================================
============================================
  RMGR - delete
============================================
DB connected
 #2 of #2 backup sets erased
RMGR delete ends

$ tbsql sys/tibero

SQL> set line 200
SQL> col START_TIME for a20
SQL> col FINISH_TIME for a20
SQL> select * from V$BACKUP_SET a;

0 rows selected.
```

11.5. Tibero와 NetBackup 연동

NetBackup은 엔터프라이즈 백업 및 복구 소프트웨어로서, 다양한 환경에 대한 완벽하고 유연한 데이터 보호 솔루션을 제공한다.

Tibero에서 복구 관리자를 사용해서 Veritas NetBackup으로 백업 파일 저장 및 복원할 수 있다. 아래에서 소개할 Tibero와 NetBackup 연동을 위해서는 NetBackup과 Tibero가 정상적으로 설치되어 있어야 한다.

참고

1. NetBackup 연동을 위해서는 TmaxData 기술지원에 요청하여 NetBackup 연동이 가능한 Tibero를 별도로 제공 받아야 한다.

2. NetBackup 설치 및 관리에 대한 자세한 내용은 "NetBackup Documentation"을 참고한다.

11.5.1. NetBackup 환경설정

복구 관리자를 사용해서 NetBackup의 백업 파일 저장 및 복원하기 위해서 라이브러리를 연결하고, NetBackup 서버 및 클라이언트의 기본 정보를 설정해야 한다.

NetBackup에 필요한 Tibero 라이브러리 연결 설정

초기 상태에는 $TB_HOME/client/lib에 있는 dummy 라이브러리들과 연결되어 있다. dummy 라이브러리들은 사용하지 않으면 삭제하거나 직접 NetBackup 관련 라이브러리를 연결해주어야 한다.

- 32bit 환경
  - libnbclientcST32.so
  - libnbbasecST32.so
  - libxbsa.so
  - libvxcPBXST.so
- 64bit 환경
  - libnbclientcST.so
  - libnbbasecST.so
  - libxbsa64.so
  - libvxcPBXST.so

NetBackup 서버 및 클라이언트 기본 설정

- NetBackup 서버가 설치된 머신의 /etc/hosts 파일에 해당 머신의 호스트와 IP 주소 및 NetBackup 클라이언트가 설치된 머신들의 호스트와 IP 주소를 추가한다.
- NetBackup 서버가 설치된 머신에서 NetBackup이 설치된 경로에 db/altnames 디렉터리를 생성한 후 NetBackup 클라이언트가 설치된 머신들의 호스트를 이름으로 가지는 빈 파일을 생성한다.
- NetBackup 클라이언트가 설치된 머신의 /etc/hosts 파일에 NetBackup 서버가 설치된 머신의 호스트와 IP 주소를 추가하고, 해당 머신의 호스트와 IP 주소를 추가한다.
- NetBackup 클라이언트가 설치된 머신에서 NetBackup이 설치된 경로에 있는 bp.conf 파일에 다음의 내용을 추가한다.

```
SERVER=[NetBackup 서버가 설치된 머신의 호스트]
CLIENT_NAME=[NetBackup 클라이언트가 설치된 머신의 호스트]
```

복구 관리자를 위한 NetBackup 관리자 환경설정

다음은 복구 관리자를 사용하기 위해서 NetBackup 관리자 환경을 설정하는 방법에 대한 설명이다.

- NetBackup 관리자 콘솔이 설치된 머신의 /etc/hosts에 NetBackup 마스터가 설치된 머신의 호스트와 IP 주소를 추가한다.
- NetBackup 관리자 콘솔 실행한 후 NetBackup 마스터가 설치된 머신의 root와 해당하는 비밀번호로 접속한다.
- NetBackup 관리자 콘솔의 **[Master Server]** 탭에서 **'Configure Disk Storage Servers'**를 선택하여 NetBackup 마스터에서 사용할 디스크를 'AdvancedDisk'로 선택한다. ReadyStream 이후 실제 백업에 사용할 디렉터리를 선택한다.
- NetBackup 관리자 콘솔 **[Policies]** 탭에서 New Policy를 생성한다. Policy의 이름은 NBU_BACKUP_POLICY_NAME, NBU_ARCHIVE_POLICY_NAME iparam에서 설정한 값으로 생성한다.

  [Attributes] 탭의 **'Policy type'**을 'DataStore'로 생성하고 그외는 임의로 설정한다. **[Clients]** 탭에서 **[New]** 버튼을 클릭하고 NetBackup 클라이언트를 설치한 머신을 등록한다.
- NetBackup 관리자 콘솔의 **[Host Properties]** 탭에 있는 **[Clients]** 탭에 앞서 등록한 NetBackup 클라이언트 머신이 표시되고 실제 정보와 일치하는지 확인한다.
- 복구 관리자의 옵션 중 -p, --parallel 옵션을 사용할 수 있도록 아래와 같이 설정한다.
  - Storage unit 설정에서 해당 스토리지에 Maximum concurrent jobs를 parallel로 실행될 thread 개수의 최대값으로 설정한다(Tibero는 최대 16개를 parallel로 사용하고 있다).
  - Master server 설정에서 **[Global Attribute]** 탭에 있는 Maximum jobs per client 값을 하나의 클라이언트에서 parallel로 실행될 thread 개수의 최대값으로 설정한다.

- Master server 설정에서 **[Client Attribute]** 탭에 있는 Maximum data streams 값을 하나의 클라이언트에서 parallel로 실행될 thread 개수의 최대값으로 설정한다.
- Policy 설정에 있는 **[Schedules Tab]**의 스케줄마다 설정에서 Media multiplexing 값을 하나의 클라이언트에서 parallel로 실행될 thread 개수의 최대값으로 설정한다.

11.5.2. 복구 관리자 환경설정

NetBackup에서는 백업 파일에 대한 권한 관리가 이루어지 때문에 복구 관리자를 위한 파라미터들이 정상적으로 설정되지 않은 상태에서 백업이 수행되면 복구가 불가능해질 수 있다.

다음은 복구 관리자의 환경설정에 사용되는 파라미터에 대한 설명이다. 각 파라미터는 **동적 설정**이 가능하다.

파라미터	**설명**
USE_NBU_FOR_BACKUPSET	다음은 설정값에 대한 설명이다. – Y : 복구 관리자에 의한 BackupSet 생성은 모두 NetBackup 서버에 전송되어 생성된다. – N : NetBackup 서버에 저장된 BackupSet을 가져올 수 있다. (기본값) TAC 환경에서는 모든 노드가 동일한 값으로 설정되어 있어야 한다.
USE_NBU_FOR_ARCHIVELOG	설정 값이 Y일 경우 데이터베이스에서 생성되는 아카이브 로그가 NetBackup 서버에 전송되어 생성되며, 복구할 때에도 NetBackup 머신에 존재하는 아카이브 로그를 읽어 복구를 진행한다. (기본값: N) **[참고]** USE_NBU_FOR_ARCHIVELOG=Y일 경우 with-archivelog, archive-only, restore-archive-only 기능은 지원하지 않는다.
NBU_ARCHIVELOG_SEARCH	설정 값이 Y일 경우 NetBackup에 생성된 아카이브 로그들을 v$archive_dest_files로 조회 가능하고 이를 읽어 복구할 수 있다. (기본값: N)
NBU_OBJ_OWNER_NAME	NetBackup을 통해 생성되는 백업 파일의 권한을 해당 파라미터에 설정한 값으로 설정해주며, 복구할 때에는 해당 파라미터의 값이 아닌 클라이언트 머신에서 로그인한 username을 참고한다. (기본값: "")
NBU_OBJ_OWNER_GROUP_NAME	NetBackup을 통해 생성되는 백업 파일의 권한을 해당 파라미터에 설정한 값으로 설정해주며, 복구할 때에는 해당 파라미터의 값이 아닌 클라이언트 머신에서 로그인한 group anme을 참고한다.

파라미터	설명
	해당 값을 설정해주지 않으면 NBU_OBJ_OWNER_NAME으로 설정해 준 user name에만 권한이 있게된다. (기본값: "")
NBU_CLIENT_COUNT	Tibero 인스턴스가 위치하는 독립적인 머신을 숫자로 설정한다. 0에서 10 사이로 설정해주어야 하며, NBU_ARCHIVELOG_SEARCH를 Y로 설정한 경우 NBU_CLIENT_COUNT의 값과 NBU_CLIENT_HOSTNAME_#의 수가 같아야 한다. (기본값: 0)
NBU_CLIENT_HOSTNAME_#	SID#N을 인스턴스 ID로 갖는 Tibero 인스턴스가 위치하는 머신의 호스트로 #을 0부터 시작해 설정한다. (기본값: "")
NBU_SKIP_HOST_CHECK	설정값이 Y일 경우 Tibero 인스턴스가 위치하는 머신의 호스트와 NBU_CLIENT_HOSTNAME_0에 설정되어 있는 값이 같은지 비교하지 않는다. (기본값: N)
NBU_BACKUP_INST_SID	NBU_CLIENT_HOSTNAME_0에 설정된 머신의 호스트에 해당하는 Tibero 인스턴스의 SID로 설정한다. (기본값: "")

참고

for-standby 기능은 지원하지 않는다.

11.5.3. 복구 관리자를 이용한 NetBackup 사용 예제

연동 절차가 모두 완료된 후 복구 관리자로 백업을 수행했을 때 아래와 같이 NetBackup에 백업한다는 메시지가 찍히고 백업이 정상적으로 진행된다면 연동에 성공한 것이다.

[예 11.25] NetBackup 시나리오

```
$ tbrmgr backup
================================================================================
= Recovery Manager(RMGR) starts                                                =
=                                                                              =
= TmaxData Corporation Copyright (c) 2008-. All rights reserved.      =
================================================================================
RMGR '-n' option not used (USE_NBU_FOR_BACKUPSET=Y)
:backing up to RMGR_NBU_BACKUP_DEST= /tibero/

================================================================================
    RMGR - Backup (FULL)
================================================================================
Initializing the backup progress, it may take few minutes...

BACKUP   (set_id:  1, ts_id:  0, df_id:  0)
```

```
 100.00% |================================>|  12800/12800  blks   16.01s
Synchronizing...
BACKUP   (set_id:  1, ts_id:  1, df_id:  1)
 100.00% |================================>|  25600/25600  blks   17.02s
Synchronizing...
BACKUP   (set_id:  1, ts_id:  3, df_id:  2)
 100.00% |================================>|  12800/12800  blks   16.01s
Synchronizing...
BACKUP   (set_id:  1, ts_id:  4, df_id:  3)
 100.00% |================================>|   1280/1280   blks   15.04s
Synchronizing...

Switching an online logfile...

Backing up the control file...
Control file backup seucceeded

Database backup succeeded

RMGR backup ends
$ tbsql sys/tibero
SQL> select * from V$BACKUP_SET;

    SET_ID STATUS
---------- ------------------------------------
START_TIME           FINISH_TIME          ELAPSED(SEC)
-------------------- -------------------- ------------
START_TSN  FINISH_TSN RESETLOGS_TSN   SIZE(MB)   BASE_SET BACKUP_TYPE
---------- ---------- ------------- ---------- ---------- ------------
BACKUP_OPTION                           PARTIAL_BACKUP_OPTION
--------------------------------------- ---------------------------------------
BACKUP_PATH
--------------------------------------------------------------------------------
          1 COMPLETED
2020/12/30           2020/12/30                    83
36321      36338             0            445          0  FULL
NET_BACKUP                              NONE
/tibero/

1 row selected.
```

앞서 기술한 미지원 기능 목록을 제외한 나머지 복구 관리자 기능 역시 수행하여 위와 동일한 메시지를 확인할 수 있다면 NetBackup 연동에 성공한 것으로 볼 수 있다. 복구 관리자에서 실제로 생성된 백업 파일 이름, 갯수와 NetBackup에서 생성되는 파일 이름, 갯수가 다른 사실은 복구에 영향을 주지 않는다.

11.6. 플래시백 데이터베이스

Tibero는 다양한 백업 및 복구 시나리오를 제공하지만, 백업 파일 용량이 매우 클 때는 백업 파일 복원과 이후 복구까지 굉장히 오랜 시간이 걸릴 수 있다. 특히, 유저의 실수를 만회하고자 데이터베이스를 아주 가까운 과거로 되돌리기 위해서도 백업 파일을 복원해야 한다는 점 때문에 시간 소요가 너무 크다.

이러한 면을 보완하기 위하여 Tibero는 백업 파일 없이도 DB 전체를 과거로 되돌릴 수 있는 플래시백 데이터베이스(Flashback Database) 기능을 제공한다.

11.6.1. 기본 기능 및 특징

플래시백 데이터베이스는 단어 의미 그대로 데이터베이스를 가까운 과거로 빠르게 되돌릴 수 있는 기능이다. SQL 명령어 **ALTER DATABASE** 구문으로 플래시백 로그 파일 관리 및 플래시백 수행 가능하다.

Tibero 플래시백 데이터베이스 기능의 특징은 다음과 같다.

- 데이터베이스를 과거로 되돌리기 위한 플래시백 로그 파일을 따로 생성하고 관리한다. 추가적인 로그 파일을 생성하기 때문에 데이터베이스 성능 저하는 불가피하다. 플래시백 로그 파일은 “11.1. Tibero 구성 파일”을 참고한다.
- Flashback Marker 시점까지 데이터베이스를 과거로 되돌린 후 사용자가 입력한 시점까지 불완전 복구한다. Flashback Marker는 블록의 변경되는 이미지를 플래시백 로그 파일에 로깅하는 기준이다.

  사용자가 SQL 구문을 통해 직접 추가할 수 있으며, 초기화 파라미터 FLASHBACK_MARKER_INTERVAL(단위: 초)를 설정하여 백그라운드 작업으로 주기적으로 남기게 할 수 있다. Flashback Marker는 자주 남길 수록 플래시백 데이터베이스 기능 수행이 빠르지만, 그만큼 플래시백 로그가 많이 생성되기에 데이터베이스 운영 성능이 더 저하된다.
- 백업 파일들 복원하지 않고 특정 과거 시점으로 데이터베이스를 회귀시킬 수 있다.
- 백업 파일 복원 후 불완전 복구와 효과는 같지만, 총 작업 시간이 백업 파일 복원 시간만큼 절감된다.
- 현재의 Resetlogs TSN 이전으로도 회귀시킬 수 있다. 이 특징을 사용하면 사이트의 운영 장비가 아닌 별도의 개발 장비에서 Resetlogs 부팅으로 특정 시나리오 테스트 후 데이터베이스를 재생성하지 않고 빠르게 기존 상태로 원복할 수 있다.
- Tibero Standby Cluster 사용 중 switchover/failover를 발생할 때 기존 Primary Database를 재구축하지 않고 빠르게 stanby로 전환시킬 수 있다.

11.6.2. 전제 조건 및 제약 사항

플래시백 데이터베이스는 현재 존재하는 파일을 블록 단위로 과거로 돌리는 작업이기 때문에 다음과 같은 사항들이 고려되어야 한다.

- 지워지지 않고 실제 존재하는 정상적인 데이터 파일에 대해서만 플래시백 가능하다. 손상된 데이터 파일을 복구하는 것은 불가하다.

- 플래시백하는 시점 이후로 추가된 파일은 자동으로 데이터베이스에서 삭제된다. 물리적 파일은 삭제하지 않기 때문에 직접 관리해줘야한다.
- Drop된 데이터 파일을 Drop되기 전으로 플래시백하려면, 해당 데이터 파일의 백업 파일이 존재해야만 한다. 플래시백 수행 후 백업 파일을 복원하라는 가이드가 화면에 안내된다.
- 플래시백하는 시점 기준으로 테이블 스페이스는 rename되지만, 데이터 파일은 보존된다.
- Offline된 테이블 스페이스 및 데이터 파일은 플래시백 대상에서 제외된다. 추후 drop해야 한다.
- 크기가 Extend된 데이터 파일은 다시 줄어들지 않는다.
- Shrink된 데이터 파일은 플래시백 불가하다. 단, 해당 데이터 파일을 offline시키면 이를 제외한 나머지 데이터베이스에 대해서는 플래시백 가능하다. 이후 해당 파일에 대해서는 직접 백업 파일을 복원하여 불완전 복구해야한다.
- 컨트롤 파일을 재생성했다면 이전까지 축적한 플래시백 로그 정보는 사라진다. 플래시백 로깅을 다시 활성화해야 하며, Flashback Marker도 처음부터 다시 생성된다. 즉, 컨트롤 파일 재생성 이전으로는 플래시백 데이터베이스 불가하다. 컨트롤 파일 재생성 이전에 생성된 플래시백 로그 파일들은 백업 데이터 파일과 컨트롤 파일이 있지 않는 이상 사용할 수 없다.
- 백업 데이터 파일과 컨트롤 파일로 Media Recovery 후에도 기존 플래시백 로그 파일을 사용하고자 한다면, 완전 복구를 통해 Resetlogs 부팅은 하지 않아야 하며, Flashback Marker 이후의 플래시백 로그 파일들의 존재가 보장되어야 한다.

  일반적으로 백업 데이터 파일과 컨트롤 파일을 사용하여 데이터베이스를 복구할 때 Flashback Marker를 초기화여 플래시백 로깅을 새롭게 시작하는 것이 안전한다. 추가적으로 플래시백 로깅을 활성화한 채로 Media Recovery를 수행할 때 Redo 로그를 기반으로 플래시백 로그도 복구하기 때문에 복구 성능이 매우 저하될 수 있다. 따라서 플래시백 로깅을 비활성화하는 것을 추천하는데, 비활성화 할 경우 플래시백 로그 파일 정보들은 모두 초기화된다.
- NOLOGGING 모드는 권장하지 않는다. NOLOGGING 모드로 데이터베이스를 사용하면 플래시백은 가능하지만 NOLOGGING 모드를 사용한 기간 동안의 플래시백 로그가 없기 때문에 해당 내용들은 플래시백이 불가하여 정합성이 깨진 미래의 데이터베이스로 남아있을 뿐이다.
- 데이터베이스를 정상 종료해야만 MOUNT 모드에서 플래시백 데이터베이스 기능이 실행 가능하다. 정합성이 보장된 상태에서 플래시백해야 하기 때문이다.

11.6.3. 플래시백 데이터베이스 실행 예제

본 절에서는 다음의 시나리오들을 통해서 플래시백 데이터베이스를 위한 플래시백 로그 파일 생성, 로깅 활성화, 그리고 실제 플래시백 데이터베이스 수행을 설명한다. 이들은 모두 MOUNT 상태에서만 수행 가능하다.

- 플래시백 로그 파일 생성과 로깅 활성화 시나리오
- 플래시백 데이터베이스 실행 시나리오

- 플래시백 데이터베이스 실행 중 MISSING 파일 생성 시나리오

플래시백 로그 파일 생성과 로깅 활성화

일반 로그 파일 구조, 구성과 동일하기에 이를 관리하는 원리도 같다. 단, 현재 플래시백 로그 파일 다중화 기능은 지원하지 않기 때문에 플래시백 로그 멤버는 그룹당 하나만 가능하다. 플래시백 로그 파일에 대해서는 일반 로그 파일인 "3.3. 로그 파일"을 참고한다.

플래시백 로그 파일을 추가한 후에는 해당 스레드와 플래시백 로깅을 활성화해야 운영 중 플래시백 로깅이 동작한다. 플래시백 로깅을 활성화시키기 위해서는 초기화 파라미터인 FLASHBACK_LOG_BUFFER가 세팅되어 있어야한다.

FLASHBACK_LOG_BUFFER 값은 일반적으로 LOG_BUFFER 값의 2배가 권장된다. 자세한 값은 실제 사이트의 운영 방식에 맞게 튜닝이 필요하다.

[예 11.26] 플래시백 로그 파일 생성과 로깅 활성화 시나리오

```
ALTER DATABASE ADD FLASHBACK LOGFILE THREAD 0 GROUP 1 '/usr/tibero/log/fblog001.fb'

SIZE 512M;
ALTER DATABASE ADD FLASHBACK LOGFILE THREAD 0 GROUP 2 '/usr/tibero/log/fblog002.fb'

SIZE 512M;
ALTER DATABASE ADD FLASHBACK LOGFILE THREAD 0 GROUP 3 '/usr/tibero/log/fblog003.fb'

SIZE 512M;
ALTER DATABASE ENABLE PUBLIC FLASHBACK THREAD 0;
ALTER DATABASE FLASHBACK LOGGING ON;
```

플래시백 데이터베이스 실행 시나리오

현재 TSN 값이 155670인 데이터베이스를 가까운 과거인 150000으로 플래시백하려면 데이터베이스를 정상 종료 후 MOUNT 모드에서 다음을 구문을 수행한 후 완료 여부를 확인한다.

[예 11.27] 플래시백 데이터베이스 실행 시나리오

```
SQL> ALTER DATABASE FLASHBACK TO TSN 150000;
Database altered.
```

플래시백 데이터베이스 실행 중 MISSING 파일 생성 시나리오

현재 TSN 값이 155670인 데이터베이스를 가까운 과거인 150000으로 플래시백 수행 도중 다음과 같은 에러가 발생할 때 Drop된 파일을 되살려야 한다. 에러와 함께 안내되는 절차대로 수행하면 데이터베이스가 원하던 시점으로 온전히 플래시백된다.

[예 11.28] 플래시백 데이터베이스 실행 중 MISSING 파일 생성 시나리오

```
SQL> ALTER DATABASE FLASHBACK TO TSN 150000;
TBR-24127: Database flashed back to the target TSN %1$u but not completed,
because CF-DD correction is required due to the originally existed at the target
TSN or newly added datafiles during roll-forward.
```

```
1) Find MISSING datafiles from v$datafile.
2) Restore the corresponding backup datafiles.
3) If they were newly added after the TSN %lu, create the physical files with
the datafile id each.
  : SQL> ALTER DATABASE CREATE DATAFILE <file_id>;
4) Rename(OS cmd) the corresponding files at DB_CREATE_FILE_DEST to their
original names."
5) Rename(SQL) them with each of the names at step 2 and 4."
  : SQL> ALTER DATABASE RENAME FILE 'current_file_path' to 'renamed_file_path';"
6) Request flashback database query again as followed.
  : SQL> ALTER DATABASE FLASHBACK TO TSN 150000 CONTINUE;
```

제12장 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.

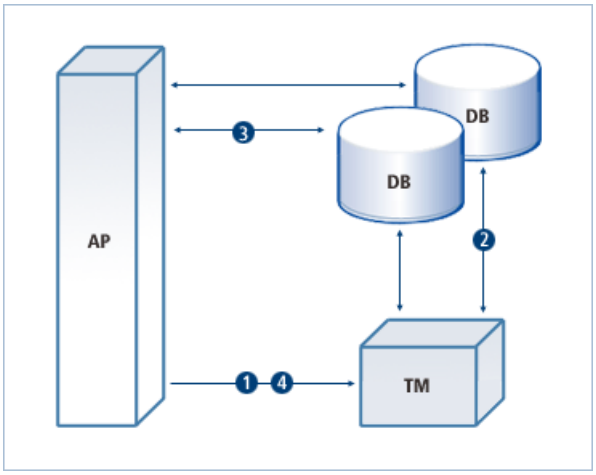
이렇게 여러 개의 노드 또는 다른 종류의 데이터베이스가 참여하는 하나의 트랜잭션을 **분산 트랜잭션(Distributed Transaction)**이라고 한다. Tibero에서는 분산 트랜잭션을 처리하기 위해 **XA**와 **데이터베이스 링크(DBLink)**를 통해 지원한다.

12.1. XA

Tibero는 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 12.1] 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.

12.2. 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 **롤백** 메시지를 보내 해당 작업을 롤백한다.
커밋	모든 데이터베이스 노드로부터 **prepare ok** 메시지를 받으면 다시 모든 데이터베이스 노드에 **커밋** 메시지를 보내고 모든 작업을 커밋한다.

12.3. 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 또는 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 되는데 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 롤백시킴으로써 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.

12.3.1. DBA_2PC_PENDING 뷰

DBA_2PC_PENDING 뷰는 현재 정체되고 있는 XA 트랜잭션 브랜치(XA Transaction Branch)의 정보를 보여주는 뷰이다.

다음은 XA 트랜잭션 브랜치의 정보를 조회하는 예이다. 본 예제에서는 XID와 FAIL_TIME 정보를 이용하여 커밋과 롤백을 수행할 브랜치를 선택한다.

[예 12.1] DBA_2PC_PENDING 뷰 조회

```
SQL> SELECT LOCAL_TRAN_ID, XID, STATUS FROM DBA_2PC_PENDING;

LOCAL_TRAN_ID
-------------------------------------
XID
-------------------------------------
STATUS
---------------
2.16.18
1.1000.1000
PREPARED

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 USGMT_ID || '.' || SLOTNO || '.' || WRAPNO as XID,
     TO_CHAR(FORCE_TIME, 'YYYY-MM-DD HH24:MI:SS') as FORCE_TIME
     FROM _DD_PENDING_TX;

XID
-----------------------------------
FORCE_TIME
---------------

1.1000.1000
2018-01-01 15:00:00


1 selected
```

해당 XA 트랜잭션 브랜치의 정보는 TM이 **xa_forget**을 이용하여 더 이상 XA 트랜잭션 브랜치 정보가 필요 없다고 판단하면 해당 정보를 제거한다. 자원 관리자(RM: Resource Manager, 이하 RM)에서는 TM의 요청이 있기 전까지는 XA 트랜잭션 브랜치의 정보를 제거하지 않는다.

12.4. 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.

12.4.1. 데이터베이스 링크 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_2'는 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 예에서 using 절 이후의 'remote_1'는 연결할 데이터베이스를 가리키는 이름으로 tbdsn.tbr 파일에 해당 데이터베이스의 연결 정보가 저장되어 있어야 한다. 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;
```

12.4.2. 원격 데이터베이스 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. 패스워드는 작은따옴표(' ') 사이에 패스워드가 입력된 경우에만 유효하다. 예를 들어 아래 예제에서 "password"로 패스워드가 입력된 경우에는 데이터베이스 링크 생성에 실패하게 된다.

설정 방법	설명
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';
```

12.4.3. 게이트웨이

데이터베이스 링크를 통해 질의를 수행할 때 데이터베이스 링크의 대상이 Tibero가 아닌 다른 DBMS라면 각각의 DBMS를 위한 게이트웨이를 통해 데이터베이스 링크를 수행할 수 있다.

Tibero 서버는 다른 DBMS에 필요한 질의를 해당 게이트웨이에 전달한다. 게이트웨이는 원격 DBMS에 접속하여 Tibero 서버로부터 전달 받은 질의를 수행하고 그 결과를 다시 Tibero 서버로 전송한다. 다른 DBMS로의 데이터베이스 링크 기능을 사용하는 경우에는 해당 DBMS에 대한 게이트웨이 바이너리와 설정 파일이 필요하다.

본 절에서는 DBMS 벤더별로 게이트웨이의 종류를 설명하고, 게이트웨이와 Tibero 서버가 같은 머신에 존재하는 경우와 다른 머신에 존재하는 경우를 알아본다. 또한 게이트웨이에서 제공하는 옵션 및 로깅도 설명한다.

DBMS 벤더별 게이트웨이

다음은 Tibero에서 데이터베이스 링크 기능을 지원하고 있는 다른 DBMS의 종류와 게이트웨이 바이너리 명이다.

DBMS 벤더명	게이트웨이 바이너리명	프로그래밍 언어	DBMS 버전
Oracle	gw4orcl	C	Oracle 9i, 10g, 11g, 12c, 18c, 19c **[참고]** – Oracle 의 경우는 64-bit OS에서만 지원한다. – client/bin/에 있는 gw_install.sh 스크립트를 수행하여 쉽게 환경설정을 할 수 있다.
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
GREENPLUM	tbgateway.jar	Java	GREENPLUM or PostgreSQL
MySQL	tbgateway.jar	Java	MySQL or MariaDB

각각의 게이트웨이 바이너리는 DBMS의 버전에 따라 다를 수 있기 때문에 버전에 맞는 게이트웨이 바이너리를 사용할 것을 권장한다.

참고

1. DB2 게이트웨이의 경우 HP PA-RISC 머신은 지원하지 않는다.

2. MySQL 게이트웨이의 경우 mysql-connector-java 5.1.19 이상의 버전 사용해야 한다.

3. 게이트웨이를 통해 Oracle의 프러시저를 수행하는 경우 해당 프러시저 내에서 local commit을 수행할 수 없다.

게이트웨이 프로세스 생성 방식

게이트웨이는 연결할 DBMS가 제공하는 라이브러리가 필요하다. 라이브러리를 Tibero 서버가 설치된 곳에서 사용할 수 있다면 Tibero 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 바이너리 위치에 대한 절대 경로이다. 게이트웨이 바이너리가 $TB_HOME/client/bin 디렉터리에 있는 경우 바이너리명만 명시할 수 있다.
CONFIG	게이트웨이 설정 파일 위치에 대한 절대 경로이다.
TARGET	DBMS별로 다음과 같이 의미하는 것이 다르다. – Oracle 서버 : 네트워크 서비스명이다. – DB2 서버 : 데이터베이스명이다.
TX_MODE	글로벌 트랜잭션(Global Transaction) 또는 로컬 트랜잭션(Local Transaction)으로 처리할지의 여부를 설정한다. 글로벌 트랜잭션은 커밋을 요청할 때 Two-phase 커밋으로 동작하고, 로컬 트랜잭션은 Two-phase 커밋으로 동작하지 않는다. TX_MODE의 값은 처리 여부에 따라 다음과 같이 설정할 수 있다. – GLOBAL : 글로벌 트랜잭션인 경우 – LOCAL : 로컬 트랜잭션인 경우
CHARACTER_SET	게이트웨이의 문자 집합을 재설정할 때 설정한다. Tibero 서버에서 지원하는 문자 집합을 사용할 수 있다.
SKIP_CHAR_CONV	게이트웨이의 초기화 파라미터 SKIP_CHAR_CONV 값을 재설정한다. Y 또는 y로 설정된 경우에 게이트웨이의 SKIP_CHAR_CONV의 값이 Y로 재설정되고, N 또는 n으로 설정된 경우에 게이트웨이의 SKIP_CHAR_CONV의 값이 N으로 재설정된다.

항목	설명
	TARGET 서버가 Oracle 서버인 경우에만 설정 가능하다. SKIP_CHAR_CONV에 대한 자세한 사항은 게이트웨이 설정(ORACLE, DB2)을 참고한다.

멀티 스레드 서버 방식

사용자는 Tibero 서버와 같은 머신 또는 원격에 있는 머신에서 게이트웨이를 멀티 스레드 서버 방식으로 시작할 수 있다. Tibero 서버의 세션은 tbdsn.tbr 파일에 명시된 접속 정보를 통해 게이트웨이와 TCP/IP 통신을 한다. 멀티 스레드 서버 방식의 게이트웨이는 Tibero 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 각 벤더별 서버와 연결하는 데이터베이스 링크를 사용하기 위해 tbdsn.tbr 파일을 설정하는 예와 항목에 대한 설명이다.

- Oracle 서버

```
ora_link_remote=(
                (GATEWAY=(LISTENER=(HOST=12.34.56.78)
                                   (PORT=9999))
                         (TARGET=orcl)
                         (TX_MODE=GLOBAL))
)
```

- DB2 서버

```
db2_link_remote=(
                (GATEWAY=(LISTENER=(HOST=12.34.56.78)
                                   (PORT=9999))
                         (TARGET=sample)
                         (TX_MODE=GLOBAL))
)
```

- MS-SQL 서버

```
mssql_link_remote=(
                  (GATEWAY=(LISTENER=(HOST=12.34.56.78)
                                     (PORT=9093))
                           (TARGET=12.34.56.87:1433:master)
                           (TX_MODE=LOCAL))
)
```

- Sybase ASE 서버

```
ase_link_remote=(
                (GATEWAY=(LISTENER=(HOST=12.34.56.78)
```

```
                                    (PORT=9093))
                            (TARGET=12.34.56.87:5000:master)
                            (TX_MODE=LOCAL))
)
```

- GREENPLUM 서버

```
gp_link_remote=(
                  (GATEWAY=(LISTENER=(HOST=12.34.56.78)
                                    (PORT=9093))
                            (TARGET=12.34.56.87:5432:mydb)
                            (TX_MODE=LOCAL))
)
```

- MySQL 서버

```
mysql_link_remote=(
                  (GATEWAY=(LISTENER=(HOST=12.34.56.78)
                                    (PORT=9093))
                            (TARGET=12.34.56.87:3306:mydb)
                            (TX_MODE=LOCAL))
)
```

항목	설명
LISTENER	게이트웨이의 접속 정보이다. – 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 서버: 네트워크 서비스명이다. – DB2 서버 : 데이터베이스명이다. – MS-SQL 서버 : 서버의 연결 정보(IP:PORT:DATABASE NAME)이다. – Sybase ASE 서버 : 서버의 연결 정보(IP:PORT:DATABASE NAME)이다. – GREENPLUM 서버 : 서버의 연결 정보(IP:PORT:DATABASE NAME)이다. – MySQL 서버 : 서버의 연결 정보(IP:PORT:DATABASE NAME)이다.
TX_MODE	글로벌 트랜잭션(Global Transaction) 또는 로컬 트랜잭션(Local Transaction)으로 처리할지의 여부를 설정한다.

항목	설명
	글로벌 트랜잭션은 커밋을 요청할 때 Two-phase 커밋으로 동작하고, 로컬 트랜잭션은 Two-phase 커밋으로 동작하지 않는다. TX_MODE의 값은 처리 여부에 따라 다음과 같이 설정할 수 있다. – GLOBAL : 글로벌 트랜잭션인 경우 – LOCAL : 로컬 트랜잭션인 경우
CHARACTER_SET	게이트웨이의 문자 집합을 재설정할 때 설정한다. Tibero 서버에서 지원하는 문자 집합을 사용할 수 있다. C 언어를 사용하는 게이트웨이인 경우에만 설정 가능하다.
SKIP_CHAR_CONV	게이트웨이의 초기화 파라미터 SKIP_CHAR_CONV의 값을 재설정할 때 설정한다. Y 또는 y로 설정된 경우에 게이트웨이의 SKIP_CHAR_CONV의 값이 Y로 재설정되고, N 또는 n으로 설정된 경우에 게이트웨이의 SKIP_CHAR_CONV의 값이 N으로 재설정된다. TARGET 서버가 Oracle 서버인 경우에만 설정 가능하다. SKIP_CHAR_CONV에 대한 자세한 사항은 게이트웨이 설정(ORACLE, DB2)을 참고한다.
BYTES_CHARSET	TARGET 서버와 게이트웨이 사이의 문자 집합 변환에 사용되는 게이트웨이의 문자 집합을 게이트웨이의 초기화 파라미터 ENCODING의 값과 다른 문자 집합으로 재설정할 때 설정한다. Tibero 서버에서 지원하는 문자 집합을 사용할 수 있다. Tibero 서버가 한글을 지원하지 않는 문자 집합과 한글 데이터를 동시에 사용하고 있는 특수한 경우에 사용한다. Java 프로그래밍 언어를 사용하는 게이트웨이인 경우에만 설정 가능하다.

원격에 있는 게이트웨이를 사용하기 위해서는 먼저 원격에 있는 게이트웨이를 실행시켜야 한다.

- Oracle 서버

```
$ gw4orcl
```

Oracle과 DB2 서버용 게이트웨이의 경우 -q 옵션을 통해 기존에 수행 중인 게이트웨이를 종료할 수 있다.

```
$ gw4orcl -q
```

- MS-SQL 서버

```
$ tbgw
```

게이트웨이 관련 디렉터리 구조(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\gateway'로 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_HOME이라고 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 위치시킨다.

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
DB2	gw4db2.log	gw4db2_lsnr.log

게이트웨이 설정(ORACLE, DB2)

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
FETCH_SIZE=32k
```

옵션	설명
CHARACTER_SET	게이트웨이의 문자집합을 설정한다. 이 값이 설정되지 않은 경우 TB_NLS_LANG 환경변수에 정의된 값을 사용한다. (기본값: MSWIN949)
FETCH_SIZE	데이터베이스에 질의 처리를 할 때 한 번에 가져오는 데이터의 크기를 설정한다. (기본값: 32KB, 최댓값: 64KB)
IGNORE_WARNING	원격 데이터베이스에서 발생한 경고 메시지를 무시할지 여부를 설정한다. (기본값: N)
IP_VERSION	리스너가 사용하는 IP 프로토콜을 명시하기 위해 사용한다. – 4 : IPv4를 사용하는 경우 설정 값이다. (기본값) – 6 : IPv6를 사용하는 경우 설정 값이다. 대상 DBMS가 Oracle인 경우에만 IPv6 사용이 가능하다.
LOG_DIR	게이트웨이의 로그 파일을 저장할 경로를 설정한다. (기본값: ${TBGW_HOME}/{*DBMS 벤더명*}/log)
LOG_LVL	로그 파일에 남길 로그 레벨을 설정한다. (기본값: 2) **[참고]** 게이트웨이 프로세스가 이미 실행 중인 경우 게이트웨이 실행 명령어에 "-l [0~6 사이의 값]" 옵션을 주어 동적으로 로그 레벨 변경이 가능하다. `gw4orcl -l 4`
MAX_LOG_BACKUP_SIZE	로그 백업 기능을 사용하는 경우 백업 로그 파일들의 최대 크기이다. (기본값: 0, 단위: Byte) – 값이 0인 경우 백업된 로그 파일들의 크기 제한이 없다. – 값을 명시한 경우 백업 로그 파일들의 크기의 합이 설정된 최대 크기를 초과하면 오래된 순서로 로그 파일을 지운다.
MAX_LOG_SIZE	로그 파일의 최대 크기이다. (기본값: 0, 단위: Byte) – 값이 0인 경우 로그 파일의 최대 크기를 설정하는 데 제한이 없다. – 값을 명시한 경우 로그 파일이 설정된 최대 크기를 초과하면 로그 파일을 백업한다.
DUMP_FILE	게이트웨이의 allocator dump 파일명과 경로를 설정한다. (기본값: ${TBGW_HOME}/{*DBMS 벤더명*}/dump)

옵션	설명
	[참고] 게이트웨이 프로세스가 이미 실행 중인 경우 게이트웨이 실행 명령어에 "-ad" 옵션을 주어 allocator dump를 남길 수 있다. `gw4orcl -ad`
ORACLE_FETCH_SIZE	Oracle용 게이트웨이에만 사용 가능한 옵션이다. Oracle에 질의 처리를 할 때, 한 번에 가져오는 데이터의 크기이다. FETCH_SIZE보다 우선으로 적용된다. (기본값: 32KB, 최댓값: 1MB)
QUERY_WITH_UR	DB2용 게이트웨이에만 사용 가능한 옵션이다. 쿼리에 WITH UR 구문을 추가할지 여부를 설정한다. (기본값: N)
SKIP_CHAR_CONV	Oracle용 게이트웨이에만 사용 가능한 옵션이다. Oracle에서 한글을 지원하지 않는 문자 집합과 한글 데이터를 동시에 사용하고 있는 특수한 경우에 사용된다. 값이 Y인 경우 Oracle 데이터베이스에 있는 데이터를 문자 집합 변환 없이 가져온다. (기본값: N)
VALIDATION_IDLE_TIME	Oracle용 게이트웨이에만 사용 가능한 옵션이다. (기본값: 0, 단위: 초) – 0이 아닌 값을 설정할 경우 설정한 만큼의 시간 간격마다 OCI Ping을 수행하여 Target 서버와의 Connection을 유지한다. (최댓값: 2,678,400) – 값이 0인 경우 OCI Ping을 수행하지 않는다.

다음은 게이트웨이를 리스너 모드로 사용할 때 설정할 수 있는 옵션이다.

항목	설명
LISTENER_PORT	IPv4를 사용하거나 IPv6를 사용하는 경우 그에 대한 리스너 포트 번호를 설정한다. 설정한 포트 번호의 값으로 포트가 오픈되며, 설정값+1의 추가 포트가 Statement Cancel을 처리하기 위해 오픈된다. (기본값: 9999)
MIN_POOL_SIZE	동시에 접속 가능한 최소 세션 개수를 설정한다. (기본값: 10)
MAX_POOL_SIZE	동시에 접속 가능한 최대 세션 개수를 설정한다. (기본값: 100, 최댓값: 1024)

게이트웨이 관련 디렉터리 구조(MS-SQL, Sybase ASE, GREENPLUM, MySQL)

사용자는 ${TB_HOME}/client/bin에 있는 tbJavaGW.zip 파일을 설치할 디렉터리에 복사한 후 압축을 해제한다. 기본적으로 타깃 데이터베이스에 대한 JDBC 드라이버 파일(Sybase: jconn3.jar, MS-SQL: sqljdbc.jar, GREENPLUM: postgresql-8.4-701.jdbc3.jar)은 제공하지 않는다. 따라서 사용자는 해당 서버의

JDBC 드라이버 파일을 구한 후 LIB 디렉터리에 복사해야 한다. 일반적으로 해당 서버의 홈페이지에서 다운로드 받을 수 있다.

게이트웨이가 사용하는 설정 파일과 로그 파일이 존재하는 디렉터리 구조는 다음과 같다.

```
설치 디렉터리
    |--- tbJavaGW
            |--- tbgw
            |--- jgw.cfg
            |--- jgwlog.properties
            |--- jgw_service.bat
            |--- prunsrv.exe
            |--- lib
                  |--- tbgateway.jar
                  |--- commons-collections.jar
                  |--- commons-daemon-1.0.6.jar
                  |--- commons-pool.jar
                  |--- log4j-1.2.15.jar
                  |--- jconn3.jar
                  |--- sqljdbc.jar
                  |--- postgresql-8.4-701.jdbc3.jar
            |--- log
                  | --- 게이트웨이의 로그 파일
```

tbJavaGW/tbgw

Java 게이트웨이를 실행시키는 스크립트 파일이다.

tbJavaGW/jgw.cfg

게이트웨이 설정 파일이다. 사용자가 게이트웨이와 관련된 설정 값을 변경하고 싶을 때 생성하며 위의 디렉터리 구조에 맞게 위치시킨다.

tbJavaGW/jgwlog.properties

로그에 대한 설정 파일이다. 로그 파일의 크기와 로그 레벨 등을 설정할 수 있다. 자세한 형식은 LOG4J를 참고한다.

tbJavaGW/jgw_service.bat

Windows 환경에서 게이트웨이를 서비스로 사용할 수 있다. 서비스에 등록/삭제를 해주는 실행 파일이다. 이 파일을 실행하기 위해선 tbJavaGW/prunsrv.exe 파일이 반드시 있어야 한다.

tbJavaGW/prunsrv.exe

Windows 환경에서 게이트웨이를 서비스에 등록/삭제를 해주는 실행 파일이다. tbJavaGW/jgw_service.bat 파일을 실행하기 위해서 반드시 필요한 파일이다. 이 파일은 기본으로 제공되지 않으며 Apache Commons의 Daemon에서 다운받을 수 있다.

tbJavaGW/lib

Java 게이트웨이에서 사용하는 JAR 파일이 있는 디렉터리이다. 타깃 데이터베이스의 JDBC 드라이버도 해당 디렉터리에 있어야 한다.

tbJavaGW/log

게이트웨이와 관련된 로그 파일이 생성된다.

게이트웨이 설정(MS-SQL SERVER, Sybase ASE, GREENPLUM, MySQL)

jgw.cfg 파일에 초기화 파라미터의 설정 값을 명시함으로써 게이트웨이와 관련된 설정을 변경할 수 있다.

다음은 게이트웨이를 설정하는 예이다.

<<jgw.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	설명
DATABASE	타깃 데이터베이스를 설정한다. – SQL_SERVER : MS-SQL SERVER (기본값) – ASE : Sybase ASE – GREENPLUM : GREENPLUM – MYSQL : MySQL
LISTENER_PORT	리스너의 포트 번호를 설정한다. 일반적인 요청의 처리를 위해 설정한 번호의 포트를 사용하며, Statement cancel 등의 제어 요청 처리를 위해 설정에 1을 더한 번호의 포트를 추가로 사용한다. (기본값: 9093)
INIT_POOL_SIZE	게이트웨이가 시작할 때 미리 생성할 워킹 스레드의 개수를 설정한다. (기본값: 10)
MAX_POOL_SIZE	게이트웨이가 최대로 생성할 수 있는 워킹 스레드의 개수를 설정한다. (기본값: 100)
MAX_CURSOR_CACHE_SIZE	워킹 스레드 당 최대로 캐시 가능한 커서의 개수를 설정한다. (기본값: 100)
ENCODING	Tibero 서버의 세션에 문자열을 전달할 때 사용할 인코딩을 설정한다. 단, Tibero 서버의 문자 집합과 일치시켜야 한다. 설정할 수 있는 문자 집합은 다음과 같다. – ASCII – EUC-KR

초기화 파라미터	설명
	– MSWIN949 (기본값) – UTF-8 – UTF-16 – SHIFT-JIS
MAX_LONGVARCHAR	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. (기본값: 4KB, 최댓값: 32KB)
MAX_LONGRAW	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. (기본값: 4KB, 최댓값: 32KB)
TARGET_DB_FETCH_SIZE	TARGET DB에서 한 번에 FETCH해 올 row 수를 설정한다. (기본값: 32) MySQL의 경우 -2147483648(interger.MIN_VALUE)로 설정하면 1 row씩, 그 외의 값이면 모든 row를 한 번에 FETCH해온다.
VALIDATION_QUERY	게이트웨이에서 타깃 데이터베이스로의 연결이 유효한지 확인하기 위한 SQL 문장을 지정한다. 이 값이 설정되어 있다면 다음의 경우에 연결 확인 과정이 수행된다. – Tibero 서버로부터 연결 확인 요청이 오는 경우 – 동일한 연결 내에서 마지막 수행 이후 VALIDATION_IDLE_TIME 설정에 지정된 시간동안 새로운 요청이 오지 않는 경우
VALIDATION_IDLE_TIME	마지막 요청 이후 연결 유효 확인을 수행할 때까지의 대기 시간을 설정한다. 단위는 밀리초이며, 0은 무제한을 의미한다. (기본값: 0, 최솟값: 60000) VALIDATION_QUERY 항목이 설정되어 있는 경우에만 함께 적용된다.

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)
```

12.4.4. 데이터베이스 링크 사용

데이터베이스 링크를 통해 접근할 수 있는 데이터베이스 객체의 종류는 다음과 같다.

- 테이블(LOB 타입의 컬럼에는 접근할 수 없다.)
- 뷰
- 시퀀스

데이터베이스 링크를 통해 Oracle의 LONG 타입 컬럼을 접근할 때 LONG 타입의 컬럼은 항상 마지막 컬럼으로 설정하여 접근해야 한다. 데이터베이스 링크를 사용할 수 있는 SQL 문장은 SELECT, INSERT, UPDATE, DELETE 이다. 단, SELECT 문에서 FOR UPDATE 절은 사용할 수 없다.

12.4.5. Global Consistency

Homogeneous DB Link로 구성된 분산 트랜잭션은 **Global Consistency**를 제공한다. 이를 위해 분산 트랜잭션에 참여한 데이터베이스 간에 TSN을 동기화하는 작업을 한다. TSN을 동기화하는 작업은 데이터베이스 사이에 메시지를 교환할 때 이루어지며, 커밋할 때에는 모든 노드의 Commit TSN을 동일하게 동기화해야 한다.

12.4.6. 데이터베이스 링크 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.

12.4.7. 데이터베이스 링크 정보 조회

Tibero에서는 생성한 데이터베이스 링크의 정보를 제공하기 위해 다음 표에 나열된 정적 뷰를 제공하고 있다.

정적 뷰	설명
DBA_DB_LINKS	Tibero 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 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 종료 또는 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 열려 있는 커서는 없으며 트랜잭션을 수행하고 있다. 동일한 Tibero 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.

제13장 Tibero Standby Cluster

본 장에서는 Tibero Standby Cluster의 구성요소와 동작 및 운영 방법을 설명한다.

13.1. 개요

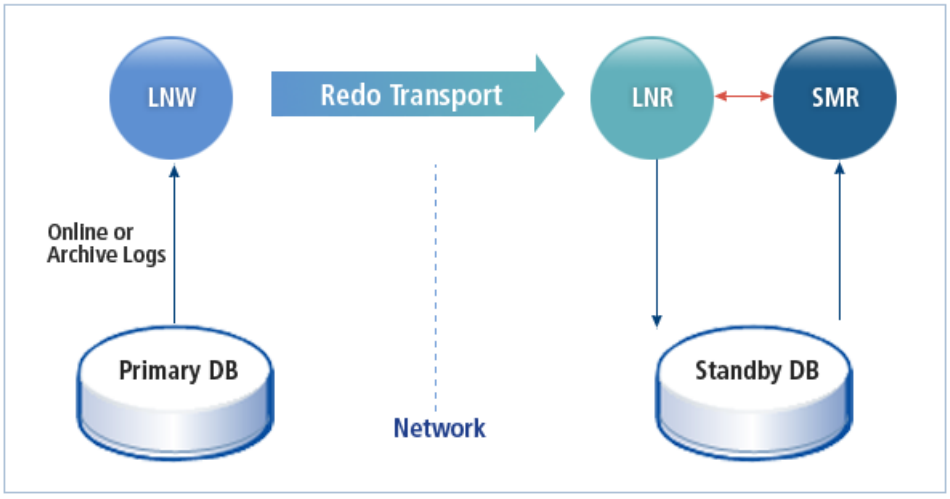
Tibero Standby Cluster는 데이터베이스의 고가용성, 데이터의 보호, 재난 복구 등을 목적으로 제공하는 Tibero의 핵심 기능이다.

Tibero Standby 서버는 물리적으로 독립된 장소에 원본 데이터베이스의 복사본을 트랜잭션 단위로 보관한다. 복사할 대상이 되는 원본 데이터베이스를 **Primary DB**(이하 Primary)라 하고, 복사된 데이터가 저장되는 데이터베이스를 **Standby DB**(이하 Standby)라고 한다.

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 13.1] 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.

13.2. 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 Standby마다 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. LNR은 RCWP 프로세스의 스레드 중 하나로 동작한다.

- SMR(Standby Managed Recovery)

  온라인 Redo 로그를 읽어 Standby에 적용하는 복구 과정을 수행하는 프로세스이다. SMR은 RCWP 프로세스의 스레드 중 하나로 동작한다.

13.3. 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.

13.4. Primary 설정 및 운용

Primary DB의 동작 모드에는 다음과 같다.

구성요소	설명
PROTECTION 모드	적어도 하나 이상의 LGWR SYNC를 포함해야 한다. 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.

구성요소	설명
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.

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 또는 성능을 최대화할지에 대한 전체적인 동작 모드를 설정한다. 한 번만 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. 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.

– 연결 정보 중 포트 번호는 기본적으로 LISTENER_PORT+4 값으로 설정한다. 해당 값은 파라미터 설정을 통하여 변경할 수 있다.

– 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.

- LOG_REPLICATION_N_ENABLE

  – Primary DB down 없이 동적으로 Standby를 추가할 수 있는 파라미터다. 아래와 같은 SQL 문을 이용하여 운영 중 Standby를 추가할 수 있다.

```
sql> alter system set LOG_REPLICATION_MODE={PROTECTION|AVAILABILITY|PERFORMANCE}
sql> alter system set LOG_REPLICATION_DEST_1 = "hostname_1:port_1
{LGWR SYNC|LGWR ASYNC|ARCH ASYNC}"
sql> alter system set LOG_REPLICATION_1_ENABLE=Y; (N으로 설정하면 동기화 모드 비활성화)
```

주의

동적으로 추가하고자 하는 DEST의 Index는 반드시 순서를 지켜야 한다. 예를 들어 DEST_1 설정 없이 DEST_5를 설정하려고 할 경우 DDL이 실패한다.

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.

13.5. Standby 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 13.1] Standby의 $TB_SID.tip 파일의 경로 변환

   ```
   STANDBY_FILE_NAME_CONVERT="Primary의 절대 경로, Standby의 절대 경로"
   ```

   Primary와 Standby의 절대 경로는 각각 Primary와 Standby의 instance 디렉터리의 절대 경로이다.

4. Standby를 MOUNT 모드로 기동한 후 다음의 DDL 문장을 수행하면 해당 DB를 Standby로 설정하고, 변환된 경로가 컨트롤 파일에 적용된다.

   [예 13.2] Standby 컨트롤 파일 설정

   ```
   SQL> ALTER DATABASE Standby controlfile;
   ```

   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 13.3] Standby의 기동

```
$ tbboot -t RECOVERY
```

NORMAL 모드로 Standby를 한 번이라도 기동하게 되면 더 이상 Standby로서의 기능은 할 수 없고, 앞서 설명한 과정을 다시 반복하여 Standby를 설정해야 한다는 점에 주의한다.

Standby가 기동하기 전에 DB 파일이 이전의 버전이라 하더라도 일관성만 유지한다면 동작에는 문제가 없다. 내부적으로 Primary가 접속하여 Primary와 Standby 사이의 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.

13.5.1. 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.

Standby를 read only 클러스터용으로 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 13.4] 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.

Standby를 다시 RECOVERY 모드로 전환시킬 때는 다음의 DDL 문장을 수행한다. 단, 연결 된 세션이 있다면 세션 작업이 끝날 때까지 기다린 후 RECOVERY 모드로 전환된다.

[예 13.5] RECOVERY 모드 전환

```
SQL> ALTER DATABASE Standby;
```

세션 작업이 끝날 때까지 기다리지 않고 바로 RECOVERY 모드로 전환해야 될 경우 다음의 DDL 문장을 수행한다. 다음의 DDL은 연결 된 모든 세션을 강제로 중지시킨 후 RECOVERY 모드로 전환한다.

[예 13.6] RECOVERY 모드 강제 전환

```
SQL> ALTER DATABASE Standby immediate;
```

13.6. TAC-TSC 구성

Primary가 Single이 아니라 TAC일 때도 Standby를 운용할 수 있다.

다음은 TAC-TSC를 구성하는 절차이다.

1. TAC-TSC를 구성하기 위해선 Standby도 TAC 구성이어야 한다. 단, Standby에서 복구는 한 노드에서 진행되므로 Standby 쪽은 1-node TAC로 구성한다. TAC 구성 방법은 "14.6. TAC 구성"을 참고한다.
2. Primary 모든 노드의 $TB_SID.tip 파일에 LOG_REPLICATION_MODE와 LOG_REPLICATION_DEST_N 파라미터를 설정한다. 가급적이면 모든 노드에 대해 동일하게 설정해 주는 것을 권장한다.
3. "13.5. Standby 설정 및 운용"과 마찬가지로 Primary에서 백업한 DB 파일로 Standby를 구성하고, DB_NAME 수정, 파일 경로 변환 등을 수행한 후 기동한다.

TAC-TSC 구성에서 Primary 쪽에 다운되어 있는 노드가 있다면 Standby 노드는 해당 노드의 Redo 로그를 전송받지 못하고, 복구를 정상적으로 수행할 수 없다. Primary 쪽에서 아래의 파라미터를 활성화시켜주면 Primary의 다른 노드가 다운되어있는 노드의 Redo 로그를 대신 전송해주어 Standby 쪽에서 복구가 정상적으로 수행될 수 있다. Primary 노드가 일부 다운된 상황에서도 동기화를 이어가고 싶다면 필수적으로 설정해주어야 한다.

<<$TB_SID.tip>>

```
STANDBY_ENABLE_LOG_RECOVERY=Y
```

이렇게 다운된 노드의 로그를 대신 전송해주는 방식을 LGWR PROXY라고 하며, 위의 파라미터를 적용한 채 Primary를 기동하면 기본적으로 생성되어 있는 LGWR PROXY LNW들 중 자기 자신을 제외한 노드 개수 만큼의 LNW가 할당된다. 한 노드에서 최대 9개의 LNW가 가능하기 때문에 2-node TAC-TSC 구성에서는 8개의 LGWR PROXY LNW가 생성되고, 그 중 하나가 할당된다. Primary 중 한 노드가 다운되면 다른 노드에서는 다운된 노드에 대한 LGWR PROXY LNW의 상태가 CONNECTED로 바뀌고, 다운된 노드의 로그를 대신 전송해준다.

```
SQL> select * from v$standby_dest;
STANDBY_ADDR
--------------------------------------------------------------------------------
TYPE                THREAD# FLAGS                              SENT_SEQ
--------------- ---------- -------------------------------- ----------
SENT_BLKNO  ACKED_SEQ ACKED_BLKNO      DELAY
---------- ---------- ----------- ----------
127.0.0.1:11004
LGWR ASYNC                0 CONNECTED                                11
       894         11         894          0

127.0.0.1:11004
LGWR PROXY                1 CONNECTED                                 7
       880          7         880          0
```

```
Not Assigned:0
LGWR PROXY             65535 NOT CONNECTED                              -1
        -1         -1          -1          0

(중략)

Not Assigned:0
LGWR PROXY             65535 NOT CONNECTED                              -1
        -1         -1          -1          0


9 rows selected.
```

위의 파라미터를 사용할 때 로그 전송 방식이 ASYNC 모드일 경우 다운된 노드의 아카이브 로그를 대신 읽어야 할 경우가 생길 수 있다. 이를 위해선 Primary 각 노드의 LOG_ARCHIVE_DEST가 동일한 위치로 설정되어 있어야 한다. 하지만 일반적인 공유 디스크 환경에서는 이것이 불가능하므로 TAS를 사용해야 한다. TAS-TAC-TSC를 구성하는 방법은 TASCMD 의 cptolocal, cpfromlocal 명령어를 사용하여 Cold Backup을 이용해 구축하는 방법과 tbrmgr의 --for-standby 옵션을 사용해 Hot Backup으로 구축하는 방법이 있다.

참고

STANDBY_ENABLE_LOG_RECOVERY=Y일 경우 '설정된 DEST의 수'에 'TAC의 노드 수'를 곱한 값이 9를 초과하면 안된다.

멀티노드 TSC

TAC-TSC 구성에서 Standby는 TAC이지만 기본적으로 복구를 담당하는 노드 하나만 부팅이 가능하다. 하지만 Standby를 read only 모드로 사용하고자 할 때 한 노드가 복구와 사용자의 쿼리를 동시에 처리해야 하기 때문에 성능의 저하가 생길 수 있다.

멀티노드 TSC 기능은 TAC-TSC 구성에서 최대 Primary의 노드 개수만큼 Standby 노드를 부팅시킬 수 있는 기능이다.

Standby 쪽에 TAC를 구성 후 Standby 모든 노드의 $TB_SID.tip 파일에 아래 파라미터를 설정해주면 멀티노드 TSC 기능이 활성화된다. 멀티노드 TSC를 구성하는 경우 Primary와 마찬가지로 Standby 각 노드에는 고유의 Redo 스레드가 할당되어야 한다.

<<$TB_SID.tip>>

```
STANDBY_ENABLE_TAC_MULTINODE=Y
```

멀티노드 TSC 기능을 사용하면 복구를 담당하는 노드 외에 추가적으로 read only 모드로 동작 가능한 Standby 노드를 부팅시킬 수 있다. 추가로 부팅된 read only 노드를 이용하면 사용자의 쿼리 부하를 분산시킬 수 있어 복구 및 쿼리의 성능이 향상되는 효과를 기대할 수 있다.

Standby에서 복구를 담당하는 노드는 Primary의 $TB_SID.tip 파일에서 설정된 노드이며, 복구는 한 노드가 담당하므로 모든 Primary의 $TB_SID.tip 파일에 동일하게 설정되어 있어야 한다.

Primary를 재기동하지 않고 복구를 담당하는 노드를 바꾸고 싶다면 Primary 모든 노드에서 아래의 sql을 수행해주면 된다. 단, Primary 각 노드에 설정된 LOG_REPLICATION_DEST가 다른 상황은 허용되지 않으므로 모든 노드의 연결 정보를 수정한 후에 동기화를 활성화해야 한다.

```
sql> alter sysetm set LOG_REPLICATION_1_ENABLE=N; (동기화 비활성화)
sql> alter system set LOG_REPLICATION_DEST_1 = "hostname_1:port_1
{LGWR SYNC|LGWR ASYNC|ARCH ASYNC}"; (연결 정보 수정)
sql> alter system set LOG_REPLICATION_1_ENABLE=Y; (바꾼 설정으로 동기화 활성화)
```

참고

멀티노드 TSC 기능을 사용하는 Standby는 RECOVERY 모드로 부팅 시 READONLY 모드로 전환되고, RECOVERY 모드로 변경될 수 없다. 따라서 아래의 DDL은 멀티노드 TSC 기능을 사용하는 Standby에서는 수행이 제한된다.

```
alter database standby;
alter database standby immediate;
```

13.7. 데이터베이스의 역할 전환

본 절에서는 Standby를 Primary로 전환하는 과정을 두 가지 시나리오로 나누어 설명한다.

13.7.1. Switchover

Primary를 Switchover 모드로 종료하고 Standby 중 하나를 Primary로 전환하려면 다음의 절차를 수행한다.

1. Primary에서 다음의 명령어를 입력한다.

   [예 13.7] Switchover 명령어의 실행

   ```
   $ tbdown -t SWITCHOVER
   ```

   Switchover 모드로 종료할 경우 NORMAL 모드와 동일하게 작동을 하며, 추가적으로 Primary에서 생성된 모든 Redo 로그를 Standby에 전송을 한뒤 종료하게 된다.

2. Standby 중 하나를 종료한 후 Failover 모드로 기동하면 그 DB가 새로운 Primary가 된다.

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 Failover 모드로 기동하면 서로 역할을 전환하여 수행할 수 있다. 이때 두 DB는 이미 동기화된 상태이므로 Standby를 구성할 때 필요한 새로

운 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.

13.7.2. Failover

현재 Primary가 갑자기 비정상적으로 종료되었거나 접근이 불가능해진 경우 Standby 중 하나를 Primary로 사용할 수 있다. Standby를 Primary로 사용하기 위해서는 종료 후 다시 Failover 모드로 기동해야 한다. 만약 USE_STANDBY_REDO_LOG 기능을 사용하던 Standby가 Failover 모드 기동을 원할 경우 USE_STANDBY_REDO_LOG 파라미터를 유지한 상태에서 기동해야 한다.

참고

기존에 사용하던 Primary는 새로운 Primary가 운영된 후에는 더 이상 어떤 용도(Primary나 Standby)로도 Tibero Standby Cluster에 포함될 수 없다.

13.8. 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 SID 별로 설정해야 한다.

참고

Primary에서 장애가 발생하면 Standby에 자동으로 접속하는 방법이 있다. 이 방법에 대한 자세한 내용은 “Appendix A. tbdsn.tbr”를 참고한다.

13.9. Tibero Standby Cluster 정보 조회

Tibero에서는 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 로그와 이미 반영된 Redo 로그의 상태를 조회하는 뷰이다. Standby에서만 이 뷰를 사용할 수 있다.
V$STANDBY_LOG	Standby에서 USE_STANDBY_REDO_LOG 기능을 사용할 경우 Standby Redo 로그의 정보를 조회하는 뷰이다.
V$STANDBY_LOGFILE	Standby에서 USE_STANDBY_REDO_LOG 기능을 사용할 경우 Standby Redo 로그 파일의 정보를 조회하는 뷰이다.
V$STANDBY_ARCHIVED_LOG	Standby에서 USE_STANDBY_REDO_LOG 기능을 사용할 경우 생성된 아카이브 로그의 정보를 조회하는 뷰이다.

참고

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

13.10. 제약 사항

Tibero Standby Cluster는 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.

Redo 로그를 적용하는 Standby Cluster의 특성으로 데이터베이스 파일의 상태를 변경하는 DDL 중 일부는 허용하지 않는다. 이와 같은 연산은 Standby 없이 수행해야 한다. 즉, 데이터베이스를 백업하여 Standby에 적용해야 하는 제약이 있다.

다음은 Standby Cluster에서 지원하지 않는 작업들이다. 기존에는 로그가 남지 않는 DPL, DPI에 대해 지원하지 않았지만, 현재 Standby가 연결되어 있을 때 강제로 로깅을 하고 있다.

- 일부 데이터베이스 변경 DDL

```
ALTER DATABASE ADD LOGFILE
ALTER DATABASE DROP LOGFILE
ALTER DATABASE TEMPFILE
ALTER DATABASE DATAFILE OFFLINE
```

- JAVA EXTERNAL PROCEDURE 컴파일을 수행하는 DDL

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE
CREATE OR REPLACE AND COMPILE JAVA SOURCE
```

참고

1. Standby가 Read only 모드일 경우 DB Link 기능은 사용할 수 없다.

2. Standby Cluster에서 암호화 테이블 스페이스를 사용하려면 Primary와 Standby의 보안 지갑 마스터 키가 같아야 한다. 즉, 한 쪽에서 생성된 보안지갑을 복사해서 써야 한다. 또한 Standby에서 보안 지갑이 열려있어야 한다.

13.11. 유용한 추가 기능

TSC 는 다양한 추가 기능들을 제공한다.

13.11.1. Standby Redo Log Group

로그 스위치가 발생하는 경우 재사용할 로그의 checkpoint 여부를 확인하지 않고 아카이빙 여부만 확인하고 덮어쓰는 standby 전용의 새로운 online log group이다.

backup set을 restore한 후 아래와 같은 절차로 이용할 수 있다.

1. 초기화 파라미터 USE_STANDBY_REDO_LOG를 설정한다.

```
USE_STANDBY_REDO_LOG=Y
```

2. Standby Redo Log Group을 구성한다.

```
sql> alter database standby controlfile;
sql> alter database add standby logfile thread 0 group 1
'/usr/tibero/log/stlog001.slf' size 100M;
(size는 primary의 online log file과 일치하게 구성)
sql> alter database add standby logfile thread 0 group 2
'/usr/tibero/log/stlog002.slf' size 100M;
sql> alter database add standby logfile thread 0 group 3
'/usr/tibero/log/stlog003.slf' size 100M;
sql> alter database add standby logfile thread 0 group 4
'/usr/tibero/log/stlog004.slf' size 100M;
sql> alter database add standby logfile thread 0 group 5
'/usr/tibero/log/stlog005.slf' size 100M;
sql> alter database add standby logfile thread 0 group 6
'/usr/tibero/log/stlog006.slf' size 100M;
(개수는 자유롭게 설정 가능하나 Primary의 online로그 개수의 2배 권장)
sql> alter database enable public standby redo thread 0;
sql> alter database recover automatic for standby;
```

3. DB를 다운시킨 후 RECOVERY 모드로 부팅한다.

13.11.2. Snapshot Standby

Snapshot Standby는 Primary와의 동기화는 계속 진행됨과 동시에 독자적으로 ddl/dml을 수행할 수 있는 Standby이다. 사용자는 자신이 원하는 시점에 Standby를 Snapshot Standby로 전환할 수 있으며, Snapshot Standby로 전환된 후 언제든지 다시 Standby로 전환할 수 있다. Snapshot Standby에서는 Redo 로그는 계속 동기화되지만, 받은 로그에 대한 SMR의 리커버리는 중단된다. Standby로 전환할 때 전환 후에 다시 Primary와의 동기화를 이어가기 위해서 Snapshot Standby로 전환된 후에 발생된 모든 변경 사항은 버려지고, DB는 처음 Snapshot Standby로 전환되었던 시점으로 돌아간다.

Standby는 독자적인 ddl/dml이 불가능한 데에 반해, Snapshot Standby는 독자적으로 ddl/dml을 수행할 수 있어 Standby를 테스트 용도로 사용할 수 있고, 그러면서도 Primary와의 동기화는 계속 유지되므로 DR로서의 본연의 역할도 수행할 수 있다.

13.11.2.1. Standby에서 Snapshot Standby로의 전환

Standby에서 Snapshot Standby로 전환하기 위한 절차는 다음과 같다.

1. 초기화 파라미터 FLASHBACK_LOG_BUFFER와 USE_STANDBY_REDO_LOG를 설정한다. (“11.6.3. 플래시백 데이터베이스 실행 예제” 참고)

```
FLASHBACK_LOG_BUFFER=100M
  USE_STANDBY_REDO_LOG=Y
```

2. DB를 MOUNT 모드로 기동한다.
3. Standby Redo Log Group이 구성되어 있지 않다면 구성한다. (“13.11.1. Standby Redo Log Group” 참고)

```
sql> alter database add standby logfile thread 0 group 1
'/usr/tibero/log/stlog001.slf' size 100M;
sql> alter database add standby logfile thread 0 group 2
'/usr/tibero/log/stlog002.slf' size 100M;
sql> alter database add standby logfile thread 0 group 3
'/usr/tibero/log/stlog003.slf' size 100M;
sql> alter database add standby logfile thread 0 group 4
'/usr/tibero/log/stlog004.slf' size 100M;
sql> alter database add standby logfile thread 0 group 5
'/usr/tibero/log/stlog005.slf' size 100M;
sql> alter database add standby logfile thread 0 group 6
'/usr/tibero/log/stlog006.slf' size 100M;
sql> alter database enable public standby redo thread 0;
sql> alter database recover automatic for standby;
```

4. 플래시백 로그가 구성되어 있지 않다면 구성한다. (“11.6.3. 플래시백 데이터베이스 실행 예제” 참고)

```
sql> alter database add flashback logfile thread 0 group 1
'/usr/tibero/log/fblog001.fb' size 512M;
sql> alter database add flashback logfile thread 0 group 1
'/usr/tibero/log/fblog002.fb' size 512M;
sql> alter database add flashback logfile thread 0 group 1
'/usr/tibero/log/fblog003.fb' size 512M;
sql> alter database enable public flashback thread 0;
```

5. Snapshot Standby로의 전환 ddl을 수행한다.

```
sql> alter database convert to snapshot standby;
```

6. DB를 RESETLOGS 모드로 기동한다.

Snapshot Standby에서 수행된 모든 ddl/dml의 변경 사항은 Redo 로그에 저장된다. 따라서 Primary로부터 받은 Redo 로그를 저장하기 위해선 Standby Redo Log Group 기능을 사용해야 한다. Standby Redo Log Group 기능에서는 로그 스위치의 경우 재사용할 로그 파일에 대한 checkpoint를 기다리지 않기 때문에 Primary로부터 계속 Redo 로그를 받아 놓을 수 있다.

Snapshot Standby를 다시 Standby로 전환한 후에는 SMR의 리커버리가 재개되어야 하는데, 이를 위해서는 SMR의 리커버리가 중단되었던 즉 Snapshot Standby로 전환되었던 시점으로 돌아가야 한다. Standby를 과거로 돌리기 위해 Flashback Database 기능이 사용되며, 이를 위해 플래시백 로그를 구성해야 한다. 또한 전환 ddl 수행 시 Snapshot Standby에서 수행될 변경 사항에 대한 플래시백 로깅이 활성화된다.

13.11.2.2. Snapshot Standby에서 Standby로의 전환

Snapshot Standby에서 Standby로 전환하기 위한 절차는 다음과 같다.

1. DB를 MOUNT 모드로 기동한다.

2. Standby로의 전환 ddl을 수행한다.

```
sql> alter database convert to standby;
```

3. 초기화 파라미터 FLASHBACK_LOG_BUFFER를 제거한 후 DB를 RECOVERY 모드로 기동한다.

Standby로 돌아가기 위해 전환 ddl을 입력하면 플래시백 로깅이 비활성화되고 Snapshot Standby로 전환된 시점으로 플래시백이 이루어진다. 그 후 RECOVERY 모드로 부팅하면 SMR은 리커버리가 중단됐던 시점부터 그동안 Primary로부터 받아 놓았던 리두 로그를 리커버리한다. Snapshot Standby 상태였던 기간이 길었다면, Primary로부터 받아 놓았던 Redo 로그의 양이 많기 때문에 Primary의 상태를 따라 잡는데 오래 걸릴 수 있다.

참고

Snapshot Standby 상태에서는 Failover가 불가능하다.

13.11.3. Cascading Standby

Primary에 여러 Standby 노드를 연결하면 Primary에게 부담이 될 수 있다. Cascading Standby를 이용하면 Primary가 모든 Standby 노드에게 redo 로그를 전송하지 않고, Primary 노드는 Cascading Standby 노드에만 Redo 로그를 전송, Cascading Standby 노드가 다른 Cascaded Standby 노드에게 Redo 로그를 전송하도록 해 Primary의 부담을 줄여준다.

Cascading Standby 기능을 사용하기 위해 STANDBY_ENABLE_CASCADING 파라미터를 설정한다. 그리고 Primary의 $TB_SID.tip 파일 설정과 동일하게 Cascading Standby 노드의 $TB_SID.tip 파일을 설정한다. Cascading Standby는 LOG_REPLICATION_MODE 중 AVAILABILTY와 PERFORMANCE만 지원하며 LOG_REPLICATION_DEST_N 중 LGWR ASYNC와 ARCH ASYNC만 지원한다.

<<$TB_SID.tip>>

```
STANDBY_ENABLE_CASCADING = Y
LOG_REPLICATION_MODE   = {AVAILABILITY|PERFORMANCE}
LOG_REPLICATION_DEST_1 = "hostname_1:port_1 {LGWR ASYNC|ARCH ASYNC}"
LOG_REPLICATION_DEST_2 = "hostname_2:port_2 {LGWR ASYNC|ARCH ASYNC}"
...
LOG_REPLICATION_DEST_N = "hostname_N:port_N {LGWR ASYNC|ARCH ASYNC}"
```

주의

Cascading Standby 노드와 연결될 Cascaded Standby 노드를 구축할 때는 반드시 Cascading Standby 노드 구축 백업과 동일한 백업 혹은 이전의 백업을 이용해야 정합성에 문제가 없다.

제14장 Tibero Cluster Manager

본 장에서는 Tibero Cluster Manager의 기본 개념과 환경설정 및 실행 방법을 설명한다.

14.1. 개요

Tibero Cluster Manager(이하 CM)는 클러스터의 가용성을 높이고 관리의 편의를 지원한다. 또한, Tibero Active Cluster 서비스를 위한 인스턴스간 멤버십 관리를 지원한다. CM에서는 클러스터의 각종 구성 요소들을 리소스라는 개념으로 관리하도록 되어있다.

다음은 현재 CM에서 관리 가능한 클러스터 리소스이다.

- File
- Network
- Cluster
- Service
- DB(Database)
- AS(Active Storage)
- VIP

CM은 등록된 리소스들에 대한 모니터링을 수행하며, 상태 변화에 따른 필요 동작들을 수행한다. 설정을 통해 같은 클러스터에 속하게 된 CM들 간에는 지정된 네트워크와 공유 디스크를 통해 주기적으로 자신이 살아있음을 알리는 Heartbeat를 서로 주고받으며 클러스터의 멤버십 구성을 파악하게 된다.

클러스터 구성 CM 중 하나의 CM이 마스터의 역할을 맡아 비정상적인 상황이 발생하는 경우 클러스터 멤버십을 자동으로 변경하여, 해당 클러스터 상의 서비스가 중단되지 않도록 한다. 마스터에 문제가 생긴 경우 클러스터에 속한 다른 노드가 마스터 역할을 넘겨 받는다.

14.2. 환경변수 및 초기화 파라미터

14.2.1. 환경변수

환경변수 CM_HOME과 CM_SID를 설정해야 한다. 다음은 그 예제이다. 이 외에 RDBMS를 실행하기 위해 필요한 기본 환경변수들은 “제2장 관리의 기본”을 참고하여 설정한다.

환경변수	설명
$CM_HOME	CM이 설치된 경로를 지정한다. – 예) `export CM_HOME=/home/tibero7`
$CM_SID	노드를 식별할 수 있는 ID를 지정한다. 각 노드는 서로 다른 값을 가져야 한다. – 예) `export CM_SID=cm0`

14.2.2. 초기화 파라미터

CM용 TIP에는 다음과 같은 파라미터들을 설정할 수 있다. 필수 항목은 사용자가 TIP에 명시적으로 값을 지정해야 하며, 선택 항목은 사용자가 지정하지 않을 수 있고 이 경우 기본값으로 입력된다.

초기화 파라미터	필수 여부	설명
CM_NAME	필수	클러스터를 생성할 때에 node identifier로 쓸 이름이다. 서로 다른 노드의 CM은 다른 값을 가져야 한다.
CM_UI_PORT	필수	cmrctl 명령어 수행할 때에 CM으로 접속하는 용도로 사용할 네트워크 포트 번호이다. 이 포트는 노드 간에 열려 있어야 한다.
CM_RESOURCE_FILE	필수	CM 리소스 파일의 경로를 지정한다. CM 리소스 파일은 다음과 같은 상황에 사용된다. – 동작 중인 CM에 등록된 리소스들의 내용을 저장해두기 위해 사용하는 파일 – 리소스의 추가, 변경 혹은 삭제할 때 리소스 파일에 자동으로 동기화 – 부팅할 때 해당 파일을 읽어 이전 동작 상황에서 관리하던 리소스들을 자동으로 추가
CM_RESOURCE_FILE_BACKUP	선택	CM 리소스 파일의 백업 경로를 지정한다.
CM_RESOURCE_FILE_BACKUP_INTERVAL	선택	CM_RESOURCE_FILE_BACKUP 경로에 CM 리소스 파일의 백업을 수행할 시간 간격을 나타낸다. (단위: 분)

초기화 파라미터	필수 여부	설명
LOG_LVL_CM	선택	CM이 로그를 남기는 수준을 지정한다. 값이 높을수록 CM이 더 많은 로그를 저장하며, 1~6 사이의 정수값을 가질 수 있다. (기본값: 2)
CM_LOG_DEST	선택	CM이 로그를 남길 디렉터리를 지정한다. 이 값은 반드시 절대 경로여야 한다. (기본값: $CM_HOME/instance/$CM_SID/log/)
CM_GUARD_LOG_DEST	선택	CM Guard가 로그 파일을 저장할 장소로 이 값은 반드시 로컬 디스크의 절대 경로여야 한다. 또한, CM이 root로 실행되어야만 CM Guard가 로그를 남길 수 있다. 다음은 각 플랫폼별 기본값이다. – HP OS : /var/adm/syslog/cm_guard/$CM_SID/ – Windows와 HP 제외 다른 OS : /var/log/cm_guard/$CM_SID/ (Windows는 Guard가 뜨지 않는다.)
CM_LOG_FILE_SIZE	선택	CM 로그 파일의 최대 크기(용량)를 지정한다. 로그가 계속 생성되어 파일의 크기가 이 값을 넘어서려 할 경우 현재의 로그 파일이 닫히고, 새로운 파일이 생성되어 그 파일에 로그가 저장된다. 값은 100KB~1GB 사이의 정수값을 가질 수 있다. (단위: Byte, 기본값: 10MB)
CM_LOG_TOTAL_SIZE_LIMIT	선택	CM_LOG_DEST에서 지정한 디렉터리에 생성되는 로그 파일들의 크기 총합의 최댓값을 지정한다. 로그 파일들의 크기의 합이 이 값을 넘어가게 되면, 가장 오래된 파일을 삭제함으로써 파일 용량이 끝없이 증가하는 것을 막는다. 이 값은 100KB~1GB 사이의 정수값을 가질 수 있다. (단위: byte, 기본값: 300MB)
CM_TIME_UNIT	선택	CM을 컨트롤할 때 사용되는 단위 시간의 길이를 지정한다. (단위: 0.1초, 기본값: 10)
CM_HEARTBEAT_EXPIRE	선택	Heartbeat expire란 다른 노드의 CM에 문제가 생겼다는 것을 CM이 알아차리는데 필요한 표준 시간을 의미한다. 즉, 현재 노드의 CM에서 Heartbeat expire 시간이 지나도록 통신이 되지 않는 CM이 있으면, 현재 노드의 CM은 통신이 되지 않는 노드의 CM에 문제가 생겼다고 인식하게 된다. (단위: 초(second), 기본값: 300)

초기화 파라미터	필수 여부	설명
CM_NET_EXPIRE_MARGIN	선택	CM을 컨트롤하기 위한 네트워크의 Heartbeat expire 시간이다. 이 값은 5보다 크거나 같아야 한다. (단위: 초(second), 기본값: 5)
CM_WATCHDOG_EXPIRE	선택	RDBMS와 CM 사이에 watchdog 기능이 활성화되어 있을 경우 만료 기간(expiration period)을 지정한다. 만약 CM이 이 값이 지정한 시간 동안 동작하지 않으면, RDBMS가 자동적으로 종료된다. CM_HEARTBEAT_EXPIRE보다 작은 값을 사용해야 한다. (단위: 초(second), 기본값: 290)
CM_FENCE	선택	CM fence 데몬을 실행시킬지를 결정한다. CM이 I/O 수행을 CM_WATCHDOG_EXPIRE에 지정된 시간보다 오래할 경우, 다른 CM들이 이 CM의 노드를 클러스터에서 제외시키기 때문에 이 CM의 노드는 OS를 재부팅해야 한다(그 노드의 RDBMS가 I/O하는 것을 막기 위해). CM fence 데몬은 이러한 일을 수행하며, 재부팅 권한이 필요하므로 이 값을 Y로 지정하려면 CM이 ROOT 권한으로 실행되어야 한다. (기본값: N)
CM_ENABLE_FAST_NET_ERROR_DETECTION	선택	다른 CM들과 연결된 네트워크의 장애를 감지함으로써 다른 CM의 상태 변화를 빨리 알아차릴 수 있도록 하는 기능을 활성화시킬지를 결정한다. (기본값: N)
_CM_BLOCK_SIZE	선택	CM 파일의 I/O 단위 크기로 Byte 단위이다. 대부분의 운영체제는 기본값을 사용하면 되지만, HP-UX는 1024를 사용해야 한다. (기본값: 512)

14.3. CM 실행

CM 데몬 실행 이후의 작업들은 cmrctl 명령어를 사용하여 수행한다.

CM은 다음의 방법으로 실행한다.

```
tbcm [option] <argument>
```

- [option]

항목	설명
-b	CM 데몬을 실행한다.

항목	설명
-d	CM 데몬을 종료한다.
-i <*file path*>	지정된 경로의 text 파일에서 리소스 정보를 import한다.
-e <*file path*>	현재 CM에 추가된 리소스 정보를 지정된 경로에 text 파일로 export한다.
-x <*file path*>	현재 CM에 추가된 리소스 정보를 지정된 경로에 파일로 export한다.
-X	현재 CM에 추가된 리소스 정보를 CM_RESOURCE_FILE로 지정해 놓은 경로에 export한다.
-s	CM의 상태를 보여준다. 초기화 파라미터에 등록된 내용이 표시된다.
-v	CM의 버전을 보여준다.
-h	tbcm 명령어에 대한 도움말을 보여준다.

14.4. CM 명령어

본 절에서는 CM 명령어인 cmrctl, crfconf에 대해서 설명한다.

14.4.1. cmrctl 명령어

cmrctl은 New CM에서 리소스들을 관리 및 제어하기 위해 사용하는 명령어 set이다.

기본적인 cmrctl 명령어 용법은 다음과 같다.

```
cmrctl <action> <resource_type> [--<attr_key> <attr_val>|...]
```

항목	설명
action	add, del, show, start, stop, act, deact, modify, relocate
resource_type	network, cluster, service, db, as, vip, file

참고

어떤 action, resource type의 조합은 사용이 불가능할 수도 있다. (예: add, file)

cmrctl 명령어를 통한 CM 원격 제어

같은 클러스터를 구성 중인 CM에 대해서 cmrctl 명령에 추가 attribute를 명시함으로써 원격 제어가 가능하다. 원격 제어를 할 CM 노드의 이름과 해당 노드가 속한 클러스터의 이름을 알고 있어야 하며, 다음과 같은 attribute를 cmrctl 명령에 추가한다.

```
--remote <node_name>@<cluster_name>
```

다음의 예제 구문처럼 해당 attribute는 cmrctl add와 cmrctl del 명령에는 사용할 수 없다.

```
# cls1 클러스터에 속한 cm2 노드의 리소스 조회
$ cmrctl show --remote cm2@cls1

# cls1 클러스터에 속한 cm1 노드의 tibero1 db down
$ cmrctl stop db --name tibero1 --remote cm1@cls1

# 원격으로 resource add 시 error 발생
$ cmrctl add db --name tac1 ... --remote cm1@cls1
[ERROR] Cannot add (or delete) resource remotely
```

지정한 클러스터가 down 상태이거나 지정한 노드가 down 상태일 경우 또는 잘못된 클러스터/노드 이름을 입력했을 경우에는 에러 메시지를 출력한다.

14.4.1.1. cmrctl add

cmrctl add는 다음의 명령어로 구성된다.

명령어	설명
cmrctl add network	네트워크 리소스를 추가하기 위한 명령어이다.
cmrctl add cluster	클러스터 리소스를 추가하기 위한 명령어이다.
cmrctl add service	서비스 리소스를 추가하기 위한 명령어이다.
cmrctl add db	Tibero 인스턴스를 추가하는 명령어이다.
cmrctl add as	AS(Active Storage) 인스턴스를 추가하는 명령어이다.
cmrctl add vip	VIP를 등록하기 위한 명령어이다.

cmrctl add network

네트워크 리소스를 추가하기 위한 명령어이다. 네트워크가 public/private 용도에 따라서 필요한 attribute가 나뉜다. 네트워크 리소스는 interconnect로 사용할 IP, PORT 조합이나 VIP 등록을 위해 필요한 public 네트워크 인터페이스 등록을 위해 사용하는 자원이다. 등록된 이후에 지정된 네트워크 인터페이스를 감시하여 자동으로 상태를 갱신한다.

```
cmrctl add network --name <network_name> --nettype <private|public>
--ipaddr <network_ipaddr/netmask_addr> --portno <port_no> --ifname <interface_name>
```

Key	Value Type	설명
name	string	네트워크 리소스 이름이다. (unique, mandatory)
nettype	string	네트워크 리소스의 type이다. – private : interconnect를 의미한다. (기본값) – public : VIP 용으로 사용한다.

Key	Value Type	설명
ipaddr	string	interconnect로 사용할 IP 주소이다. 이 포트는 노드 간에 열려 있어야 한다. (only for nettype 'private'. mandatory)
portno	integer	CM 간의 interconnect로 사용할 포트 번호이다. (only for nettype 'private', mandatory)
ifname	string	public 용도로 사용할 interface name(VIP 등록용)이다. (only for nettype 'public', mandatory)

cmrctl add cluster

클러스터 리소스를 추가하기 위한 명령어이다. 클러스터 리소스는 환경의 개념으로 노드 간 연결에 사용하는 interconnect, 노드 간 공유하는 스토리지, VIP를 사용하는 경우에 필요한 public network interface로 구성된다.

```
cmrctl add cluster --name <cluster_name> --incnet <network_resource_name>
--pubnet <public_network_resource_name> --cfile <file_path>
```

Key	Value Type	설명
name	string	클러스터 리소스 이름이다. (unique, mandatory)
incnet	string	interconnect 용도로 사용할 네트워크 리소스 이름이다. (mandatory)
pubnet	string	public 용도로 사용할 네트워크 리소스 이름이다. VIP를 사용하는 경우 등록한다.

cfile	file path	클러스터 파일 경로이다. 콤마(,)로 구분하여 여러 개 등록할 수 있다. (mandatory)
		
		cfile attribute의 경우 가장 앞에 '+'를 붙이면 TAS용 경로(diskstring)로 인식한다.
		
		– 예제 1)
		
		`--cfile "+/my/dev/disk1,/my/dev/disk2,/my/dev/disk3"`
		
		– 예제 2)
		
		`--cfile "+/my/dev/disk*"`
		
		스토리지 서버를 사용할 경우에는 다음과 같이 지정한다.
		
		`--cfile "-"`
		
		[참고]

Key	Value Type	설명
		1. cfile을 TAS상의 경로로 설정할 경우 TAS diskstring과 동일하게 설정해야 하며, TAS diskstring이 아닌 raw device나 파일로 지정할 경우 홀수 개로 지정하는 것을 권장한다(과반 법칙). 2. 파일 리소스의 경우 사용자가 수동으로 추가하는 것이 아니라, 클러스터 리소스의 --cfile로 등록된 내용으로 자동 생성된다. TAS diskstring을 등록한 경우 +0, +1, +2와 같은 형태로 리소스가 생성된다.

cmrctl add service

서비스 리소스를 추가하기 위한 명령어이다. 서비스는 클러스터라는 환경 위에서 실제 어떤 서비스를 제공하는 인스턴스의 집합을 관리하기 위한 개념이다.

```
cmrctl add service --name <service_name> --type <DB|AS> --mode <AC|HA>
--cname <cluster_resource_name>
```

Key	Value Type	설명
name	string	서비스 리소스 이름이다. (unique, mandatory) 서비스 리소스 이름은 해당 서비스와 매핑되는 DB의 DB_NAME 파라미터와 같아야 한다(TAS는 TIP 파일에 DB_NAME을 안적어도 되지만, 적는다면 서비스의 이름과 같아야 한다).
type	string	서비스 타입이다. – DB : Tibero 인스턴스가 동작하는 서비스이다. (기본값) – AS : TAS 인스턴스가 동작하는 서비스이다.
mode	string	서비스 인스턴스 클러스터링 모드이다. – AC : 한 서비스에 대한 인스턴스가 여러 노드에서 동시에 실행되어 동작 가능한 모드이다. (기본값) – HA : 한 서비스에 대한 인스턴스가 하나만 동작 가능한 모드이다.
cname	string	해당 서비스 리소스가 속할 클러스터 리소스 이름이다. (mandatory)

AS 서비스의 경우에는 한 클러스터 당 하나만 등록 가능하다. 서비스 리소스는 특정 클러스터를 지정하여 추가하도록 되어 있는데, 한 노드에 추가하면 해당 노드가 속한 클러스터의 모든 노드에 자동으로 공유된다.

cmrctl add db

Tibero 인스턴스를 추가하는 명령어이다. DB type의 서비스에만 추가 가능하다.

```
cmrctl add db --name <db_resource_name> --svcname <service_name>
--dbhome <directory_path> --envfile <file_path> --retry_cnt<retry_cnt>
--retry_interval<retry_interval>
```

Key	Value Type	설명
name	string	DB 리소스 이름이다. DB 리소스의 name은 해당 DB 인스턴스의 TB_SID와 동일해야 한다. (unique, mandatory)
svcname	string	DB 리소스가 속할 서비스 리소스 이름이다. (mandatory)
dbhome	string(directory path)	기존의 TB_HOME 개념으로 DB 바이너리가 위치한 경로이다. (mandatory)
envfile	string(file path)	DB 바이너리를 실행하기 위한 환경설정용 파일이다. (recommended) **[참고]** envfile은 DB 리소스별로 지정하는 것을 권장한다. envfile에는 RDBMS를 부팅하고 종료하기 위해 필요한 환경변수들을 export하는 내용이 들어간다.
retry_cnt	integer	최대 retry 시도 횟수이다. (기본값: 3)
retry_interval	integer	retry 시도 간의 간격이다. (단위: 초, 기본값: 0(retry 기능 사용하지 않음))

다음은 envfile을 입력하지 않았을 때 default로 수행되는 내용이다. 이 외의 환경변수들은 CM이 부팅할 때 터미널이 가지고 있던 환경변수가 그대로 적용된다. LD_LIBRARY_PATH는 Linux 또는 SunOS를 사용할 경우이고, AIX를 사용할 경우 LIBPATH, HP-UX는 SHLIB_PATH를 export한다. 환경변수 설정에 대한 자세한 내용은 "Tibero 설치 안내서"를 참고한다.

```
export TB_SID=name                                    #db 리소스의 name
export TB_HOME=dbhome                                 #db 리소스의 dbhome
export PATH=$TB_HOME/bin:$TB_HOME/client/bin:/usr/bin:$PATH
export LD_LIBRARY_PATH=$TB_HOME/lib:$TB_HOME/client/lib:$LD_LIBRARY_PATH
```

cmrctl add as

AS(Active Storage) 인스턴스를 추가하는 명령어이다. AS type의 서비스에만 추가 가능하다.

```
cmrctl add as --name <as_resource_name> --svcname <service_name>
--dbhome <directory_path> --envfile <file_path> --retry_cnt<retry_cnt>
--retry_interval<retry_interval>
```

Key	Value Type	설명
name	string	AS 리소스 이름이다. (unique, mandatory)
svcname	string	AS 리소스가 속할 서비스 리소스 이름이다. (mandatory)
dbhome	string(directory path)	기존의 TB_HOME 개념으로 AS 바이너리가 위치한 경로이다. (mandatory)
envfile	string(file path)	AS 바이너리를 실행하기 위한 환경설정용 파일이다. (recommended)
retry_cnt	integer	최대 retry 시도 횟수이다. (기본값: 3)
retry_interval	integer	retry 시도 간격이다. (단위: 초, 기본값: 0(retry 기능 사용하지 않음))

참고

AS 리소스의 name은 해당 AS 인스턴스의 TB_SID와 동일해야 한다. envfile에 관한 설명은 "cmrctl add db"를 참고한다.

cmrctl add vip

VIP를 등록하기 위해서는 tbcm이 ROOT 권한으로 실행되어 있어야 하며, svcname으로 지정한 서비스에 pubnet attribute가 등록되어 있어야 한다. 환경변수 PATH에 /sbin 경로가 잡혀있지 않으면 VIP alias에 실패하므로 확인해야 한다.

- VIP failover/failback

  클러스터의 특정 노드에서 장애가 발생하면 사용 중인 VIP를 정상적으로 작동하는 노드 중 한 노드에 VIP를 옮겨 설정한다(failover). 이를 통해서 Tibero는 장애가 발생한 노드의 VIP로도 계속해서 서비스를 할 수 있다. 장애가 발생했던 노드가 복구되면 다시 원래 소유했던 노드로 VIP를 옮겨 설정한다(failback).

```
cmrctl add vip --name <vip_name> --node <CM_SID> --svcname <service_name>
--ipaddr <vip_ipaddr/netmask_addr> --bcast <bcast_addr>
```

Key	Value Type	설명
name	string	VIP 리소스 이름이다. (unique, mandatory)

Key	Value Type	설명
node	string	VIP를 원래 소유했던 CM_SID 이다. (optional, 기본값: 해당 명령어를 수행한 노드의 CM_SID)
svcname	string	VIP를 사용할 서비스 이름이다. (mandatory)
ipaddr	string(IP address/Netmask)	VIP IP address/Netmask로 조합된 주소이다. - IP address (mandatory) - Netmask (optional, 기본값: public interface의 netmask)
bcast	Broadcast	VIP broadcast 주소이다. (optional)

참고

VIP로 접속할 때 사용해야 할 포트는 DB인스턴스에서 LISTENER_VIP_PORT로 설정이 가능하다. 설정한 포트는 VIP failback 과정에서 세션을 정리해주기 위해 잠시 막힌다. LISTENER_VIP_PORT와 LISTENER_PORT를 다르게 설정한다면 VIP failback 과정 중에도 LISTENER_PORT로의 접속은 가능하다.

14.4.1.2. cmrctl del (delete)

특정 리소스를 삭제하기 위한 명령어이며, DOWN 또는 DEACT 상태의 리소스만 삭제 가능하다.

```
cmrctl del <resource_type> --name <resource_name>
```

14.4.1.3. cmrctl show

CM에 등록된 리소스의 정보를 확인하기 위한 명령어이다.

다음의 3가지 방법으로 사용이 가능하다.

- 방법 1)

  ```
  cmrctl show
  ```

  현재 CM 데몬에 등록된 리소스의 목록을 출력한다.

  ```
  cmrctl show all
  ```

  현재 CM 데몬의 클러스터에 등록된 모든 노드의 리소스의 목록을 출력한다.

- 방법 2)

  ```
  cmrctl show <resource_type>
  ```

현재 CM 데몬에 등록된 리소스들 중 <resource_type>의 리소스 목록을 출력한다.

- 방법 3)

```
cmrctl show <resource_type> --name <resource_name>
```

지정한 특정 리소스의 상세한 정보를 출력한다.

14.4.1.4. cmrctl start

특정 리소스를 시작하기 위한 명령어이다. 서비스 리소스를 start하는 경우 해당 서비스에 속한 모든 인스턴스를 기동시킨다.

```
cmrctl start <resource_type> --name <resource_name> [--option <options>]
```

14.4.1.5. cmrctl stop

특정 리소스를 중지하기 위한 명령어이다. 서비스 리소스를 stop하는 경우 해당 서비스에 속한 모든 인스턴스를 중지시킨다. 또한, auto-restart 기능이 동작 중이라면 중지된다.

auto-restart 모드가 동작 중에 서비스의 인스턴스가 중지된 경우 해당 상황을 인지한 후에 중지된 인스턴스를 재시작시켜주는 기능이다.

```
cmrctl stop <resource_type> --name <resource_name> [--option <options>]
```

14.4.1.6. cmrctl act

다음의 이유로 인해 deactivate된 리소스를 다시 activate시켜주기 위한 명령어이다.

- DB 또는 AS 리소스를 추가할 때 입력한 retry_cnt(기본값: 3) 이상 start를 수행했는데도 실패한 경우
- 사용자가 cmrctl deact 명령어를 명시적으로 사용하여 비활성화시킨 경우

서비스 리소스를 act하는 경우에는 해당 서비스 리소스의 인스턴스들에 대한 auto-restart 기능을 동작시키겠다는 의미로 사용된다.

```
cmrctl act <resource_type> --name <resource_name>
```

14.4.1.7. cmrctl deact

특정 리소스를 deactivate시켜주기 위한 명령어이며, deactivate된 리소스는 auto-restart 대상에서 제외된다. 서비스 리소스를 deact하는 경우에는 해당 서비스 리소스의 인스턴스들에 대한 auto-restart 기능을 중지시키겠다는 의미로 사용된다.

```
cmrctl deact <resource_type> --name <resource_name>
```

14.4.1.8. cmrctl modify

인스턴트 리소스의 retry_cnt, retry_interval을 변경시켜 주기 위한 명령어이다.

```
cmrctl modify <resource_type> --name <resource_name> [--option <options>]
```

14.4.1.9. cmrctl relocate

CM 멀티노드 환경에서 vip의 소유권 이전/변경하기 위한 명령어이다.

```
cmrctl relocate vip --name <vip_name> --nid <target_nid>
```

target_nid는 vip 소유권을 가질 노드로서 cmrctl show cluster --name [cluster_name] 명령어를 통해 노드별 nid 및 현재 vip를 소유하는 nid를 확인할 수 있다.

14.4.2. crfconf 명령어

cmrctl 명령어의 경우 CM이 온라인 상태일 때 CM에 접속하여 사용자가 리소스를 관리할 수 있는 명령어라면, crfconf는 CM이 오프라인 상태일 때 CM TIP에 지정된 경로에 있는 CM 리소스 파일에 접근하여 해당 파일을 관리할 수 있도록 하는 유틸리티이다. CM 리소스 파일의 경우 바이너리 파일이기 때문에 사용자가 임의로 파일을 수정할 수 없다. 따라서, 사용자가 CM을 부팅하기 전에 CM 리소스 파일에 있는 리소스 정보를 변경하려면 **crfconf**를 사용하여 원하는 작업을 수행할 수 있다. 이때 VIP 등과 같은 글로벌 리소스의 변경을 시도하는 경우에는 마스터가 관리하기 때문에 마스터로 기동할 CM의 CM 리소스 파일을 변경해야 정상적으로 적용된다.

crfconf의 사용법은 cmrctl과 동일하며, 사용 가능한 action이 add, del, show의 3가지로 제한된다는 점만 다르다.

```
crfconf <action> <resource_type> [--<attr_key> <attr_val>|...]
```

CM이 리소스 파일에 접근할 수 있는 CM 동작 상황에서는 crfconf를 수행할 때 다음과 같은 에러를 출력하며 실패로 처리된다.

```
$ crfconf show
[ERROR] CM is online. use 'cmrctl' command
crfconf failed!
```

14.5. Cluster Resource의 ROOT 모드

CM이 VIP 등과 같은 글로벌 리소스를 관리할 때 클러스터의 구성원이 모두 ROOT 권한이어야 하는 일들이 발생한다. 따라서 클러스터 리소스에는 ROOT 모드라는 것이 존재하는 데, 어떤 클러스터에 포함된 모든 CM들이 ROOT 권한으로 실행되었을 때 그 클러스터는 ROOT 모드가 된다. ROOT 모드의 클러스터는 ROOT 모드를 유지하기 위해 ROOT 권한이 없는 CM이 접속하는 것을 막는다.

다음은 클러스터가 ROOT 모드가 되는 두 가지 예이다.

- **클러스터에 처음 접속한 CM이 ROOT 권한으로 띄워진 경우**

  이 경우 해당 클러스터는 시작부터 ROOT 모드가 되므로, 이 이후로 클러스터에 접속하려는 CM들은 모두 ROOT 권한으로 실행되어야만 접속이 가능하다.

- **클러스터 내의 ROOT 권한으로 실행되지 않은 노드들이 모두 down된 경우**

  5개의 노드로 구성된 클러스터를 생각해보자. 1~2번 노드는 ROOT 권한 없이 실행된 CM이고, 3~5번 노드는 ROOT 권한으로 실행된 CM이라면 이 순간 해당 클러스터는 ROOT 모드가 아니다. 이때 1번과 2번노드를 다운시키면, 클러스터에 접속해있는 CM은 3개가 되고, 3개 모두 ROOT 권한을 가진 CM이므로 클러스터가 ROOT 모드가 된다. 이 후로는 클러스터가 ROOT 모드이기 때문에 1번과 2번 노드가 다시 클러스터에 접속하기 위해서는 ROOT 모드로 CM을 실행시켜야만 한다.

ROOT 모드 클러스터에는 ROOT 권한이 없는 CM이 접속할 수 없으므로, ROOT 모드 클러스터를 ROOT 모드가 아닌 클러스터로 만들려면 모든 노드에서 클러스터를 down시킨 후 ROOT 권한이 없는 CM이 처음 클러스터에 접속하도록 해야 한다. 단, ROOT 모드가 아닌 클러스터에는 ROOT 권한 유무에 상관 없이 CM이 접속할 수 있지만, ROOT 권한을 가진 CM이라 해도 VIP 사용이 불가능하다는 점을 주의할 필요가 있다.

어떤 CM이 ROOT 권한을 가지고 실행되었는지와 클러스터가 ROOT 모드인지는 다음의 방법으로 확인할 수 있다.

```
cmrctl show cluster --name 'cluster name'
```

다음은 2-node-cluster에서 두 노드 모두 ROOT 권한이 있는 CM으로 접속한 상황이다. Status의 UP 옆에 (ROOT)라는 표시는 cluster가 ROOT 모드임을 의미하며, NODE LIST에 Mst 아래 R은 각 노드 CM의 ROOT 권한 소유를 의미한다.

```
cmrctl show cluster --name cls1

Cluster Resource Info
================================================================
Cluster name      : cls1
Status            : UP        (ROOT)
Master node       : (1) cm0
Last NID          : 2
Local node        : (2) cm1
Storage type      : Active Storage
As Diskstring     : /data/*
No. of cls files  : 3
  (1) +0
  (2) +1
  (3) +2
================================================================
|                          NODE LIST                           |
|--------------------------------------------------------------|
| NID   Name         IP/PORT        Status Schd Mst  FHB  NHB |
| --- -------- ------------------- ------ ---- --- ---- ---- |
|   1      cm0      123.1.1.1/29000     UP    Y R M   30   35 |
```

```
|   2      cm1      124.1.1.1/29100     UP    Y R   [ LOCAL ] |
================================================================
```

다음은 두 노드 모두 ROOT 권한이 없는 CM으로 클러스터에 접속한 상황이다. 앞에서와 달리 Status에 (ROOT)가 없고, NODE LIST에도 Mst에 R을 가진 노드가 없다.

```
cmrctl show cluster --name cls1

Cluster Resource Info
================================================================
Cluster name      : cls1
Status            : UP
Master node       : (1) cm0
Last NID          : 2
Local node        : (2) cm1
Storage type      : Active Storage
As Diskstring     : /data/*
No. of cls files  : 3
  (1) +0
  (2) +1
  (3) +2
================================================================
|                          NODE LIST                           |
|--------------------------------------------------------------|
| NID   Name         IP/PORT        Status Schd Mst  FHB  NHB |
| --- -------- -------------------- ------ ---- --- ---- ---- |
|   1      cm0      123.1.1.1/29000     UP    Y   M   30   35 |
|   2      cm1      124.1.1.1/29100     UP    Y     [ LOCAL ] |
================================================================
```

다음은 한 노드가 ROOT 권한이 없는 CM으로 클러스터에 접속하고, 다른 노드는 ROOT 권한을 가진 CM으로 접속한 상황이다. 노드 2번만 Mst에 R이 있고, 클러스터가 ROOT 모드가 아니므로 Cluster Resource Info의 Status에 (ROOT)가 없다.

```
cmrctl show cluster --name cls1

Cluster Resource Info
================================================================
Cluster name      : cls1
Status            : UP
Master node       : (1) cm0
Last NID          : 2
Local node        : (2) cm1
Storage type      : Active Storage
As Diskstring     : /data/*
```

```
No. of cls files  : 3
  (1) +0
  (2) +1
  (3) +2
================================================================
|                          NODE LIST                           |
|--------------------------------------------------------------|
| NID   Name         IP/PORT        Status Schd Mst  FHB  NHB |
| --- -------- -------------------- ------ ---- --- ---- ---- |
|   1      cm0      123.1.1.1/29000     UP    Y   M   30   35 |
|   2      cm1      124.1.1.1/29100     UP    Y R   [ LOCAL ] |
================================================================
```

마지막으로 바로 위의 상황에서 루트 권한이 없는 1번 노드의 클러스터를 stop시킨 후 다시 cluster start를 하려는 상황이다. 1번 노드에서 클러스터가 stop됨으로 인해 클러스터에 ROOT 권한 노드만 남게 되고, 이에 따라 클러스터가 ROOT 모드가 됨을 확인할 수 있다. 또한 클러스터가 ROOT 모드이므로, ROOT 권한이 없는 1번 노드 CM은 클러스터에 재접속할 수 없게 된다.

– 1번 노드

```
cmrctl stop cluster --name cls1
```

```
cmrctl start cluster --name cls1

Failed to start the resource 'cls1'
 [ERROR] To join this cluster(cls1), you must be root
```

– 2번 노드

```
cmrctl show cluster --name cls1

Cluster Resource Info
================================================================
Cluster name      : cls1
Status            : UP        (ROOT)
Master node       : (1) cm1
Last NID          : 2
Local node        : (2) cm1
Storage type      : Active Storage
As Diskstring     : /data/*
No. of cls files  : 3
  (1) +0
  (2) +1
  (3) +2
================================================================
|                          NODE LIST                           |
|--------------------------------------------------------------|
```

```
| NID   Name         IP/PORT        Status Schd Mst  FHB  NHB |
| --- -------- -------------------- ------ ---- --- ---- ---- |
|   1      cm0      123.1.1.1/29000   DOWN    N        0    0 |
|   2      cm1      124.1.1.1/29100     UP    Y R M [ LOCAL ] |
================================================================
```

14.6. TAC 구성

본 절에서는 TAC의 구성을 위해 Linux 환경에서 참고할 수 있는 예제를 설명한다.

1번 노드에서의 TB_SID와 CM_SID는 각각 ac1와 cm1, 2번 노드에서의 TB_SID와 CM_SID는 각각 ac2와 cm2로 정하였다. 먼저 CM TIP 파일을 설정해야 한다. 위의 초기화 파라미터에 보면 필수 항목이 3개이므로 그 3개는 반드시 설정해야 한다.

1. 본 예제에서는 1번 노드를 위한 CM TIP 파일을 1번 노드의 $TB_HOME/config 아래에 cm1.tip으로, 2번 노드를 위한 CM TIP 파일을 2번 노드의 $TB_HOME/config 아래에 cm2.tip으로 저장하였으며, 다음과 같이 TIP 파일을 작성하였다(config 폴더에 저장해야 한다).

   <<cm1.tip>>

   ```
   CM_NAME=cm1
   CM_UI_PORT=8635
   CM_RESOURCE_FILE=/home/tibero7/cm1_res.crf
   ```

   <<cm2.tip>>

   ```
   CM_NAME=cm2
   CM_UI_PORT=8655
   CM_RESOURCE_FILE=/home/tibero7/cm2_res.crf
   ```

2. 다음으로 TAC용 TIP 파일을 설정해야 한다.

   본 예제에서는 1번 노드를 위한 TAC TIP 파일을 1번 노드의 $TB_HOME/config 아래에 ac1.tip으로, 2번 노드를 위한 TAC TIP 파일을 2번 노드의 $TB_HOME/config 아래에 ac2.tip으로 저장한다. “제15장 Tibero Active Cluster”를 보면 각 파라미터들의 의미를 알 수 있다(본 예제에서는 TB_HOME이 /home/tibero7 이다).

   <<ac1.tip>>

   ```
   DB_NAME=ac   #DB_NAME은 ac1과 ac2가 같은 값을 가진다.
   LISTENER_PORT=21000
   CONTROL_FILES="/home/tibero7/database/ac/c1.ctl"

   MAX_SESSION_COUNT=20
   TOTAL_SHM_SIZE=512M
   ```

```
MEMORY_TARGET=1G
THREAD=0
UNDO_TABLESPACE=UNDO0

CLUSTER_DATABASE=Y
LOCAL_CLUSTER_ADDR=123.1.1.1
LOCAL_CLUSTER_PORT=21100
CM_PORT=8635 #cm1의 CM_UI_PORT
```

<<ac2.tip>>

```
DB_NAME=ac   #DB_NAME은 ac1과 ac2가 같은 값을 가진다.
LISTENER_PORT=21010
CONTROL_FILES="/home/tibero7/database/ac/c1.ctl"

MAX_SESSION_COUNT=20
TOTAL_SHM_SIZE=512M
MEMORY_TARGET=1G
THREAD=1
UNDO_TABLESPACE=UNDO1

CLUSTER_DATABASE=Y
LOCAL_CLUSTER_ADDR=124.1.1.1
LOCAL_CLUSTER_PORT=21110
CM_PORT=8655 #cm2의 CM_UI_PORT
```

3. 1번 노드부터 구성을 하면 먼저 CM을 실행시켜야 하는데, 이를 위해서는 CM_SID가 앞서 작성한 TIP 파일의 파일 이름과 같아야 한다(따라서 본 예제에서는 CM_SID가 cm1이어야 한다).

   CM_SID를 설정하기 위해 아래의 내용을 터미널에서 실행시킨다. TB_SID도 같이 설정한다(나중에 데이터베이스를 만들 때 필요하다).

   ```
   export CM_SID=cm1
   export TB_SID=ac1
   ```

4. 이렇게 CM_SID 설정을 완료하면, 아래의 명령어를 이용해 CM을 실행한다.

   ```
   tbcm -b
   ```

   성공적으로 부팅되었다면, 다음과 같이 출력된다.

   ```
   CM Guard daemon started up.
   import resources from '/home/tibero7/cm1_res.crf'...

   Tibero 7

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

```
Tibero cluster manager started up.
Local node name is (cm1:8635).
```

이 과정을 거치고 나면 CM_RESOURCE_FILE에 설정해 놓은 디렉터리에 cm1_res.crf라는 리소스 바이너리 파일이 생성되게 된다. 이 파일에 앞으로 등록할 리소스에 대한 정보들이 저장된다.

5. 완료되면 CM의 부팅 상태를 확인하기 위해 다음과 같이 명령어를 실행한다.

```
cmrctl show
```

다음과 같이 나온다면 정상적인 상황이다(아직 아무런 리소스도 등록하지 않았기 때문에 CLUSTER, TYPE, NAME, STATUS, DETAIL이 모두 빈칸이다).

```
Resource List of Node cm1
========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------- --------- ----------------------
========================================================================
```

6. 먼저 아래의 명령어처럼 네트워크 리소스를 등록한다.

```
cmrctl add network --name net1 --ipaddr 123.1.1.1 --portno 29000
```

성공적으로 리소스가 등록되면 다음과 같이 출력된다.

```
Resource add success! (network, net1)
```

7. 다시 한 번 cmrctl show를 이용하여 리소스 상태를 확인하면 다음과 같이 출력된다.

```
Resource List of Node cm1
========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------- --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29000
========================================================================
```

8. 다음으로 아래의 명령어와 같은 방법으로 클러스터를 등록한다. 이때, cfile이 저장될 폴더는 미리 만들어 놓아야 한다. 또한, cfile 경로는 공유 디스크에 있도록 지정해야 한다.

```
cmrctl add cluster --name cls1 --incnet net1 --cfile /'shared disk 경로'/cls1_cfile
```

성공적으로 클러스터 리소스가 등록되면 다음과 같이 출력된다.

```
Resource add success! (cluster, cls1)
```

9. cmrctl show를 이용하여 리소스 상태를 확인하면 다음과 같이 출력된다.

```
Resource List of Node cm1
=========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------ --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29000
    COMMON   cluster          cls1      DOWN inc: net1, pub: N/A
=========================================================================
```

10. 등록된 클러스터 아래에 TAC 서비스 리소스를 등록해야 하는데, 그에 앞서 클러스터 cls1을 활성화시켜야 한다.

```
cmrctl start cluster --name cls1
```

성공적으로 클러스터가 시작되면 다음과 같이 출력된다(이때 실패한다면 cfile이 저장될 디렉터리를 미리 안만들어 놓았을 가능성이 높다).

```
SUCCESS!
```

11. cmrctl show를 이용하여 리소스 상태를 확인하면 다음과 같이 출력된다.

```
Resource List of Node cm1
=========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------ --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29000
    COMMON   cluster          cls1        UP inc: net1, pub: N/A
      cls1      file        cls1:0        UP /'shared disk 경로'/cls1_cfile
=========================================================================
```

12. 다음과 같은 명령어를 이용해 서비스 리소스를 등록한다(이때, 서비스의 이름은 나중에 만들 데이터베이스의 이름과 같아야 한다).

```
cmrctl add service --name ac --cname cls1
```

성공적으로 등록하면 다음과 같이 출력된다.

```
Resource add success! (service, ac)
```

13. cmrctl show를 이용하여 리소스 상태를 확인하면 다음과 같이 출력된다.

```
Resource List of Node cm1
=========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------ --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29000
```

```
    COMMON   cluster          cls1        UP inc: net1, pub: N/A
      cls1      file        cls1:0        UP /'shared disk 경로'/cls1_cfile
      cls1   service            ac      DOWN Database, Active Cluster
                                            (auto-restart: OFF)
================================================================================
```

14. 마지막으로 DB 리소스를 등록한다.

    여기서 --name의 값은 Active Cluster에서 해당 노드가 사용할 TB_SID(본 예제에서는 ac1를 사용)와 같아야 한다. ac1를 위한 환경변수 설정을 담은 envfile은 /home/tibero7/ 아래에 envfile_ac1으로 저장하였다.

```
cmrctl add db --name ac1 --svcname ac --dbhome /home/tibero7
--envfile /home/tibero7/envfile_ac1
```

    성공적으로 등록하면 다음과 같이 출력된다.

```
Resource add success! (db, ac1)
```

15. cmrctl show를 이용하여 리소스 상태를 확인하면 다음과 같이 출력된다.

```
Resource List of Node cm1
================================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------ --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29000
    COMMON   cluster          cls1        UP inc: net1, pub: N/A
      cls1      file        cls1:0        UP /'shared disk 경로'/cls1_cfile
      cls1   service            ac      DOWN Database, Active Cluster
                                            (auto-restart: OFF)
      cls1        db           ac1      DOWN ac, /home/tibero7
================================================================================
```

16. 이제 실제로 운용할 데이터베이스를 만들어야 한다. “15.5. TAC를 위한 데이터베이스 생성” 과정의 2번 ~ 6번까지의 과정을 실행한다. 과정을 수행하는 데 다음의 두 가지 유의사항을 지켜야 한다.

    – tbsql로 접속하기 위해 tbdsn.tbr을 설정할 때 다음과 같은 내용들을 넣어주어야 한다.

```
ac1=(
    (INSTANCE=(HOST=123.1.1.1)
              (PORT=21000)
              (DB_NAME=ac)
    )
)
```

– 접속 후 “15.5. TAC를 위한 데이터베이스 생성”에서 데이터베이스를 만들 때 CREATE DATABASE "ac" (앞서 입력한 서비스 리소스의 이름)로 해야 한다. 데이터베이스 생성을 모두 마치고 나면, cmrctl show의 결과에서 다음과 같이 STATUS가 모두 UP으로 바뀌어 있다.

```
Resource List of Node cm1
=========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------ --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29000
    COMMON   cluster          cls1        UP inc: net1, pub: N/A
      cls1      file       cls1:0        UP /'shared disk 경로'/cls1_cfile
      cls1   service           ac        UP Database, Active Cluster
                                            (auto-restart: OFF)
      cls1        db          ac1        UP ac, /home/tibero7
=========================================================================
```

17. 이로써 첫 노드에서의 구성이 끝났다. 이제 다음 노드를 구성해야 한다. 먼저 2번 노드의 tbdsn.tbr 설정을 마친 후 다음의 명령어들을 차례대로 입력한다. 이때 유의할 점은 클러스터를 추가할 때 cfile의 경로가 앞서 1번 노드에서 설정한 cfile의 경로와 같아야 한다.

```
export CM_SID=cm2
export TB_SID=ac2
tbcm -b
cmrctl add network --name net1 --ipaddr 124.1.1.1 --portno 29100
cmrctl add cluster --name cls1 --incnet net1 --cfile /'shared disk 경로'/cls1_cfile
cmrctl start cluster --name cls1
```

18. cmrctl show를 해보면 다음과 같이 서비스가 이미 등록되어 있는 것을 확인할 수 있다. 이는 cfile에 있는 내용을 cm1에서 직접 읽어왔기 때문이다.

```
Resource List of Node cm2
=========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------ --------- ----------------------
    COMMON   network          net1        UP (private) 124.1.1.1/29100
    COMMON   cluster          cls1        UP inc: net1, pub: N/A
      cls1      file       cls1:0        UP /'shared disk 경로'/cls1_cfile
      cls1   service           ac      DOWN Database, Active Cluster
                                           (auto-restart: OFF)
=========================================================================
```

19. 마지막으로 ac2이 사용할 envfile을 저장한 후 다음의 명령어들을 입력하면 2 노드 TAC 구성과 실행이 완료된다.

```
cmrctl add db --name ac2 --svcname ac --dbhome /home/tibero7
--envfile /home/tibero7/envfile_ac2
```

```
cmrctl start service --name ac
```

14.7. TAS-TAC 구성

본 절에서는 먼저 TAS를 구성하는 과정에 대해서 설명한다.

다음은 TAS-TAC(Tibero Active Storage - Tibero Active Cluster)의 구성을 위해 Linux 환경에서 참고할 수 있는 예제이다.

- 1번 노드에서 TAS를 위한 TB_SID를 as1, TAC를 위한 TB_SID를 ac1, CM을 위한 CM_SID를 cm1으로 한다. TB_SID를 따서 예제 내용을 as1.tip은 1번 노드의 $TB_HOME/config/as1.tip에 저장한다.
- 2번 노드에서 TAS를 위한 TB_SID를 as2, TAC를 위한 TB_SID를 ac2, CM을 위한 CM_SID를 cm2로 한다. TB_SID를 따서 예제 내용을 as2.tip은 2번 노드의 $TB_HOME/config/as2.tip에 저장한다. TAS는 DB_NAME을 지정하지 않고, TAC는 DB_NAME을 ac로 한다.

전체적인 흐름은 TAS를 구성하고 “14.6. TAC 구성”에 설명된 정보를 수정해서 수행한다.

다음은 수행과정에 대한 설명이다.

1. cm1.tip과 cm2.tip은 “14.6. TAC 구성”을 참고해서 생성한다.

   다음 TIP 파일 내용에서 메모리 사이즈 등 다양한 수치들은 예제일 뿐이고, 사용 목적에 맞게 조정할 수 있다(본 예제에서는 raw 디바이스를 사용하지 않고, 각각의 용량이 5GB인 /data/disk01과 /data/disk02라는 두 파일을 사용하여 한 컴퓨터에 두 노드를 띄울 예정이다). 디스크 구성은 "Tibero Active Storage 관리자 안내서"의 "A.1.2 디스크 준비"를 참조한다.

   <<as1.tip>>

```
LISTENER_PORT=30011
MEMORY_TARGET=2G
MAX_SESSION_COUNT=50
TOTAL_SHM_SIZE=1G

CLUSTER_DATABASE=Y #필수
THREAD=0

CM_PORT=8635  #cm1의 CM_UI_PORT
LOCAL_CLUSTER_ADDR=123.1.1.1
LOCAL_CLUSTER_PORT=30111

INSTANCE_TYPE=AS  #필수
AS_ALLOW_ONLY_RAW_DISKS=N  #RAW 디바이스를 사용하지 않으므로, 이 설정이 필요하다.
```

```
AS_THR_CNT=10
AS_DISKSTRING="/data/*"  #/data/disk01과 data/dis02를 사용하므로 /data/*값을 준다.
```

<<as2.tip>>

```
LISTENER_PORT=40011
MEMORY_TARGET=2G
MAX_SESSION_COUNT=50
TOTAL_SHM_SIZE=1G

CLUSTER_DATABASE=Y #필수
THREAD=1

CM_PORT=8655 #cm2의 CM_UI_PORT
LOCAL_CLUSTER_ADDR=123.1.1.1
LOCAL_CLUSTER_PORT=40111

INSTANCE_TYPE=AS  #필수
AS_ALLOW_ONLY_RAW_DISKS=N  #RAW 디바이스를 사용하지 않으므로, 이 설정이 필요하다.
AS_THR_CNT=10
AS_DISKSTRING="/data/*"  #/data/disk01과 data/dis02를 사용하므로 /data/*값을 준다.
```

2. TAS용 TIP 파일이 준비되면 1번 노드부터 순서대로 설정한다.

   먼저 1번 노드에서 export CM_SID=cm1으로 환경변수를 설정한다. 그 후, CM을 부팅하여 네크워크를 추가하는 곳까지 앞의 내용대로 수행하여 cmrctl show 커맨드를 수행하였을 때 다음과 같이 조회될 수 있도록 한다.

```
Resource List of Node cm1
================================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------ --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29000
================================================================================
```

3. 다음으로 클러스터를 등록해야 하는데, 다음과 같이 cfile 부분이 앞에서와 달라진다.

```
cmrctl add cluster --name cls1 --incnet net1 --cfile "+/data/*"
```

4. 성공적으로 수행 되었다면,TB_SID를 as1으로 export 한 후 다음의 커맨드를 수행하여 TAS 인스턴스를 띄운다(“14.6. TAC 구성”과 달리 클러스터를 start시키지 않는다).

```
tbboot nomount
```

5. 부팅이 완료되면 tbsql을 통해 TAS 인스턴스에 접속하여 다음과 같은 sql로 디스크 스페이스를 만든다 (TAS에서도 Tibero 혹은 TAC와 마찬가지로 tbdsn.tbr에 as1의 설정을 써주어야 tbsql로 인스턴스에 접속할 수 있다).

```
CREATE DISKSPACE ds0 NORMAL REDUNDANCY
FAILGROUP fg1 DISK
'/data/disk01' NAME disk101
FAILGROUP fg2 DISK
'/data/disk02' NAME disk201
ATTRIBUTE 'AU_SIZE'='4M';
```

6. 수행 완료 후 tbsql을 종료하고, TAS 인스턴스도 종료되었음을 확인한 후 아래의 커맨드를 통해 클러스터를 실행시킨다.

```
cmrctl start cluster --name cls1
```

7. 그 후 cmrctl show를 이용해 리소스를 확인해 보면 다음과 같이 조회된다.

```
Resource List of Node cm1
========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------- --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29000
    COMMON   cluster          cls1        UP inc: net1, pub: N/A
      cls1      file        cls1:0        UP +0
      cls1      file        cls1:1        UP +1
      cls1      file        cls1:2        UP +2
========================================================================
```

8. 이제 다음의 커맨드를 사용하여 as용 서비스 리소스를 등록한다.

```
cmrctl add service --name as --cname cls1 --type as
```

9. 다음으로 as를 등록한다. envfile은 tibero7/ 아래에 envfile_for_as1으로 저장하였고, --name의 값은 TIP 파일에 사용된 이름인 as1로 해야 한다.

```
cmrctl add as --name as1 --svcname as --dbhome /home/tibero7
--envfile /home/tibero7/envfile_for_as1
```

10. 다음의 커맨드를 통해 TAS 인스턴스를 normal 모드로 부팅시킨다.

```
cmrctl start service --name as
혹은
cmrctl start as --name as1
```

11. 위의 커맨드를 통해 TAS 인스턴스가 성공적으로 부팅되었으면, 다음과 같이 tbsql로 커맨드를 수행하여 2번 노드의 TAS 인스턴스가 사용할 스레드를 생성해야 한다.

```
tbsql sys/tibero@as1
sql> alter diskspace ds0 add thread 1;
```

12. 여기까지 왔으면 나머지 2번 노드에서도 TAS 인스턴스를 띄울 준비가 완료된 것이다.

    2번 노드에서 export CM_SID=cm2와 export TB_SID=as2 커맨드를 수행하여 환경변수를 설정해준다. 그 후 tbcm -b로 CM을 부팅하고, 다음의 커맨드들을 수행해준다(본 예제에서는 같은 컴퓨터에 두 노드를 구성하므로 네트워크의 ippaddr이 cm1에서와 같고 portno만 다르다).

```
cmrctl add network --name net1 --ipaddr 123.1.1.1 --portno 29100
cmrctl add cluster --name cls1 --incnet net1 --cfile "+/data/*"
cmrctl start cluster --name cls1
```

13. 이제 cmrctl show를 실행해보면 다음과 같이 서비스가 등록된 것을 확인할 수 있다.

```
Resource List of Node cm2
========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------- --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29100
    COMMON   cluster          cls1        UP inc: net1, pub: N/A
      cls1      file        cls1:0        UP +0
      cls1      file        cls1:1        UP +1
      cls1      file        cls1:2        UP +2
      cls1   service            as      DOWN Active Storage, Active Cluster
                                             (auto-restart: OFF)
========================================================================
```

14. 다음의 커맨드로 AS를 등록한 후 2번 노드에서도 TAS 인스턴스를 실행시킨다.

```
cmrctl add as --name as2 --svcname ac --dbhome /home/tibero7
--envfile /home/tibero7/envfile_for_as2
cmrctl start as --name as2
```

15. 정상적으로 위 과정이 수행되었다면 각 노드에서의 cmrctl show의 결과는 다음과 같이 출력된다.

    – 노드 1에서의 cmrctl show 결과

```
Resource List of Node cm1
========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------- --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29000
    COMMON   cluster          cls1        UP inc: net1, pub: N/A
      cls1      file        cls1:0        UP +0
      cls1      file        cls1:1        UP +1
```

```
      cls1      file        cls1:2        UP +2
      cls1   service            as        UP Active Storage, Active Cluster
                                             (auto-restart: OFF)
      cls1        as           as1        UP as, /home/tibero7
========================================================================
```

– 노드 2에서의 cmrctl show 결과

```
Resource List of Node cm2
========================================================================
  CLUSTER     TYPE        NAME       STATUS          DETAIL
----------- -------- ------------- --------- ----------------------
    COMMON   network          net1        UP (private) 123.1.1.1/29100
    COMMON   cluster          cls1        UP inc: net1, pub: N/A
      cls1      file        cls1:0        UP +0
      cls1      file        cls1:1        UP +1
      cls1      file        cls1:2        UP +2
      cls1   service            as        UP Active Storage, Active Cluster
                                             (auto-restart: OFF)
      cls1        as           as2        UP as, /home/tibero7
========================================================================
```

16. 두 노드에서의 TAS 구성이 완료되었다. 이제 각 노드에서 TAC 구성을 진행하면 된다. 먼저 각 노드의 $TB_HOME/config에 ac1.tip과 ac2.tip을 만드는데, 내용은 다음과 같이 한다.

<<ac1.tip>>

```
DB_NAME=ac
LISTENER_PORT=21000
CONTROL_FILES="+DS0/c1.ctl","+DS0/c2.ctl"
DB_CREATE_FILE_DEST="+DS0"
LOG_ARCHIVE_DEST="/home/ac/data/archive1"

MEMORY_TARGET=1G
MAX_SESSION_COUNT=50
TOTAL_SHM_SIZE=512M

USE_ACTIVE_STORAGE=Y
AS_PORT=30011

CLUSTER_DATABASE=Y
THREAD=0
UNDO_TABLESPACE=UNDO0

LOCAL_CLUSTER_ADDR=123.1.1.1
CM_PORT=8635
LOCAL_CLUSTER_PORT=21100
```

<<ac2.tip>>

```
DB_NAME=ac
LISTENER_PORT=21010
CONTROL_FILES="+DS0/c1.ctl","+DS0/c2.ctl"
DB_CREATE_FILE_DEST="+DS0"
LOG_ARCHIVE_DEST="/home/ac/data/archive2"

MEMORY_TARGET=1G
MAX_SESSION_COUNT=50
TOTAL_SHM_SIZE=512M

USE_ACTIVE_STORAGE=Y
AS_PORT=40011

CLUSTER_DATABASE=Y
THREAD=1
UNDO_TABLESPACE=UNDO1

LOCAL_CLUSTER_ADDR=123.1.1.1
CM_PORT=8655
LOCAL_CLUSTER_PORT=21110
```

17. TAC용 TIP 파일을 저장하였으면, 각 노드에 아래의 커맨드를 수행한다(각 노드의 TB_HOME에 envfile_ac1과 envfile_ac2를 만들었다. envfile에 대한 내용은 “14.4.1.1. cmrctl add”와 “14.6. TAC 구성”을 참고한다).

   – 1번 노드

   ```
   cmrctl add service --name ac --cname cls1
   cmrctl add db --name ac1 --svcname ac --dbhome /home/tibero7
   --envfile /home/tibero7/envfile_ac1
   ```

   – 2번 노드

   ```
   cmrctl add db --name ac2 --svcname ac --dbhome /home/tibero7
   --envfile /home/tibero7/envfile_ac2
   ```

18. 이제 1번 노드에서 “15.5. TAC를 위한 데이터베이스 생성”의 2번부터 6번까지 과정을 수행한다. 이때 DB 인스턴스를 nomount 모드로 부팅하는 커맨드로 다음의 명령어를 사용한다.

   ```
   cmrctl start db --name ac1 --option "-t NOMOUNT"
   ```

   또는

   ```
   tbboot nomount
   ```

19. "15.5. TAC를 위한 데이터베이스 생성"의 6번까지 과정을 수행하고 나면 다음의 명령어를 통해 각 노드에 DB를 띄워 TAS-TAC 구성을 마친다. 단, 데이터베이스를 생성하는 과정에서 logfile의 경로를 지정할 때에는 '+DS0/log001'(+는 TAS용 경로임을 표시하고, DS0는 앞서 생성한 디스크 스페이스 이름이다.)과 같은 TAS용 경로를 사용해야 한다.

```
cmrctl start service --name ac
```

14.8. HA Service 구성

본 절에서는 HA(High Availability) 구성을 위해 Linux 환경에서 참고할 수 있는 예제를 설명한다.

HA를 구성하는 방법은 TAC와 비슷하여 TIP 파일 설정과 서비스 리소스에 --mode ha가 들어가는 점만 다르다. 1번 노드의 TB_SID는 ha1, CM_SID는 cm1이고, 2번 노드의 TB_SID는 ha2, CM_SID는 cm2이다.

1. cm1.tip과 cm2.tip은 "14.6. TAC 구성"에서와 똑같이 작성하고, ha1.tip과 ha2.tip만 다음과 같이 작성한다(본 예제에서는 쉬운 설명을 위해 같은 컴퓨터에 두 노드를 띄웠다).

   <<ha1.tip>>

   ```
   DB_NAME=ha   #DB_NAME은 ac1과 ac2가 같은 값을 가진다.
   LISTENER_PORT=25001
   CONTROL_FILES="/home/tibero7/database/ha/c1.ctl"
   DB_CREATE_FILE_DEST="/home/tibero7/database/ha"
   LOG_ARCHIVE_DEST="/home/tibero7/database/ha/archive1"

   MAX_SESSION_COUNT=20
   TOTAL_SHM_SIZE=1G
   MEMORY_TARGET=2G

   CLUSTER_DATABASE=Y
   THREAD=0
   UNDO_TABLESPACE=UNDO0

   LOCAL_CLUSTER_ADDR=123.1.1.1
   LOCAL_CLUSTER_PORT=21100
   CM_PORT=8635
   ```

   <<ha2.tip>>

   ```
   DB_NAME=ha   #DB_NAME은 ac1과 ac2가 같은 값을 가진다.
   LISTENER_PORT=35001
   CONTROL_FILES="/home/tibero7/database/ha/c1.ctl"
   DB_CREATE_FILE_DEST="/home/tibero7/database/ha"
   LOG_ARCHIVE_DEST="/home/tibero7/database/ha/archive1"
   ```

```
MAX_SESSION_COUNT=20
TOTAL_SHM_SIZE=1G
MEMORY_TARGET=2G

CLUSTER_DATABASE=Y
THREAD=0   #TAC에서와 달리 ha1과 ha2가 같은 thread, UNDO_TABLESPACE 값을 가진다.
UNDO_TABLESPACE=UNDO0

LOCAL_CLUSTER_ADDR=123.1.1.1
LOCAL_CLUSTER_PORT=31100
CM_PORT=8655
```

2. HA용 TIP 파일이 준비되었으니, ha1과 ha2를 위한 envfile을 만든 후 다음과 같은 커맨드로 네트워크와 클러스터, 서비스, HA를 등록한다.

   – 1번 노드

   ```
   export CM_SID=cm1
   export TB_SID=ha1
   tbcm -b
   cmrctl add network --name net1 --ipaddr 123.1.1.1 --portno 29000
   cmrctl add cluster --name cls1 --incnet net1 --cfile /home/tibero7/cfile/cls1_cfile
   cmrctl start cluster --name cls1
   cmrctl add service --name ha --cname cls1 --mode ha
   cmrctl add db --name ha1 --dbhome /home/tibero7 --envfile /home/tibero7/envfile_ha1
   ```

   – 2번 노드

   ```
   export CM_SID=cm2
   export TB_SID=ha2
   tbcm -b
   cmrctl add network --name net1 --ipaddr 123.1.1.1 --portno 29100
   cmrctl add cluster --name cls1 --incnet net1 --cfile /home/tibero7/cfile/cls1_cfile
   cmrctl start cluster --name cls1
   cmrctl add db --name ha2 --dbhome /home/tibero7 --envfile /home/tibero7/envfile_ha2
   ```

3. 이제 운용할 데이터베이스를 만들어야 한다.

   먼저 NOMOUNT 모드로 부팅한 후 single Tibero에서와 같은 방식으로 데이터베이스를 생성한다. 그 후 system.sh를 수행하면 데이터베이스 생성이 완료된다. 마지막으로 다음의 커맨드를 1번 노드에 적용하면 1번 노드가 다운되었을 때 2번 노드에 DB 인스턴스가 실행되게 된다(1번 노드가 Active, 2번 노드를 Standby라고 하며 이는 cmrctl show service --name ha로 확인 가능하다).

   ```
   cmrctl act service --name ha

   cmrctl show service --name ha
   Service Resource Info
   =====================================
   ```

```
Service name    : ha
Service type    : Database
Service mode    : HA
Cluster         : cls1
Inst. Auto Start: ON
========================
| INSTANCE LIST        |
|----------------------|
| NID  Status  HA MODE |
| --- -------- ------- |
|   1       UP  Active |
|   2     DOWN Standby |
========================
```

만약 1번 노드가 다운되어 2번 노드가 자동실행되면, 1번 노드가 deact되어 있을 수 있다. 이럴 경우 deact된 리소스를 act시켜주면 1번노 드가 standby가 되어 2번 노드를 failover하게 된다.

14.9. CM Observer 구성

CM Observer는 Tibero-TSC 구조에서 관제 대상이 되는 CM들과의 통신을 토대로 TSC로의 failover 수행을 지원한다. CM Observer는 그 관제 하의 CM과의 heartbeat 및 DB 변화에 대한 메시지를 주고 받으며, 이를 기반하여 failover의 실행을 결정한다.

CM Observer를 구성하는 방법은 일반적인 CM을 구성하는 방식과 다르다. Observer는 CM TIP에 자신이 일반적인 CM이 아닌 Observer 노드로 동작함을 명시해주어야 하며, 그 관제 하의 CM 노드들과 통신할 Observer port도 명시해주어야 한다.

관제 하의 CM은 TSC 구성을 한 후 DB 서비스를 등록할 때에 Observer의 ip, port 및 TSC ID를 명시해주어야하며(CM은 이 과정을 통해 Observer에 등록된다.) 또한 DB는 TIP 파일에 몇몇 파라미터를 적용해주어야 한다. 나머지는 일반적인 TSC 구성과 유사하다.

Observer는 관제 대상 CM, Instance들과 독립된 HW에서 동작시켜야 하며, Observer가 동작하는 장비는 troubleproof해야 한다. 만약 독립된 HW가 아닌 경우 기능이 제한될 수 있다. 또한 한 TSC 내의 CM들은 서로 다른 CM NAME을 가져야 한다.

14.9.1. 환경변수 설정

CM Observer도 CM과 마찬가지로 환경변수 CM_HOME과 CM_SID를 설정해야 한다.

다음은 설정 예이다.

환경변수	설명
$CM_HOME	CM Observer가 설치된 경로를 지정한다. – 예)

환경변수	설명
	`export CM_HOME=/home/tibero7`
$CM_SID	노드를 식별할 수 있는 ID를 지정한다. 각 노드는 서로 다른 값을 가져야 한다. – 예) `export CM_SID=cmobs`

14.9.2. 파라미터 설정

본 절에서는 CM Observer용 초기화 파라미터와 Observer를 사용하는 경우 필수 DB 파라미터에 대해서 설명한다.

14.9.2.1. 초기화 파라미터

CM Observer용 TIP에는 다음과 같은 파라미터들을 설정할 수 있다. 필수 항목은 사용자가 TIP에 명시적으로 값을 지정해야 하며, 선택 항목은 사용자가 지정하지 않을 수 있고 이 경우 기본값으로 입력된다.

초기화 파라미터	필수 여부	설명
CM_NAME	필수	Observer의 identifier로 쓸 이름이다. 서로 다른 노드의 CM은 다른 값을 가져야 한다.
CM_MODE_OBSERVER	필수	CM 노드가 Observer 모드로 동작함을 명시해주어야 한다. (Y로 설정)
CM_UI_PORT	필수	cmrctl 명령어 수행할 때에 CM으로 접속하는 용도로 사용할 네트워크 포트 번호이다. 이 포트는 노드 간에 열려 있어야 한다.
CM_OBSERVER_PORT	필수	CM Observer와 그 관제대상 CM들과 연결하기 위한 용도로 사용할 네트워크 포트 번호이다.
CM_OBSERVER_EXPIRE	선택	CM Observer expire란 관제 하의 CM에 문제가 생겼다는 것을 CM Observer이 알아차리는데 필요한 표준 시간을 의미한다. 즉, 관제 하의 CM 중 Heartbeat expire 시간이 지나도록 통신이 되지 않는 CM이 있으면, CM Observer는 통신이 되지 않는 노드의 CM에 문제가 생겼다고 인식하게 된다. 이 값은 CM_HEARTBEAT_EXPIRE와 CM_NET_EXPIRE_MARGIN을 더한 값보다 작아야 한다. (단위: 초(second), 기본값: 60)
LOG_LVL_CM	선택	CM이 로그를 남기는 수준을 지정한다. 값이 높을수록 CM이 더 많은 로그를 저장하며, 1~6 사이의 정수값을 가질 수 있다. (기본값: 2)

초기화 파라미터	필수 여부	설명
CM_LOG_DEST	선택	CM이 로그를 남길 디렉터리를 지정한다. 이 값은 반드시 절대 경로여야 한다. (기본값: $CM_HOME/instance/$CM_SID/log/)
CM_LOG_FILE_SIZE	선택	CM 로그 파일의 최대 크기(용량)를 지정한다. 로그가 계속 생성되어 파일의 크기가 이 값을 넘어서려 할 경우 현재의 로그 파일이 닫히고, 새로운 파일이 생성되어 그 파일에 로그가 저장된다. 값은 100KB~1GB 사이의 정수값을 가질 수 있다. (단위: Byte, 기본값: 10MB)
CM_LOG_TOTAL_SIZE_LIMIT	선택	CM_LOG_DEST에서 지정한 디렉터리에 생성되는 로그 파일들의 크기 총합의 최댓값을 지정한다. 로그 파일들의 크기의 합이 이 값을 넘어가게 되면, 가장 오래된 파일을 삭제함으로써 파일 용량이 끝없이 증가하는 것을 막는다. 이 값은 100KB~1GB 사이의 정수값을 가질 수 있다. (단위: byte, 기본값: 300MB)
CM_TIME_UNIT	선택	CM을 컨트롤할 때 사용되는 단위 시간의 길이를 지정한다. (단위: 0.1초, 기본값: 10)

14.9.2.2. 필수 DB 파라미터

CM Observer를 정상적으로 사용하기 위해 DB TIP에 Observer에 대한 파라미터를 설정해야 한다.

파라미터	설명
STANDBY_USE_OBSERVER	일반적인 Tibero-TSC 구조에서 Observer를 통해 관제 및 Failover를 진행할지를 결정한다. (필수 설정 값: Y)

14.9.3. CM Observer 실행

CM Observer 데몬 실행 이후의 작업들은 cmrctl 명령어를 사용하여 수행한다.

CM Observer는 다음의 방법으로 실행한다.

```
tbcmobs [option]
```

- [option]

옵션	설명
-b	CM Observer 데몬을 실행한다.

옵션	설명
-d	CM Observer 데몬을 종료한다.
-s	CM Observer의 상태를 보여준다. 초기화 파라미터에 등록된 내용이 표시된다.
-h	tbcmobs 명령어에 대한 도움말을 보여준다.

14.9.4. Observer 명령어

cmrctl은 New CM 및 Observer에서 리소스들을 관리 및 제어하기 위해 사용하는 명령어 set이다. 단 지원하는 명령의 종류는 New CM일 때와 Observer일 때 서로 다르며, Observer에서 지원하는 cmrctl 명령어 용법은 다음과 같다.

14.9.4.1. cmrctl show

Observer에 등록된 리소스의 정보를 확인하기 위한 명령어이다.

다음의 2가지 방법으로 사용이 가능하다.

- 방법 1)

```
cmrctl show
```

현재 CM Observer에 등록된 CM 및 DB Instance들의 목록을 출력한다.

- 방법 2)

```
cmrctl show --tscid <tscid>
```

지정한 특정 TSC의 상세한 정보를 출력한다.

14.9.4.2. cmrctl set

Observer에 등록된 TSC의 동작을 설정하기 위한 명령어이다.

다음의 3가지 방법으로 사용이 가능하다.

- 방법 1)

```
cmrctl set --tscid <tscid> --<type> <act_var>
```

위의 명령 상으로 지원하는 type과 act_var는 다음과 같다.

type	act_var	설명
auto_failover	on	해당 TSC의 autofailover를 수행한다. (기본값)
auto_failover	off	해당 TSC의 autofailover를 수행하지 않는다.

type	act_var	설명
mode	protective	해당 TSC에 대한 Auto Failover의 동작방식을 protective 모드로 설정한다. (기본값)
mode	disaster_plan	해당 TSC에 대한 Auto Failover의 동작방식을 disaster_plan 모드로 설정한다.

– 방법 2)

```
cmrctl set --tscid <tscid> --target <target_clsid / mr>
```

지정한 특정 TSC의 failover 대상(Target)을 특정 클러스터로 설정(고정)하거나, 가장 최신의 TSN을 받은(Most Received) 클러스터가 failover 대상(Target)이 되도록 한다.

– 방법 3)

```
cmrctl set --tscid <tscid> --target <target_clsid / mr> --auto_failover <act_var>
```

Failover 대상 클러스터 설정 및 auto_failover 여부를 설정한다.

14.9.4.3. cmrctl switchover

지정된 tsc를 switchover하기 위한 명령어이다.

다음의 2가지 방법으로 사용이 가능하다.

– 방법 1)

```
cmrctl switchover --tscid <tscid>
```

현재 Target 클러스터로 switchover를 수행한다.

– 방법 2)

```
cmrctl switchover --tscid <tscid> --target <target_clsid>
```

지정한 Target 클러스터로 switchover를 수행한다.

14.9.5. Observer 구성 예시

다음은 Observer를 구성하는 과정에 대한 설명이다.

1. CM TIP 파일의 설정과 DB TIP 파일의 설정은 “13.6. TAC-TSC 구성”을 참고하여 작성한다.
2. 다음은 Observer TIP 파일의 예시이다.

   <<cmobs.tip>>

   ```
   CM_NAME=cmobs
   CM_MODE_OBSERVER=Y  # Observer로 기동시킬 것이기에 반드시 설정해야 한다.
   CM_OBSERVER_PORT=14500  # 관제 하의 CM노드에서 service를 등록할 때 이 port를 사용한다.
   CM_UI_PORT=13500

   # Observer는 resource file이 없다.
   # 따라서 Resource file의 경로를 지정할 필요가 없다.
   ```

   다음은 일반적인 CM TIP 파일의 예시이다.

   <<cm1.tip>>

   ```
   CM_NAME=cm1
   CM_UI_PORT=27500
   CM_RESOURCE_FILE=/home/tibero/cm1_res
   CM_HEARTBEAT_EXPIRE=120
   CM_WATCHDOG_EXPIRE=110

   # Observer가 아닌 CM은 resource file이 필요하다.
   # 따라서 Resource file의 경로를 지정해야 한다.
   ```

3. Observer용 TIP 파일 및 다른 CM, DB TIP 파일들이 준비되었으니, 네트워크와 클러스터, 서비스, DB를 등록한다. 대부분의 등록과정은 “13.6. TAC-TSC 구성”과 같다. 단, DB 서비스 등록과정이 약간 다른데, 아래의 예시와 같이 서비스 등록과정에서 이를 관제할 Observer의 ip와 port, TSC ID를 명시해주어야 한다(본 예제에서는 쉬운 설명을 위해 같은 컴퓨터에 옵저버와 노드를 띄웠다).

   – Observer 미사용 시 DB 서비스 등록

   ```
   cmrctl add service --type db --name tac --cname cls0
   ```

   – Observer 사용 시 DB 서비스 등록(관제 대상 CM 노드에서 서비스 등록 시)

   ```
   cmrctl add service --type db --name tac --cname cls0
                      --tscid 11 --obsip 127.0.0.1 --obsport 14500
   ```

   관제 대상 CM은 이를 통해 Observer에 등록된다.

4. 구성이 완료되면 Observer에서 cmrctl show 명령을 통해 아래와 같이 TSC가 구성되었음을 확인할 수 있다.

   – Observer가 관제하는 cm 및 DB Instance 확인

```
cmrctl show

Resource List of Observer cmobs
================================================================
 TSC_ID   CLS_ID    CM_NAME    NID   CM_STAT   INST_STAT   PRI/TAR
-------- -------- ----------- ----- --------- ----------- ---------
      11        0         cm0     1        UP    UP(NRML)   PRIMARY
                1      cm0_sb     1        UP    UP(RECO)   TARGET
================================================================
```

   – 특정 TSC에 대한 정보 확인

```
cmrctl show --tscid 11

TSC(ID: 11) Information
================================================================================
 FAILOVER     MODE      CLS_ID    CM_NAME   CONN   LOG   Heartbeat     RCVD.TSN

--------- ----------- -------- ----------- ------ ----- ---------- ------------
  ON(TSN)  PROTECTIVE        0         cm0   Y(M)    -           60            0
                             1      cm0_sb   Y(M)     1          60        13655
================================================================================
```

14.9.6. Observer 동작

다음은 Observer 동작을 위한 설정값에 대한 설명이다.

- Observer에 등록된 클러스터의 분류

설정 값	설명
Primary	Normal 모드로 부팅한 Tibero가 포함된 클러스터이며, 한 TSC 안에 복수의 Primary 클러스터는 존재할 수 없다. 만약 Normal 모드로 부팅한 Tibero가 포함된 클러스터가 복수 개라면 모두 Primary 지위를 박탈당한다.
Standby	Primary와 Observer가 등록한 Standby 클러스터, 한 TSC 안에서 여러 클러스터가 Standby일 수 있다.
Target	Standby로 등록된 클러스터 중 auto failover의 대상이 될 클러스터이며, 한 TSC 안에는 단 하나의 Target 클러스터만 존재할 수 있다.
N	위의 종류에 속하지 않는 클러스터이다.

- Auto Failover 여부

설정 값	설명
OFF	Primary가 종료되어 auto failover를 하지 않는다. 또한 Primary는 어떠한 경우에도 스스로 종료되지 않는다.
ON(TSN)	Standby 클러스터 중 received tsn이 가장 높은 Standby 클러스터를 auto failover의 대상(target)으로 설정한다.
ON(FIXED)	사용자가 지정한 Standby 클러스터를 auto failover의 대상(target)으로 설정한다.

- Auto Failover 모드

설정 값	설명
PROTECTIVE	Primary 클러스터의 안정성을 최우선으로 하는 모드로써, 어떠한 경우에도 Primary 클러스터는 스스로 종료하지 않는다. 그러나 정전 등의 이유로 관제 대상 CM까지 종료되어 버리는 경우에는 auto failover가 일어나지 못한다. 즉, Observer와 CM과의 접속이 유효할 때만 DB Instance의 상황에 따라 auto failover를 수행할 수 있다. (기본값)
DISASTER_PLAN	재난대비 모드로써, Primary 클러스터 전체가 재난 등으로 종료되었을 때도 failover를 수행할 수 있다. 즉, Primary 클러스터의 CM들이 모두 Observer와 연결이 끊겨도 auto failover가 가능하다. 그러나 특정 조건에서 Primary 클러스터가 스스로를 종료시킬 수 있다.

14.9.7. Observer 사용 시 고려 사항

Observer를 사용하는 경우 기동과 종료는 다음의 순서를 따른다.

- 기동

  1. Observer 기동
  2. Primary, Standby CM 기동 후 Observer에서 Primary, Standby CM 접속 확인 (observer cmrctl show)
  3. Primary, Standby Tibero 인스턴스 기동 후 Observer에서 정상적인 상태 확인 (Primary/Standby/Target)

- 종료

  1. Observer 종료
  2. Primary, Standby Tibero 인스턴스 종료
  3. Primary, Standby CM 종료

14.10. CM STONITH 기능 구성

cm의 cluster 구성에서 어떤 노드의 문제가 발생하게 된다면 schedule out이 진행된다. Schedule out 과정에서 해당 노드가 이미 종료되어 있지 않다면 해당 노드에게 schedule out 사실을 알려주고 종료 과정을 수행하도록 한다. 그 후 cluster에 속한 resource들의 재구성이 시작된다. (ex. service reconfiguration) 이때, 해당 노드가 hang이 걸려 종료를 처리하지 못하는 경우가 발생할 수 있다. 이 경우 cluster에 속한 resource가 재구성 되었을 때, schedule out 되었지만 아직 종료과정을 완료하지 못한 노드에 의해 오류가 발생하게 된다. 이를 해결하는 간단한 방법은 해당 노드가 종료를 알려오는 것을 기다리는 방식이다. 하지만 hang이 1-2시간 이상 지속되는 경우라면, 기다리는 해결법은 장시간 service 불가를 유발하게 된다. 따라서 정상 동작(cluster의 master로서 여러 동작들 수행) 중인 master 노드에서 schedule out된 노드가 cluster의 공유 볼륨에 접근하는 것을 차단하는 기능을 개발하게 되었다.

각 사용법을 확인 후 "14.10.3. STONITH 기능 주의 사항" 부분을 반드시 참고해야 한다.

14.10.1. sg_persist를 사용하는 STONITH 기능

14.10.1.1. 개요

본 기능의 경우 sg_persist라는 툴을 사용하여 진행된다. sg_persist는 device에 registration & reservation을 수행하기 위한 SCSI PERSISTENT RESERVE 명령어를 날려주는 툴이다. 간단하게 설명하면 각 노드는 device에 registration을 통해 자신의 key를 등록할 수 있고 이 key를 이용해 해당 device의 접근 방식을 reservation할 수 있다. 기본적으로 write exclusive mode 즉, key를 등록한 노드만 write을 할 수 있는 모드로 설정하였다. 그리고 어떤 node가 schedule out 시 해당 node의 key를 제거함으로써 더이상 device에 write을 수행하지 못하도록 만들었다. 이를 통해 schedule out된 노드가 공유 볼륨을 접근해서 유발되는 문제를 차단할 수 있게 된다.

자신의 노드에서 종료를 진행하는 기존의 fencing 방식과 달리 sg_persist를 이용하게 되면 다른 노드에서 fencing을 진행하게 되며, 이러한 방식을 일반적으로 STONITH(Shoot The Other Node In The Head)라고 부른다.

14.10.1.2. 설정 방법

device를 multipath를 이용하여 구성했을 경우, "14.10.2. mpathpersist를 사용하는 STONITH 기능" 설명을 참고한다.

sg_persist를 사용하기 위해서는 관련 툴이 설치되어 있어야 한다. 아래 명령어를 통해 관련 툴이 설치되어 있는지 확인 가능하다.

```
$ rpm -qa | grep sg3_utils
// 혹은
$ dpkg --get-selections | grep sg3_utils
// 또는 다른 패키지 확인 명령어
```

설치되어 있지 않는 경우, 설치 명령어를 통해 설치를 진행해야 한다.

```
$ sudo yum install sg3_utils
// 또는 다른 설치 명령어를 통한 sg3_utils 설치
```

sg_persist가 최신 버전으로 올라가며 여러가지 오류 수정이 있어 아래와 같이 버전 업데이트를 수행해야 한다. (2020년 이후 출시 버전을 추천한다.)

```
$ yum update sg3_utils
// 혹은 다른 버전 업데이트 명령어
```

sg_persist를 사용하기 위해서는 **CM의 root 권한 기동**이 필요하다.

CM tip에 관련 IPARAM설정이 필요하다.

```
$ cat cm.tip
_CM_STONITH_THREAD=Y
...
```

cluster add 시 --stonith 옵션을 통해 관리할 디바이스 목록을 입력해 주어야 기능이 동작한다. 이 옵션의 경우 **모든 cluster에서 입력**해야 하며 입력하지 않을 경우 cluster join이 거부된다.각 cluster에서 입력한 디바이스 목록은 **동일 디바이스를 가리키고 있어야 하며** 그렇지 않을 경우 이상 동작을 하게 된다.

아래와 같은 방식으로 옵션을 추가하여 입력한다.

```
$ cmrctl add cluster --name cls1 --incnet net1 --stonith "/dev/raw/raw5[0-8]"
--cfile "+/dev/raw/raw5[0-8]"
```

추가한 정보를 확인하는 방법은 다음과 같다.

```
$ cmrctl show cluster --name cls1 --option stonith
```

14.10.2. mpathpersist를 사용하는 STONITH 기능

14.10.2.1. 개요

서버에서 한 device에 대해 물리적으로 분리된 여러 I/O path를 가지게 할 수 있고 이를 device mapper multipath 기능이라고 한다. 이 multipath로 구성된 device에 대해서 linux의 경우 sg_persist의 multipath 전용의 mpathpersist라는 툴을 이용한다. 해당 툴의 경우 **linux에서만 동작 확인이 되었으며, solaris / hp-ux 등은 별도의 테스트가 필요하다.** 또한, linux의 경우에도 /etc/multipath.conf 에 설정할 내용 중 **redhat 7.5 이상 버전에서만 동작**하는 내용이 있어 우선 7.5 이상 버전에서만 지원하고 있다. (7.5 release date 2018-04-10) (cat /etc/redhat-release 로 OS 버전 확인이 가능하다.)

14.10.2.2. 설정 방법

mpathpersist가 최신 버전으로 올라가며 여러가지 오류 수정이 있어 아래와 같이 버전 업데이트를 수행해야 한다. (2020년 이후 출시 버전을 추천한다.)

```
$ yum update device-mapper-multipath
// 혹은 다른 버전 업데이트 명령어
```

sg_persist 버전과 마찬가지로 **root 권한 기동이 필요하다.** 루트 권한으로 /etc/multipath.conf 의 defaults 란에 reservation_key file 옵션 추가가 필요하다.

```
# cat /etc/multipath.conf
...
defaults {
     ...
     reservation_key file
}
...
```

그 후 루트 권한으로 아래 명령어를 수행한다.

```
# service multipathd reload
```

reload 후 루트 권한으로 multipath -ll 을 입력해 보았을 때 정상적으로 뜨지 않고 config file parsing 실패와 같은 에러 message가 뜬다면 옵션을 지원하지 않는 버전이므로 /etc/multipath.conf의 reservation_key file을 지우고 service multipathd start를 수행해주어야 한다.

CM tip에 관련 IPARAM설정이 필요하다.

```
$ cat cm.tip
_CM_STONITH_THREAD=Y
_CM_STONITH_SG_CMD=mpathpersist
...
```

sg_persist 사용 시와 마찬가지로 cluster add시 --stonith 옵션을 통해 관리할 디바이스 목록을 입력해 주어야 기능이 동작한다. 이 옵션의 경우 **모든 cluster에서 입력**해야하며 입력하지 않을 경우 cluster join이 거부된다. mpathpersist를 쓸 경우 cluster add 시 --stonith 옵션의 입력값으로 multipath 경로를 입력해야 한다. multipath -ll을 통해 나오는 경로로 --stonith "/dev/mapper/mpatha" 혹은 --stonith "/dev/dm-3" 와 같이 입력해야 한다. 해당 경로를 별도로 링크를 걸어둔 상태라면 그 링크를 입력해도 무방하다.

```
$ cmrctl add cluster --name cls1 --incnet net1 --pubnet pub1 --stonith "/dev/mapp
er/mpatha" --cfile "+/dev/raw/raw5[0-8]"
```

14.10.3. STONITH 기능 주의 사항

14.10.3.1. 스토리지 관련 주의사항

스토리지가 SCSI-3 Persistent Reservation을 지원해야 한다. 만약 지원하지 않는다면, 일반적으로 --stonith 옵션을 준 cluster add가 실패하고 기능을 사용할 수 없음을 의미한다.

스토리지가 별도의 툴을 통하여 이미 관리되고 있는 상태라면 사용이 불가하다. 해당 툴과 어떠한 문제를 일으킬 지 모르기 때문이다.

14.10.3.2. 기능 사용 관련 주의사항

본 기능은 기본적으로 지속적으로 사용하는 것을 가정하여 만들어 졌다. 따라서 전체 노드를 종료하더라도 device에 등록된 정보가 남아 있으며, 새로 cm이 기동하여 cluster가 최초 기동할 때, 이 정보를 초기화하고 재등록 하게 된다.

만약 전체 cluster 종료 후 cluster에 stonith 옵션으로 입력한 device에 대해 어떠한 작업을 수행해야 한다면, 아래 기술할 stonith 기능 해제 방법을 참고하여 해제 후 수행하거나 작업할 노드의 cm 기동 후 cluster 수준까지 기동하여 노드가 권한을 가져온 후 수행해야 한다.

혹은 기능을 사용하다 사용하지 않게 될 경우 device에 등록된 정보를 제거해 주어야 하며 이 방법은 다음과 같다.

아래 동작들은 반드시 전체 cluster가 종료된 후 수행되어야 한다.

device에 등록된 정보를 제거하는 과정이며 해당 device가 공유 볼륨이라 한 노드에서만 수행해도 전체에 적용이 되어 sg_persist, mpathpersist 방법 모두 한 노드에서만 수행하면 된다.

1. cm에 등록된 정보를 이용하여 제거 (**반드시 cluster stop 상태에서 수행해야 함**)

```
$ cmrctl deact cluster --name cls1 --option clear
```

2. sg_persist 명령어를 이용하여 제거 (**루트 권한 필요**)

```
# sg_persist --out --register-ignore --param-sark=434d [device 경로]
# sg_persist --out --clear --param-rk=434d [device 경로]
```

device 경로에 /dev/raw/raw*와 같이 hint를 주는 것은 동작하지 않으며 device 개수마다 별도로 수행해 주어야 한다.

mpathpersist의 경우 sg_persist 부분을 mpathpersist로 대체하면 된다.

제15장 Tibero Active Cluster

본 장에서는 Tibero Active Cluster의 기본 개념과 구성요소, 프로세스, 실행 및 운영 방법을 설명한다.

15.1. 개요

Tibero Active Cluster(이하 TAC)는 확장성, 고가용성을 목적으로 제공하는 Tibero의 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.

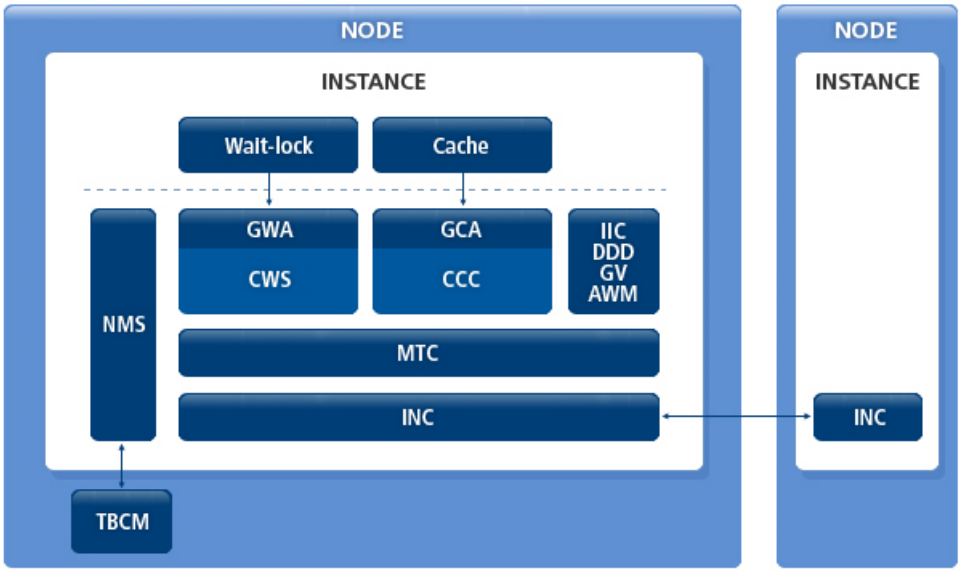
큰 업무를 작은 업무의 단위로 나누어 여러 노드 사이에 분산하여 수행할 수 있기 때문에 업무 처리 시간을 단축할 수 있다.

여러 시스템이 공유 디스크를 기반으로 데이터 파일을 공유한다. TAC 구성에 필요한 데이터 블록은 노드 간을 연결하는 고속 사설망을 통해 주고받음으로써 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.

15.2. 구성요소

다음은 TAC 구조를 나타내는 그림이다.

[그림 15.1] 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 스레드로는 WATH, WLGC, WRCF이 존재한다.
- 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 CCC에 접근할 수 있으며 이와 관련된 배경 스레드로 CATH, CLGC, CRCF이 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, UDP 등의 프로토콜을 관리한다.

- NMS(Node Membership Service)
  - TBCM으로부터 전달받은 정보(node id, ip address, port, incarnation number)와 node workload를 나타내는 가중치(weight)를 관리하는 모듈이다.
  - node 멤버십의 조회, 추가, 삭제 기능을 제공하며 이와 관련된 배경 스레드로 NMGR이 있다.

15.3. 프로세스

TAC는 1개의 프로세스(ACSD, Active Cluster Service Daemon)가 추가로 생성된다. 이 프로세스는 ACCT, NMGR, DIAG, WRCF, CRCF, WLGC, CLGC, WATH, CATH, CMPT 스레드(thread)로 구성되며, 각 스레드는 다음과 같은 그룹에 각각 포함된다.

- **Active Cluster Control Thread**

스레드	설명
ACCT	ACCT는 클러스터 간 메시지 통신을 담당하는 스레드이다. 클러스터 내의 원격 노드로부터 CWS/CCC의 lock operation과 reconfiguration 요청(request)을 받아 CMPT에게 전달해주거나, 내 노드의 세션으로부터 전송해야 할 메시지를 넘겨 받아 원격 노드로 전송하는 스레드이다. 또한, ACSD 프로세스의 메인 스레드로서 나머지 스레드를 감독하는 역할을 한다.

- **Diagnostic Thread**

스레드	설명
DIAG	DIAG는 클러스터 간 주고받는 요청에서 hang이 발생했을 때 이상 여부를 추후에 확인하기 위해 필요한 정보를 자동으로 덤프(dump)하는 스레드이다.

- **Cluster Message Processor(Message handler)**

스레드	설명
CMPT	CMPT는 다른 노드에서 보낸 메시지의 요청을 처리하는 스레드이다. CMPT 스레드는 ACF_CMPT_CNT 초기화 파라미터에 설정된 값만큼 공통 풀(pool)로 생성된다. CMPT는 ACCT로부터 메시지를 받아 주로 다음과 같은 일을 수행한다. – 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.

스레드	설명
	– 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.

- **Asynchronous Thread**

스레드	설명
WATH, CATH	CWS/CCC에서 세션을 담당하는 워킹 스레드가 처리해야 할 비동기적 업무를 대신 수행하는 스레드이다. WATH, CATH 스레드는 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 CATH 스레드만이 수행할 수 있다). – BLKW가 master로부터 받은 global write request 처리를 완료한 후 CATH에게 통보해 준다. 또한 master에게 write done notify를 보낸다. 이 특징은 CATH 스레드만이 수행할 수 있다. – shadow resource block를 reclaim하기 전에 master에게 MC lock에 대한 제거 요청을 보낸다. 요청을 보낸 후 이에 대한 응답을 받아서 처리한다.

- **Garbage Collector**

스레드	설명
WLGC, CLGC	주기적으로 lock resource을 관리하고 타임아웃을 체크하는 스레드이다. WLGC, CLGC 스레드는 다음과 같은 특징이 있다. – DDD(Distributed Deadlock Detection)를 수행하기 위해 주기적으로 타임아웃이 발생한 lock waiter를 체크한다. 체크할 때 교착 상태가 발생하면 DDD를 시작한다. – MTC retransmission queue에 설정된 메시지를 주기적으로 검사해서 타임아웃이 발생한 메시지를 다시 전송한다. – 주기적으로 TSN의 동기화를 수행한다(이 특징은 WLGC 스레드만이 수행할 수 있다). – lock resource를 위한 공유 메모리가 부족하게 되면 resource block reclaiming을 시작하여 필요한 리소스를 확보한다.

- **Reconfigurator**

스레드	설명
WRCF, CRCF	NMGR 스레드로부터 node join 및 leave event를 받아 CWS/CCC lock remastering/reconfiguration을 수행한다.

- **Node Manager**

스레드	설명
NMGR	TBCM과 통신하여 node join 및 leave event를 받아 처리하며 node 멤버십을 관리한다. 또한 WRCF, CRCF 스레드에 의해 수행되는 CWS/CCC reconfiguration을 통제(suspend 또는 resume)한다.

15.4. 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>>

```
MEMORY_TARGET=6144M
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
CM_PORT=30000
```

초기화 파라미터	설명
MEMORY_TARGET	인스턴스가 사용할 전체 메모리의 크기를 설정한다. 이것은 공유 메모리, 정렬 및 해시 등의 메모리를 요구하는 연산에서 사용하는 메모리, DBMS 내부에서 사용되는 기타 메모리를 모두 포함한다. 이 중에 공유 메모리는 DBMS의 부팅과 동시에 점유하게 되지만 나머지 메모리는 필요에 따라 할당 및 해제를 반복하게 된다.
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.

초기화 파라미터	설명
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 테이블 스페이스의 이름으로 각 인스턴스마다 고유하게 부여한다.
LOCAL_CLUSTER_ADDR	TAC 인스턴스 간에 통신할 내부 IP 주소를 설정한다.
LOCAL_CLUSTER_PORT	TAC 인스턴스 간에 통신할 내부 포트 번호를 설정한다. 이 포트는 노드 간에 열려 있어야 한다.
CM_PORT	인스턴스가 CM과 통신하기 위한 포트 번호를 설정한다. 접속할 CM의 초기화 파라미터 중 CM_UI_PORT 파라미터와 동일하게 설정한다.

다음은 TAC를 설정할 때 주의해야 할 사항이다.

- Redo 스레드의 번호와 Undo 테이블 스페이스의 이름은 동일한 데이터베이스를 서비스하는 서버 사이에서 유일해야 한다. "15.5. 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.

15.5. 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 한 번만 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. Tibero와 관련된 환경설정 파일을 설정한 후 TBCM을 기동한다. CM을 설정하고 기동하는 자세한 방법은 "제14장 Tibero Cluster Manager"를 참고한다.

   ```
   $ tbcm -b
   ```

   TBCM을 기동했지만 Tibero를 직접 제어하지는 않는다.

2. NOMOUNT 모드로 Tibero를 기동한다.

   ```
   $ tbboot -t NOMOUNT -c
   ```

3. SYS 사용자로 접속한 후 CREATE DATABASE 문을 통해 데이터베이스를 생성한다.

```
[tibero@tester ~]$ tbsql sys/tibero

SQL> CREATE DATABASE "ac"                ... ① ...
     USER sys IDENTIFIED BY tibero
     MAXINSTANCES 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_NAME을 지정한다.

② 접근할 서버의 최대 인스턴스의 개수를 8로 지정한다.

기존의 CREATE DATABASE 문과 비교해 달라진 부분은 없다. 다만, 데이터베이스 파일을 공유할 인스턴스의 최대 개수를 나타내는 MAXINSTANCE 파라미터를 주의해야 한다. 이 파라미터의 값은 컨트롤 파일과 데이터 파일의 헤더 등에 영향을 미치며 설정된 값 이상으로 Tibero의 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 테이블 스페이스 및 Redo 로그 파일은 첫 번째 인스턴스를 위한 것으로 다른 인스턴스가 데이터베이스에 접근하기 위해서는 별도의 Redo 로그 그룹과 Undo 테이블 스페이스를 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 Tibero가 자동으로 종료된다. Tibero를 다시 기동한 후 SYS 사용자로 접속하여 다음과 같이 Undo 테이블 스페이스와 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 테이블 스페이스 **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.

주의

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 추가하려면 위와 같은 과정을 참고하여 Undo 테이블 스페이스와 Redo 스레드를 생성하고 스레드를 활성화한다.

---

5. TAC raw device 환경 또는 공유 파일 시스템이면서 DB_CREATE_FILE_DEST가 적절한 경로로 지정되지 않은 환경에서는 Tibero가 정상적으로 기동되지 않을 수 있다. 이와 같은 경우 다음과 같이 TPR 관련 정보를 저장할 테이블 스페이스(SYSSUB)를 먼저 추가해야 한다.

```
[tibero@tester ~]$ tbsql sys/tibero

SQL> CREATE TABLESPACE SYSSUB DATAFILE '<SYSSUB 위치>/syssub001.dtf' ...;

Tablespace 'SYSSUB' created.
```

참고

이 과정을 생략하고 다음 단계를 수행한 경우 "Tibero 설치 안내서"의 "7장 TAC 설치와 제거"와 "Appendix A. 설치 후 문제 해결"을 참고한다.

6. $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/tibero7/scripts/pkg_standard.sql...
    Running /home/tibero/tibero7/scripts/pkg_dbms_output.sql...
    Running /home/tibero/tibero7/scripts/pkg_dbms_lob.sql...
    Running /home/tibero/tibero7/scripts/pkg_dbms_utility.sql...
    Running /home/tibero/tibero7/scripts/pkg_dbms_obfuscation.sql...
    Running /home/tibero/tibero7/scripts/pkg_dbms_transaction.sql...
    Running /home/tibero/tibero7/scripts/pkg_dbms_random.sql...
    Running /home/tibero/tibero7/scripts/pkg_dbms_lock.sql...
    Running /home/tibero/tibero7/scripts/pkg_dbms_system.sql...
    Running /home/tibero/tibero7/scripts/pkg_dbms_job.sql...
    Running /home/tibero/tibero7/scripts/pkg_utl_raw.sql...
    Running /home/tibero/tibero7/scripts/pkg_utl_file.sql...
    Running /home/tibero/tibero7/scripts/pkg_tb_utility.sql...
Creating public synonyms for system packages...
   ........................................
```

7. 다른 서버의 인스턴스를 기동하고 운영한다.

15.6. TAC 실행

본 절에서는 TAC 실행에 필요한 사항과 데이터베이스 생성, TAC의 기동 및 모니터링 방법에 대해서 설명한다.

15.6.1. 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 테이블 스페이스 파일
    - 임시 테이블 스페이스 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 권한, RAW 파일 시스템을 사용할 경우에는 각 RAW 파일의 권한을 TAC가 읽고 쓸 수 있도록 설정한다.
- 내부 네트워크의 설정

  TAC의 실행과 운영을 위해 내부 네트워크는 외부 네트워크와 분리하는 것이 데이터베이스 성능에 좋다. 내부 네트워크의 인터페이스와 IP, 속도 등을 점검한다.

참고

TAC에서 내부 네트워크를 구성할 때 크로스오버 케이블을 이용한 방식은 지원하지 않는다.

15.6.2. 데이터베이스 생성

TAC를 처음 기동할 때는 데이터베이스를 생성해야 한다. 클러스터로 사용할 한 노드에서 생성하면 된다. TAC로 Tibero를 기동할 때에는 CM이 필수이므로 CM을 먼저 설정한 후 시작하고 Tibero를 NOMOUNT 모드로 기동한다. CM 설정에 관한 내용은 자세한 방법은 “제14장 Tibero Cluster Manager”를 참고한다.

다음은 CM 설정을 마친 후 Tibero를 NOMOUNT 모드로 기동하는 예이다.

```
tac1@tester ~]$ tbboot -t NOMOUNT
listener port = xxxx
change core dump dir to /home/tibero7/bin/prof

Tibero 7

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

Tibero instance started up (NOMOUNT mode).

tac1@tester ~]$
```

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.

15.6.3. TAC 기동

데이터베이스 생성과 다른 인스턴스를 위한 환경설정 파일의 생성이 모두 완료하면 TAC를 기동할 수 있다.

다음은 다른 인스턴스를 모두 실행하고 CM, Tibero 순으로 TAC를 기동하는 예이다.

```
tac2@tester ~]$ tbcm -b
CM Guard daemon started up.
import resources from 'CM TIP에 입력된 resource file path'...

Tibero 7

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

Tibero cluster manager started up.
Local node name is (cm0:xxxx).

tac2@tester ~]$ tbboot
listener port = xxxx
change core dump dir to /home/tibero7/bin/prof
```

```
Tibero 7

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

Tibero instance started up (NORMAL mode).

tac2@tester ~]$
```

15.6.4. TAC 모니터링

$TB_HOME/scripts 디렉터리에 있는 cm_stat.sh 스크립트 파일을 통해 노드 및 인스턴스의 상태를 모니터링할 수 있다. TAC에 참여한 노드의 현재 상태, IP 정보 등을 주기적으로 확인할 수 있다.

Tibero는 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 15.1] 글로벌 뷰의 조회 - GV$SESSION

```
SQL> SELECT * FROM GV$SESSION;
```

참고

글로벌 뷰에서 조회되는 INST_ID는 INSTANCE_NUMBER와는 별개의 값이며, CM 기동 순서에 따라 변경될 수 있다.

제16장 Parallel Execution

본 장에서는 Parallel Execution의 기본개념과 동작원리를 소개하고 이를 유형별로 실행하는 방법을 설명한다.

16.1. 개요

Parallel Execution(이하 PE)은 한 개의 작업을 여러 개로 나누어 동시 처리를 실행하는 방법으로 데이터 웨어하우스(DW: Data Warehouse)와 BI(Business Information)를 지원하는 대용량 DB에서 발생하는 데이터 처리의 응답 시간을 획기적으로 줄일 수 있다. 또한 OLTP(On-Line 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 또는 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

16.2. 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

연산은 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.

Tibero에서는 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.

16.2.1. DOP 결정

하나의 SQL 문장(쿼리)에서 하나의 DOP로 PE를 수행하며 한 쿼리에 여러 개의 Parallel Hint가 있거나, Parallel Hint가 있는 동시에 Parallel Option을 가진 테이블을 쿼리에 포함하는 경우 그 중 가장 큰 DOP를 그 쿼리의 DOP로 결정한다. 명시된 DOP가 0보다 작거나 같은 경우에는 디폴트 값 4로 DOP를 결정한다. DOP는 Parallel Hint에 명시할 수 있으며, 사용 방법은 "Tibero 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.

16.2.2. DOP에 따른 워킹 스레드 할당

Tibero에서는 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 않는다.

16.3. 동작 원리

Tibero에서는 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.

16.3.1. 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.

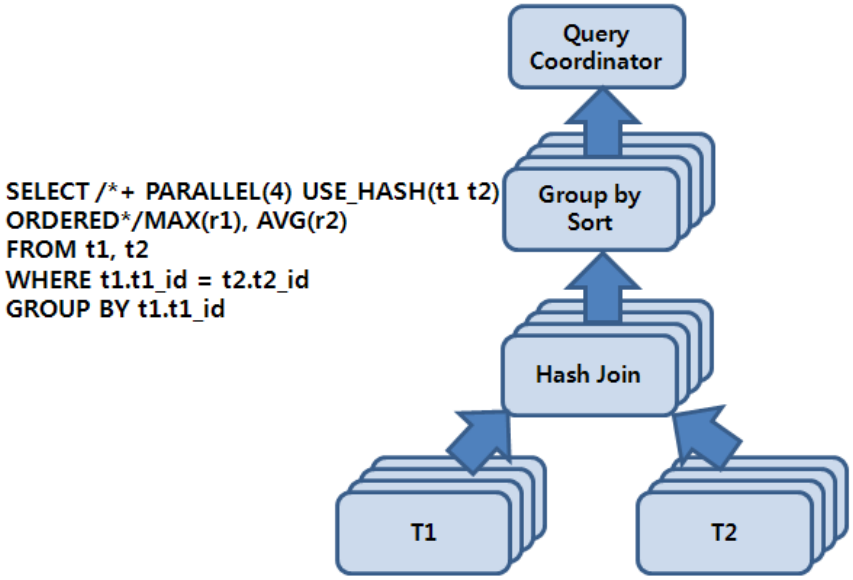
동시에 수행되는 PE_SLAVE set는 Producer set와 Consumer set으로 나뉜다.

구분	설명
Producer set	특정 부분의 실행 계획을 수행하면서 로우(중간 결과)를 추출하여 실행 계획에 명시된 방법에 따라 Consumer set의 PE_SLAVE에게 나눠준다.

구분	설명
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 16.1] Parallel Execution



위 [그림 16.1]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 때문에 하나의 PE_SLAVE set이 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.

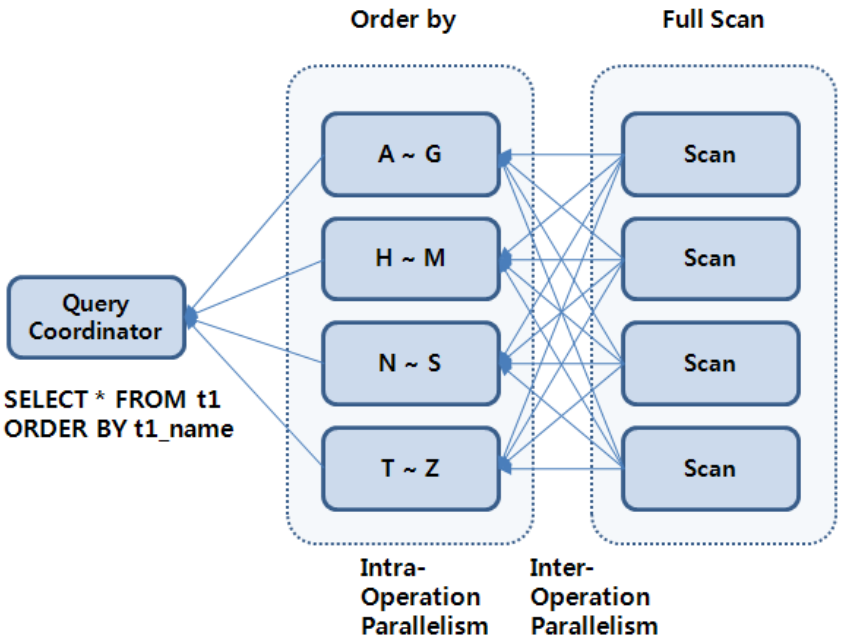
16.3.2. TPS 분배

TPS는 TCS에 연산의 결과물을 공급하여 TCS가 다음 연산을 처리할 수 있도록 한다. TPS가 결과물을 분배하는데 Tibero는 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 “16.4.1. 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 16.2]에서와 같이 range 방식을 예를 들어 설명한다.

[그림 16.2] 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.

16.4. Parallelism 유형

Tibero가 제공하는 parallel 연산은 다음과 같다.

- 액세스 방법

  테이블 스캔 그리고 index fast full scan 등

- 조인 방법

  Nested Loop, Sort Merge, 해시 조인

- DDL

  CREATE TABLE AS SELECT, CREATE INDEX, REBUILD INDEX, REBUILD INDEX PARTITION

- DML

  INSERT AS SELECT, UPDATE, DELETE

- 기타 연산

  GROUP BY, ORDER BY, SELECT DISTINCT, UNION, UNION ALL, Aggregations

16.4.1. Parallel Query

Parallel Hint가 사용되었거나 Parallel Option이 있는 테이블에 쿼리를 수행하면 Tibero는 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
2    PE SEND QC (RANDOM)
3      PE BLOCK ITERATOR
4        TABLE ACCESS (FULL): T1
```

위의 실행 계획에서는 PE SEND와 PE RECV(또는 leaf 노드) 사이를 하나의 PE_SLAVE라고 볼 수 있으며 PE_SLAVE의 최상단에는 PE SEND가 있어 TPS인 경우 TCS에게 로우를 분배한다.

항목	설명
PES(Parallel Execution Set)	Parallel Plan에서 하나의 PE_SLAVE set가 수행될 단위를 의미한다. 이를 PE_SLAVE라고도 부른다. Parallel Execution에서 하나의 연산을 하나의 PE_SLAVE set이 담당하지 않는다.
producer slave	Producer set의 PE_SLAVE이다.
consumer slave	Consumer set의 PE_SLAVE이다.

PE_SLAVE의 최하단에는 로우를 만드는 테이블 스캔(또는 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
2    PE SEND QC (RANDOM)
3      HASH JOIN (BUFFERED
4        PE RECV
5          PE SEND (HASH)
6            PE BLOCK ITERATOR
7              TABLE ACCESS (FULL): T1
8        PE RECV
9          PE SEND (HASH)
10           PE BLOCK ITERATOR
11             TABLE ACCESS (FULL): T2
```

항목	설명
PE MANAGER	PE를 시작하고 PE_SLAVE set을 coordination하는 연산이다.

항목	설명
	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 idxm이 있다. 자세한 내용은 "16.3.2. TPS 분배"를 참고한다. PE_SLAVE에서 QC로 로우를 보내는 방법은 다음과 같다. – QC random : QC 역할을 하는 스레드에 순서와 상관없이 보낸다. – QC order : QC 역할을 하는 스레드에 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. granule는 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
2  PE SEND QC (RANDOM)
3    NESTED LOOPS
4      BUFF
5        PE RECV
6          PE SEND (BROADCAST)
7            PE BLOCK ITERATOR
8              TABLE ACCESS (FULL): T1
9      PE BLOCK ITERATOR
10        TABLE ACCESS (FULL): T2
```

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
2  PE SEND QC (RANDOM)
3    NESTED LOOPS
4      BUFF
5        PE RECV
6          PE SEND (BROADCAST)
7            PE BLOCK ITERATOR
8              TABLE ACCESS (FULL): T1
9      PE BLOCK ITERATOR
10        TABLE ACCESS (FULL): T2
```

```
SQL> SELECT /*+parallel(3) pq_distribute (t2 none broadcast) use_nl(t2) ordered*/ *

     FROM t1, t2
     WHERE a=c;

Explain Plan
-------------------------------------------------------------------------------
1 PE MANAGER
2  PE SEND QC (RANDOM)
3    NESTED LOOPS
4      PE BLOCK ITERATOR
5        TABLE ACCESS (FULL): T1
6      TABLE ACCESS (FULL): T2
```

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

16.4.2. Parallel DDL

Tibero는 **'create table ... as select, create index, rebuild index'**를 Parallel로 수행할 수 있다.

'create table ... as select'는 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 SQL 참조 안내서"를 참고한다.

16.4.3. Parallel DML

Tibero는 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 Hint를 사용해야 DML을 Parallel로 수행한다. 'enable parallel dml'을 하지 않으면 DML 문에 힌트를 명시해도 Parallel로 수행하지 않는다. 즉, 'disable parallel dml'로 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 서브 쿼리에 Parallel Hint를 주어 insert뿐만 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
- 반환문이 있는 DML인 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

16.5. Parallel Execution Perfomance 분석을 위한 뷰

Tibero가 제공하는 parallel 관련 뷰는 다음과 같다.

뷰	설명
V$PE_SESSION	쿼리 서버 세션에 관한 데이터를 보여준다. 현재 Parallel Execution을 하고 있는 세션에 관한 정보를 실시간으로 보여주며, 요청된 DOP와 해당 연산에 허가된 실제 DOP도 보여준다. 추가적으로 다른 정보를 더 확인하기 위해서는 동일한 SID로 V$SESSION을 확인한다.
V$PE_TQSTAT	Parallel Execution 동안의 traffic을 Table queue 수준에서 보여준다. Table queue는 쿼리 서버간 또는 쿼리 서버와 Coodinator간의 파이프 라인이다. 이 뷰를 통해 각 producer에서 consumer로 간 데이터의 수와 크기를 확인하고, 로드가 균형있게 처리되고 있는지를 확인할 수 있다. 이 뷰는 해당 세션에서 Parallel Execution을 실행했을 때 값을 가지게 되며, 해당 세션에서 실행한 마지막 Parallel Execution에 대한 정보를 갖고 있다.
V$PE_PESSTAT	Parallel Execution이 진행되는 동안 각 스텝에서의 소요시간을 나타낸다. 각 스텝은 producer에서 consumer로 데이터 전송이 끝나고 consumer가 새로운 producer가 되거나 producer에서 coodinator로 데이터 전송이 끝날 때까지를 의미한다. 이 뷰를 통해서는 각 producer, consumer에서 수행한 연산을 확인하기 어렵기 때문에 V$PE_PESPLAN 뷰를 사용할 것을 권장한다. V$PE_PESSTAT, V$PE_PESPLAN 두 뷰 모두 V$PE_TQSTAT 뷰와 동일하게 해당 세션에서 마지막으로 수행한 Parallel Execution에 대한 정보를 갖고 있다. 만약 해당 세션에서 Parallel Execution을 수행한 적이 없을 경우 데이터가 존재하지 않는다.
V$PE_PESPLAN	해당 세션에서 마지막으로 수행한 Parallel Execution의 플랜과 Parallel Execution의 수행시간을 보여준다. 어떤 연산이 Parallel로 진행되었고 consumer로의 데이터 전송이 얼마나 걸렸는지 알 수 있고 producer에서 range 방식으로 send할 경우 샘플 수집 시간도 보여준다.

제17장 Tibero Performance Repository

본 장에서는 Tibero의 성능 진단을 위해서 제공되는 Tibero Performance Repository에 대해서 설명한다.

17.1. 개요

Tibero DBMS는 DBA가 성능 문제를 진단하는데 도움을 주기 위해 다양한 종류의 통계를 제공하고 있다. **Tibero Performance Repository(이하 TPR)**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.

TPR은 다음의 주요 기능를 수행한다.

- 스냅샷 저장 기능

  스냅샷 저장 기능은 _vt_jcntstat, 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. 단, 현재 세션 상태 저장 기능은 부하 상황에 취약할 수 있으므로 환경에 따라 주기를 더 늘리기를 권장한다.

17.2. TPR 사용법

17.2.1. tip 설정

스냅샷 저장 기능을 사용하려면 tip 파일에 'TIBERO_PERFORMANCE_REPOSITORY=Y'로 설정하고, 세션 상태 저장 기능을 사용하려면 'ACTIVE_SESSION_HISTORY=Y'로 설정하면 된다. 대부분의 경우 이 정도면 충분하다. 이때 TOTAL_SHM_SIZE는 2GB 초과값으로 설정해야 한다.

그 밖의 설정을 하기 위해서는 다음과 같은 파라미터를 조절한다.

파라미터	설명
TIBERO_PERFOR MANCE_REPOSITORY	'Y'로 설정하면 스냅샷 저장 기능 활성화한다. (기본값: Y)
TPR_SNAPSHOT_SAM PLING_INTERVAL	스냅샷을 추출하는 주기를 설정한다. (기본값: 60, 단위: 분)
TPR_SNAPSHOT_RETENTION	스냅샷을 최대 저장할 기간을 설정한다. (기본값: 7, 단위: 일)
TPR_SNAP SHOT_TOP_SQL_CNT	리포트에 출력할 상위 SQL 개수를 설정한다. (기본값: 5, 단위: 개)
TPR_SEGMENT_STATISTICS	'Y'로 설정하면 TPR에서 Segment별 Stat 수집 기능을 활성화한다. (기본값: N)
TPR_SNAPSHOT_TOP_SEG MENT_CNT	리포트에 출력할 상위 Segment 개수를 설정한다. (기본값: 5, 단위: 개)
TPR_METRIC	'Y'로 설정하면 TPR METRIC 기능을 활성화한다. (기본값: N)
TPR_AGGREGATION	'Y'로 설정하면 TPR AGGREGATION 기능을 활성화한다. (기본값: N)
ACTIVE_SESSION_HISTORY	'Y'로 설정하면 세션 상태 저장 기능 활성화한다. (기본값: N)
_ACTIVE_SESSION_HISTO RY_SAMPLING_INTERVAL	세션 상태 저장 주기를 설정한다. (기본값: 1, 단위: 초)

17.2.2. 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
_TPR_SNAPSHOT	저장된 스냅샷의 ID와 시간에 관한 정보를 관리한다.
_TPR_BASELINE	등록된 Baseline의 정보를 관리한다.
_TPR_ACTIVE_SES SION_HISTORY	저장된 ASH Sample 정보를 관리한다.
_TPR_METRIC	저장된 TPR Metric 정보를 관리한다.
_TPR_JCNTSTAT	_VT_JCNTSTAT 뷰의 스냅샷 정보를 관리한다.
_TPR_SQLSTATS	V$SQLSTATS 뷰의 스냅샷 정보를 관리한다.
_TPR_SQL_PLAN	V$SQL_PLAN 뷰의 스냅샷 정보를 관리한다.

테이블	설명
_TPR_SQL_PLAN_STAT	V$SQL_PLAN_STATISTICS 뷰의 스냅샷 정보를 관리한다.
_TPR_LATCH	V$LATCH 뷰의 스냅샷 정보를 관리한다.
_TPR_SYSTEM_EVENT	V$SYSTEM_EVENT 뷰의 스냅샷 정보를 관리한다.
_TPR_WAITSTAT	V$WAITSTAT 뷰의 스냅샷 정보를 관리한다.
_TPR_SGASTAT	V$SGASTAT 뷰의 스냅샷 정보를 관리한다.
_TPR_PGASTAT	V$PGASTAT 뷰의 스냅샷 정보를 관리한다.
_TPR_LIBRARYCACHE	V$LIBRARYCACHE 뷰의 스냅샷 정보를 관리한다.
_TPR_SQLTEXT	V$SQLTEXT 뷰의 스냅샷 정보를 관리한다.
_TPR_FILESTAT	V$FILESTAT 뷰의 스냅샷 정보를 관리한다.
_TPR_SEGMENTSTAT	V$SEGMENT_STATISTICS 뷰의 스냅샷 정보를 관리한다.
_TPR_TEMPSEG_OP_USAGE	V$TEMPSEG_OP_USAGE 뷰의 스냅샷 정보를 관리한다.
_TPR_PROCESS	V$PROCESS 뷰의 스냅샷 정보를 관리한다.
_TPR_SESSION	V$SESSION 뷰의 스냅샷 정보를 관리한다.
_TPR_WAITER_SESSION	V$WAITER_SESSION 뷰의 스냅샷 정보를 관리한다.
_TPR_UNDOSTAT	V$UNDOSTAT 뷰의 스냅샷 정보를 관리한다.
_TPR_OSSTAT2	V$OSSTAT2 뷰의 스냅샷 정보를 관리한다.
_TPR_SQLWA_HIST	V$SQLWA_HIST 뷰의 스냅샷 정보를 관리한다.
_TPR_MODIFIED_PARAM	_VT_PARAMETER 테이블의 스냅샷 정보를 관리한다.
_TPR_MISC	세션 수와 같은 기타 정보의 스냅샷 정보를 관리한다.

다음은 저장된 스냅샷의 ID와 시간에 관한 정보를 관리하는 '_TPR_SNAPSHOT' 테이블 정보이다.

```
TABLE '_TPR_SNAPSHOT'
-------------------------------------------------------------
COLUMN_NAME                      TYPE            CONSTRAINT
-------------------------------- --------------- ----------
SNAP_ID                          NUMBER
THREAD#                          NUMBER
INSTANCE_NUMBER                  NUMBER
BEGIN_INTERVAL_TIME              DATE
END_INTERVAL_TIME                DATE
SNAP_GID                         NUMBER
```

- 세션 상태 저장 기능

  세션 상태 저장 주기에 활동 중인 세션들의 정보를 저장한다.

테이블	설명
_TPR_ACTIVE_SESSION_HISTORY	활동 중인 세션 상태 정보를 관리한다.
V$ACTIVE_SESSION_HISTORY	최근 1시간 동안의 세션 상태 정보를 관리한다.

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
THREAD#                        NUMBER
SAMPLE_TIME                    DATE
SID                            NUMBER
SESS_SERIAL_NO                 NUMBER
USER_NO                        NUMBER
WAIT_EVENT                     NUMBER
TIME_WAITED                    NUMBER
SQL_ID                         VARCHAR(13)
SQL_CHILD_NUMBER               NUMBER
CURR_HASHVAL                   NUMBER
MODULE_NAME                    VARCHAR(64)
ACTION_NAME                    VARCHAR(64)
CLIENT_INFO_NAME               VARCHAR(64)
PROG_NAME                      VARCHAR(30)
SQL_EXEC_START                 DATE
SQL_EXEC_ID                    NUMBER
SQL_PLAN_LINE_ID               NUMBER
ID1                            NUMBER
ID2                            NUMBER
WE_SEQ                         NUMBER
USGMT_ID                       NUMBER
SLOTNO                         NUMBER
WRAPNO                         NUMBER
PORT                           NUMBER
DELTA_TIME                     NUMBER
DELTA_PHY_READ_BLKS            NUMBER
DELTA_PHY_WRITE_BLKS           NUMBER
DELTA_LOG_READ_BLKS            NUMBER
PGA_SIZE                       NUMBER
WAIT_OBJ_ID                    NUMBER
WAIT_FILE_NO                   NUMBER
```

```
WAIT_BLOCK_NO                   NUMBER
WAIT_ROW_NO                     NUMBER
```

17.2.3. 수동 스냅샷 생성 기능

TPR을 설정하면 정해진 주기에 자동으로 스냅샷이 생성되지만 원하는 경우 현재 시점의 스냅샷을 남길 수 있다.

```
SQL> exec dbms_tpr.create_snapshot();
```

17.2.4. 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

_TPR_SNAPSHOT 테이블을 조회하여 원하는 기간에 대한 시작, 종료 시각을 확인한다.

다음은 원하는 기간의 시작과 종료 시점을 각각 begin_date, end_date라고 설정한 경우 tbsql에서 다음과 같이 입력하여 성능 분석 리포트를 작성하는 예이다. _TPR_SNAPSHOT 테이블의 BEGIN_INTERVAL_TIME이 begin_date과 end_date 사이에 포함되는 모든 스냅샷들이 리포트로 출력된다.

```
/* exec dbms_tpr.report_text(begin_date, end_date) */
```

```
SQL> exec dbms_tpr.report_text('2013-01-01 13:00:00', '2013-01-01 14:59:00');
```

성능 분석 리포트는 다음의 경로에 파일로 생성된다.

```
$TB_HOME/instance/$TB_SID/tpr_report.{mthr_pid}.{current_time}
```

성능 분석 항목

다음은 성능 분석 항목에 대한 설명이다.

- Overview Part
  - System Overview
    - CPU Usage
    - Memory Usage
  - Workload Overview

        - Workload Summary
        - Workload Stats
    - Instance Overview
        - Instance Efficiency
        - TAC Statistics Overview (Cluster Cache Activity, Cluster Buffer Cache, Cluster Cache and Wait Lock Statistics)
        - Top 5 Wait Events by Wait Time
        - I/O Overview
    - SQL Overview
        - PGA Work Area Statistics
        - Top 3 SQL Ordered by Elapsed Time
        - Top 3 SQL Ordered by Executions
        - Top 3 SQL Ordered by Gets
- Detail Part
    - System Detail
        - OS Statistics
        - Shared Pool Statistics
        - Physical Plan Cache Statistics
        - Data Dictionary Cache Statistics
        - PGA Statistics
    - Workload Detail
        - Workload Stats (Time-based)
        - Workload Stats (Number-based)
        - Workload Stats (Size-based)
    - Instance Detail
        - Buffer Cache Statistics
        - Wait Event Summary (by Class)
        - Wait Events by Wait Time
        - Session Status with Wait Event

    - Blocking Session Status with Wait Event
    - Wlock Statistics
    - Spinlock(Latch) Statistics
    - Spinlock(Latch) Sleep Statistics
    - Tablespace I/O Statistics
    - File I/O Statistics
    - Temp Segment Usage Statistics
    - Segments Ordered by Physical Reads ("TPR_SEGMENT_STATISTICS=Y"로 설정할 경우)
    - Segments Ordered by Logical Reads ("TPR_SEGMENT_STATISTICS=Y"로 설정할 경우)
    - Segments Ordered by ITL Waits ("TPR_SEGMENT_STATISTICS=Y"로 설정할 경우)
    - Segments Ordered by Buffer Busy Waits ("TPR_SEGMENT_STATISTICS=Y"로 설정할 경우)
    - Segments Ordered by Row Lock Waits ("TPR_SEGMENT_STATISTICS=Y"로 설정할 경우)
    - Undo Statistics
    - Wait Statistics ("_DB_BLOCK_PIN_WAIT_USE_STAT=Y"로 설정할 경우)
  - SQL Detail
    - PGA Summary
    - PGA Work Area Histogram
    - SQL Ordered by Elapsed Time (with Physical Plan)
    - SQL Ordered by Elapsed Time/Execution (with Physical Plan)
    - SQL Ordered by Executions (with Physical Plan)
    - SQL Ordered by Gets (with Physical Plan)
    - SQL Ordered by Reads (with Physical Plan)
    - SQL Ordered by Extra I/O (with Physical Plan)
    - SQL Ordered by CPU (with Physical Plan)
- Etc
  - Tibero Init. Parameters(.tip)
  - Modified Parameters

제18장 Tibero Recovery Catalog

본 장에서는 Tibero Recovery Catalog의 구성요소와 동작 및 운영 방법을 설명한다.

18.1. 개요

Tibero Recovery Catalog는 다수의 데이터베이스들의 메타데이터(metadata)를 관리를 목적으로 제공하는 기능이다.

Tibero Recovery Catalog는 물리적으로 독립된 장소에 여러 데이터베이스들의 메타데이터를 보관한다. 메타데이터를 보관하는 데이터베이스를 **Catalog 데이터베이스**(이하 Catalog)라고 한다. 관리하려는 데이터베이스를 Catalog에 등록하면 tbrmgr이 등록하려는 데이터베이스의 컨트롤 파일(Control file)을 읽어 메타데이터를 추출하고, Catalog에 이 데이터를 원격으로 보내어 저장한다. Catalog에 최신 데이터를 업데이트할 수 있다. 그 외에 RMGR 클라이언트를 이용하여 백업을 생성하거나 지우는 경우 관리자가 직접 resync 요청을 보내는 경우에도 메타데이터를 업데이트한다.

저장하는 메타데이터는 다음과 같다.

- 데이터베이스 아이디, 데이터베이스 이름과 같은 데이터베이스의 기본 정보
- 테이블 스페이스, 데이터 파일, Redo 로그와 같은 데이터베이스 구조
- 데이터베이스의 체크포인트
- 데이터베이스 및 아카이브 로그 백업

Catalog를 이용하면 다음과 같은 이점을 얻을 수 있다.

- 컨트롤 파일 이외의 장소에 메타데이터를 저장할 수 있어 정보의 다원화가 가능하다.
- 하나의 데이터베이스에 여러 데이터베이스의 메타데이터를 중앙화하여 관리할 수 있다.
- 컨트롤 파일에서는 언젠간 사라질 가능성이 있는 아카이브 로그, 로그 히스토리, 백업, 백업 아카이브 로그 정보들을 Catalog에서는 더 길게 보관하는 것이 가능하다.
- 컨트롤 파일을 재생성하거나 백업을 이용해 복구하였을 시 유실되는 정보들을 Catalog를 이용해서 복구하는 것이 가능하다.

18.2. 구성 요소

Tibero Recovery Catalog는 Catalog Database에 관리가 필요한 다른 데이터베이스들의 메타데이터를 보관하는 구조를 가진다. 데이터베이스를 Catalog로 사용하기 위해서는 catalog create가 필요하다.

Create할 때 다음을 생성한다.

- 테이블 스페이스 RECOVERY_CATALOG

  데이터베이스들의 메타데이터를 저장하기 위한 테이블 스페이스이다.

- 시퀀스 SITE_KEY_SEQ

  등록된 데이터베이스들을 구분하기 위한 고유값 SITE_KEY 생성을 위한 시퀀스이다.

- 데이터 파일 recovery_catalog.dtf

  메타데이터를 저장하기 위한 데이터 파일이다. 100MB의 크기로 생성되며 autoextent 10MB가 설정되어 있다. Single 데이터베이스 기준 매일 한번 백업을 진행한다고 하였을 때 1년에 약 10MB 정도의 용량을 사용하게 되므로 이를 참고하여 저장공간을 설정하면 된다.

RECOVERY_CATALOG 테이블 스페이스에는 메타데이터 저장을 위한 테이블을 가지고 있으며 RC_SITE_KEY 테이블을 제외한 나머지는 DBID_KEY, SITE_KEY, DBINC_KEY를 Primary Key로 한다. 사용자는 각 KEY 값을 이용하여 Catalog에서 관리할 데이터베이스의 메타데이터 정보를 열람할 수 있다.

Primary Key	**설명**
DBID_KEY	데이터베이스의 ID이다. 각 데이터베이스의 DBID와 대응하며 V$DATABASE에서 확인 가능하다.
SITE_KEY	데이터베이스의 고유 KEY 값이다. Primary와 Standby는 같은 DBID를 가지기에 이 들을 구분할 수 있는 KEY 값인 SITE_KEY를 가진다. 이 값은 SITE_KEY_SEQ를 통해 생성된다.
DBINC_KEY	데이터베이스의 Incarnation이다(RESETLOGS 부트를 한 시점 정보). 각 데이터베이스의 RESETLOGS_TSN과 대응한다. DBID를 가지기에 이 들을 구분할 수 있는 KEY 값인 SITE_KEY를 가진다. 이 값은 SITE_KEY_SEQ를 통해 생성된다.

메타데이터를 저장하기 위해 생성한 테이블의 종류는 아래와 같다.

테이블 이름	**설명**
RC_LAST_RESYNC	등록된 데이터베이스가 Catalog에 데이터를 업데이트 할때의 정보를 기록하는 테이블이다.
RC_DATABASE	등록된 데이터베이스의 기본 정보를 기록하는 테이블이다. 컨트롤 파일의 Database Entry section과 대응한다.
RC_DB_INCARNATION	데이터베이스의 INCARNATION 정보를 기록하는 테이블이다.
RC_SITE_KEY	데이터베이스의 DB_UNIQUE_NAME에 대응하는 고유한 열쇠값인 SITE_KEY를 기록하는 테이블이다. 이 테이블은 Incarnation의 구분이 필요하지 않기에 DBINC_KEY를 가지지 않는다.
RC_REDO_THREAD	데이터베이스의 Redo Thread에 대한 정보를 기록하는 테이블이다. 컨트롤 파일의 Redo Thread section과 대응한다.

테이블 이름	설명
RC_FLASHBACK_THREAD	데이터베이스의 Flashback Thread에 대한 정보를 기록하는 테이블이다. 컨트롤 파일의 Flashback Thread section과 대응한다.
RC_STANDBY_REDO_THREAD	데이터베이스의 Standby Redo Thread에 대한 정보를 기록하는 테이블이다. 이 테이블은 스탠바이 데이터베이스에서만 업데이트가 진행이 되며, 컨트롤 파일의 Standby Redo Thread section과 대응한다.
RC_LOGFILE	데이터베이스의 Redo 로그 파일에 대한 정보를 기록하는 테이블이다. 컨트롤 파일의 Log File section과 대응한다.
RC_TABLESPACE	데이터베이스의 테이블 스페이스에 대한 정보를 기록하는 테이블이다. 컨트롤 파일의 Tablespace section과 대응한다.
RC_DATAFILE	데이터베이스의 데이터 파일에 대한 정보를 기록하는 테이블이다. 컨트롤 파일의 Datafile section과 대응한다.
RC_ARCHIVED_LOG	데이터베이스의 아카이브 로그에 대한 정보를 기록하는 테이블이다. 컨트롤 파일의 Archived logs section과 대응한다.
RC_LOG_HISTORY	데이터베이스의 로그 스위치 기록에 대한 정보를 기록하는 테이블이다. 컨트롤 파일의 Log History section과 대응한다.
RC_BACKUP_SET	데이터베이스의 백업 셋에 대한 정보를 기록하는 테이블이다. RC_BACKUP_SET은 같은 DB_NAME을 공유하는 데이터베이스들이 모두 공유한다. 컨트롤 파일의 Backup Set section과 대응한다.
RC_BACKUP_ARCHIVELOG	데이터베이스의 백업 아카이브 로그에 대한 정보를 기록하는 테이블이다. RC_BACKUP_ARCHIVELOG는 같은 DB_NAME을 공유하는 데이터베이스들이 모두 공유한다. 컨트롤 파일의 Backup Archivelog section과 대응한다.

Tibero Recovery Catalog를 구성하는 기본적인 단계는 다음과 같다.

1. Catalog를 생성한다.

   Catalog에 필요한 오브젝트들을 생성하는 단계이다. 자세한 사항은 “18.3.1. Catalog Create”를 참고한다.

2. 관리하고자 하는 데이터베이스를 Catalog에 등록한다.

   등록을 통해 Catalog는 해당 데이터베이스의 메타데이터를 저장한다. 자세한 사항은 “18.3.2. Catalog Register”를 참고한다.

3. 필요에 따라 Resync를 진행하여 데이터베이스의 메타데이터를 업데이트한다.

   메타데이터의 업데이트는 사용자가 RMGR 클라이언트를 통해 수동으로 진행된다. 자세한 사항은 “18.3.4. Catalog Resync”를 참고한다.

18.3. 기능

Tibero Recovery Catalog는 RMGR 클라이언트를 이용하여 Catalog를 관리할 수 있다.

Catalog를 관리하기 위한 커맨드는 tbrmgr catalog이며, Catalog와 관련된 모든 기능은 이 커맨드를 통해 동작해야 한다. tbrmgr catalog는 기존 tbrmgr과 옵션을 공유하지 않으며, catalog 옵션은 아래와 같다.

catalog 옵션	**설명**
create	--cat-userid, RMGR_CATALOG_SID 파라미터로 지정한 데이터베이스를 Catalog로 이용하기 위해 필요한 작업을 실행한다.
register	데이터베이스를 Catalog에 등록하는 작업을 실행한다.
unregister	데이터베이스를 Catalog에서 등록 취소하는 작업을 실행한다.
resync	Catalog에 등록된 메타데이터를 현재 데이터베이스의 메타데이터로 업데이트하는 작업을 실행한다.
re-register	컨트롤 파일 재생성과 같이 기존의 컨트롤 파일이 유실된 데이터베이스를 Catalog에 다시 등록하는 작업을 실행한다.

--userid	데이터베이스에 접속할 사용자명과 패스워드 및 SID를 아래와 같은 형식으로 지정한다.
	`--userid USERID[[/PASSWD][@SID]]`
	화면상에 비밀번호 노출을 원치 않을 때에는 비밀번호를 공백으로 입력한 후 비밀번호를 입력하라는 문구가 나오면 비공개로 번호를 입력할 수 있다.
	`--userid USERID/[@SID]`
	OS 인증을 받은 계정으로 로그인하는 경우 Userid와 Passwd를 입력하지 않아도 로그인 가능하다
	`--userid /`

--cat-userid	Catalog에 접속할 사용자명과 패스워드 및 SID를 지정한다. --userid와 같은방식으로 지정하며 RMGR_CATALOG_SID를 지정했을 경우 SID 생략이 가능하다. SID와 RMGR_CATALOG_SID를 모두 지정했다면 --cat-userid에서 지정한 SID가 적용된다.
-h, --help	RMGR Catalog의 옵션 사용법을 출력한다.
-l, --log-level	클라이언트 측 RMGR 로그(tbrmgr_trace.log)의 기록 레벨을 설정한다. 레벨은 1부터 5까지 있으며 숫자가 커질수록 더 자세히 기록된다. 기본 레벨은 4이다.
-L	tbrmgr_trace.log의 기록될 위치를 지정한다.

catalog 옵션	설명
	기본 경로는 $TB_HOME/client/tbrmgr_log/이다.
--all-incarnation	Unregister할 때 모든 incarnation의 메타데이터 정보를 같이 제거한다.
--backup-set	백업 셋과 백업 아카이브 로그의 메타데이터에 대해서만 Resync를 진행한다.
--archivelog	아카이브 로그의 메타데이터에 대해서만 Resync를 진행한다.
--log-history	로그 히스토리의 메타데이터에 대해서만 Resync를 진행한다.

참고

Tibero Recovery Catalog의 모든 기능을 사용하기 위해 클라이언트는 반드시 Catalog에 연결되어야 한다. 따라서 tbrmgr catalog를 사용할 때는 반드시 --cat-userid를 [예 18.1]와 같이 설정해야 한다.

Catalog에 접속하기 위해선 tbdsn.tbr 파일에 설정이 필요하다.

[예 18.1] Catalog SID 설정 및 연결

```
cat=(
  (INSTANCE=(HOST=123.1.2.3)
            (PORT=12345)
            (DB_NAME=cat)
  )
)
```

tbrmgr에서 Catalog에 접속하려면 위에서 설정한 SID를 tip 파일의 RMGR_CATALOG_SID 또는 option 중의 하나인 --cat-userid를 설정해야 한다. 두 방법 중에는 --cat-userid를 이용한 설정이 우선 적용된다.

```
# tip을 이용한 sid 설정
RMGR_CATALOG_SID=cat

# rmgr option을 이용한 sid 설정
tbrmgr catalog create --cat-userid <userid>/<passwd>@cat
```

18.3.1. Catalog Create

Create는 RMGR_CATALOG_SID나 --cat-userid로 설정된 데이터베이스를 Catalog로써 사용할 수 있도록 필요한 작업을 실행한다.

```
tbrmgr catalog create --cat-userid <userid>/<passwd>@cat
```

위와 같이 작업이 완료되면 Catalog는 메타데이터 관리에 필요한 테이블 스페이스, 데이터 파일 및 테이블들을 생성하게 된다. 생성되는 것은 “18.2. 구성 요소”를 참고한다.

18.3.2. Catalog Register

특정 데이터베이스를 관리하기 위해선 해당 데이터베이스를 Catalog에 등록해야하며 이 과정을 Reigster(등록)라고 한다. 다음과 같이 등록을 진행할 수 있다.

```
tbrmgr catalog register --userid <userid>/<passwd>@<SID>
--cat-userid <userid>/<passwd>@cat
```

RMGR 클라이언트는 데이터베이스의 메타데이터들을 컨트롤 파일에서 직접 읽어서 Catalog에 등록한다. 등록을 완료하면 데이터베이스도 자신이 등록되었음을 컨트롤 파일에 기록한다.

이 때 Catalog는 등록된 데이터베이스들을 DB_UNIQUE_NAME 값으로 구분한다. DB_UNIQUE_NAME은 각 데이터베이스를 구분하기 위한 파라미터로 미설정 시 DB_NAME과 같은 값을 가지게 된다.

참고

만약 등록할 데이터베이스의 DB_UNIQUE_NAME이 이미 등록되어 있는 데이터베이스와 같다면 현재의 등록은 실패하게 된다. 관리자는 이 점을 유의하여 각 데이터베이스의 DB_UNIQUE_NAME을 다르게 설정해야 한다.

또한 Catalog는 데이터베이스를 Incarnation(RESETLOGS 부트를 한 시점 정보)마다 별개로 저장한다. 만약 데이터베이스를 RESETLOGS 부트한다면 기존과 다른 데이터베이스로 취급되며 Catalog 등록정보는 컨트롤 파일에서 사라지게 된다. Catalog는 기존 Incarnation의 메타데이터를 지우지는 않으나, 새 Incarnation의 데이터베이스가 다시 Catalog를 사용하기 위해선 register를 다시 진행해야 한다.

등록된 데이터베이스는 백업 및 복구 과정을 Catalog가 관리하게 되는데, 그로 인해 RMGR을 사용할 때 다음과 같은 부분의 변화가 생긴다.

- Catalog에 연결하지 않으면 RMGR Backup / RMGR Delete 기능을 사용할 수 없다.

  Catalog에 등록된 데이터베이스는 백업의 관리 주체가 자신에서 Catalog로 변한다. Catalog는 매 백업 때마다 적절한 Backup Set ID를 부여하고 한 DB_NAME을 가진 데이터베이스 내에 동일한 ID를 가진 백업이 없도록 관리한다. 등록된 데이터베이스는 Backup Set ID를 발급할 수 없기에 Catalog 없이는 RMGR 백업을 진행할 수 없고, 백업을 지우는 작업 역시 Catalog에서 관리되어야 하므로 Catalog 없이 진행이 불가하다.

- TSC 환경에서 다른 노드가 진행한 백업을 이용해 복구를 진행할 수 있다.

  TSC 환경의 Primary, Standby 모두 Catalog에 등록되어 있다면, 서로의 데이터베이스에서 각자 진행한 백업을 이용하여 복구를 진행할 수 있다. 이를 위해선 복구과정에서도 --cat-userid를 통해 Catalog에 연결이 필요하다. 복구과정에 최신 백업 셋이나 지정한 (-b|--backup-set)이 다른 데이터베이스에서 진행되어 컨트롤 파일에 존재하지 않는다면, Catalog에서 복구에 필요한 백업 셋들의 정보 자동으로 컨트롤 파일에 추가한 뒤에 복구 과정을 진행한다.

참고

백업 파일까지 보내주지는 않기에 데이터베이스가 해당 백업 파일에 접근이 가능한 경우에만 사용 가능하다.

Standby에서 백업을 진행하면 백업 셋 정보는 Catalog에 남으며, Primary에서도 Catalog를 통해 확인이 가능하다. 최신의 백업 셋이 Standby에서 진행한 2번 백업이라고 가정하자. 이때 Primary가 Catalog에 연결한채로 복구를 진행하면, 클라이언트는 Catalog에서 자동으로 2번 백업 셋 정보를 가져와 복구를 진행한다.

[예 18.2] Primary를 Standby의 백업을 이용하여 복구 진행

```
$ tbrmgr recover --cat-userid sys/1234@cat -o /disk1/backup
================================================================================
=   Recovery Manager(RMGR) starts                                              =
=                                                                              =
=   TmaxData Corporation Copyright (c) 2008-. All rights reserved.    =
================================================================================
RMGR '-o' option not used
: restoring from the paths actually backed up previously

================================================================================
    RMGR - recovery (COMPLETE)
================================================================================
Shutting down the instance...

Tibero instance terminated (ABNORMAL mode).

info file is deleted.
 unlink failed.: No such file or directory

Control file #0 (/home/tibero/c1.ctl) is accessible

All control files are accessible. No need to restore the backup control file.


Booting up the instance...

Listener port = 6001

Tibero 7

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

Last backup set id: 2
The backup set does not exist in the control file.
```

```
Copy backup set(s) infos from the recovery catalog to the control file.

Initializing the restore progress, it may take few minutes...

RMGR begins restoring backup files.
Full backup set_id: 2
Last incremental backup set_id: 2

...
```

18.3.3. Catalog Unregister

데이터베이스의 메타데이터를 catalog에서 제거하는 과정을 진행하는 기능이다.

```
tbrmgr catalog unregister --userid <userid>/<passwd>@<SID>
--cat-userid <userid>/<passwd>@cat
```

unregister의 진행은 데이터베이스의 현재 Incarnation 정보에 대해서만 진행된다. 만약 데이터베이스의 모든 정보를 삭제할 --all-incarnation 옵션을 추가해야 한다.

18.3.4. Catalog Resync

Catalog의 메타데이터를 현재 데이터베이스의 메타데이터와 동기화하는 작업을 진행한다.

```
tbrmgr catalog resync --userid <userid>/<passwd>@<SID>
--cat-userid <userid>/<passwd>@cat
```

별도로 옵션을 주지 않았을 경우 모든 메타데이터에 대해 동기화가 진행되는 Full Resync가 진행되며, Resync 옵션 중에 하나를 주었을 경우엔 지정한 섹션의 메타데이터만 동기화하는 Partial Resync가 진행된다.

백업 셋 및 백업 아카이브 로그을 제외한 나머지 메타데이터는 사용자가 직접 Resync 요청을 보내야만 진행된다. 백업 셋 및 백업 아카이브 로그는 다른 데이터베이스와 공유하는 메타데이터이기에 RMGR Backup / Delete 등이 진행될 때도 자동으로 수행된다.

18.3.5. Catalog Re-register

재생성이나 백업에서 복원하는 등의 상황에선 컨트롤 파일에 Catalog 등록에 대한 정보가 사라져 Catalog 기능을 사용할 수 없는 상태가 된다. re-register는 이와 같은 상황에서 데이터베이스가 다시 등록된 상태와 같게 해주는 작업을 진행한다.

```
tbrmgr catalog re-register --userid <userid>/<passwd>@<SID>
--cat-userid <userid>/<passwd>@cat
```

Appendix A. tbdsn.tbr

tbdsn.tbr 파일은 클라이언트가 Tibero의 데이터베이스에 접속하기 위해 필요한 정보를 가지고 있는 환경 설정 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.tbr 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

TBCLI_LOG_DIR=/home/test/log

...
```

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 언더라인(_)이며, 대소문자를 구별한다.

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 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 캐릭터 셋을 지정할 수 있다. TB_NLS_LANG의 기본값은 다음과 같다. – Tibero 6 이전 : MSWIN949 – Tibero 7 이후 : UTF8 (예: TB_NLS_LANG=UTF8)
TBCLI_LOG_LVL	CLI의 로그 레벨을 지정할 수 있다. (예: TBCLI_LOG_LVL=TRACE)
TBCLI_LOG_DIR	CLI의 로그를 저장할 디렉터리를 지정할 수 있다. 디폴트 디렉터리는 Windows 계열이 c:\ 이고, UNIX 계열은 /tmp이다. (예: TBCLI_LOG_DIR=/home/test/log)

A.2. 이중화 서버 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. 이중화 서버로 설정된 SID에 대해서는 항상 CTF(Connection Time Failover)를 지원한다.

```
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 밸런싱 설정

Tibero에서는 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 'Y'로 설정하면 로드 밸런싱 기능을 사용할 수 있다. 반면에 LOAD_BALANCE 값이 없거나 'Y' 외의 다른 값을 입력하면 로드 밸런싱 기능은 동작하지 않는다.

A.4. Failover 설정

Tibero가 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)
    (FORCE_FAILOVER_DELAY=10)
)
```

위 예제에서 보듯이 SID 내의 USE_FAILOVER 값을 'Y'로 설정하면 Failover 기능을 사용할 수 있다. 단, 문장(Statement) 레벨까지의 복구는 지원하지 않는다.

그리고 FORCE_FAILOVER_DELAY는 Failover 기능의 선택적인 옵션으로 다른 인스턴스 또는 이중화된 서버로 접속하기 전에 일정 지연시간(단위 초) 뒤 접속을 시도하는 기능이다. 위의 예제에서는 10초 후에 다른 인스턴스로 Failover가 시도된다.

참고

1. CTF(Connection Time Failover)는 USE_FAILOVER 설정과는 관련이 없으며 이중화 서버로 구성한 SID에 대해서는 항상 지원한다.

2. USE_FAILOVER 설정으로 서버에 접속하려고 할 때 서버가 normal 모드로 기동한 상태가 아니라면 오류(ERROR_CLIMSG_REDIRECT)가 발생한다. 서버가 normal 모드로 기동한 상태가 아니라면 명확하게 접속하려고 하는 서버를 명시해야 한다.

Appendix B. V$SYSSTAT

동적 뷰인 **V$SYSSTAT**를 조회하면 시스템의 각종 통계 정보를 조회할 수 있다. DBA는 조회된 항목을 통해 Tibero 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 block을 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 LGWR에 flush를 요청한 횟수

항목	설명
log flush	LGWR가 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 로그 버퍼에 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] shadow에서 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 prefetch할 때 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 cancel을 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 RSB 등이 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	TAC용의 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	TAC용의 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 context에 대한 cache hit의 횟수
ex workarea cache miss count	쿼리 실행에 필요한 private context에 대한 cache miss의 횟수
ex cache miss count for different param types	쿼리 실행에 필요한 private context에 대한 cache miss 중 바인드 파라미터의 타입이 달라서 생긴 횟수

Appendix C. 문제 해결

본 장에서는 Tibero를 사용할 때 발생할 수 있는 문제를 해결하는 방법을 설명한다.

C.1. 데이터베이스 접속

- 문제

```
$ tbsql SYS/tibero

tbSQL 7

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

TBR-2131: Generic I/O error
```

tbSQL 유틸리티를 이용하여 Tibero의 데이터베이스에 접속할 때 이러한 메시지가 출력되는 문제이다.

- 해결

Tibero가 기동되어 있는지 확인한다. **Tibero의 기동 여부에 따라 이를 해결하는 방법이 다르다.**

– Tibero가 기동되지 않은 경우

tbctl 명령어를 사용하여 Tibero의 프로세스가 실행되고 있는지 확인한다.

```
$ tbctl pid

  ...Tibero 프로세스가 없다.
```

또는 ps와 grep 명령어로 Tibero의 프로세스가 실행되고 있는지 확인한다.

```
$ ps -ef | grep tbsvr
  tibero   22919 22590  0 15:37 pts/10   00:00:00 grep tbsvr

  ...Tibero 프로세스가 없다.
```

Tibero 프로세스가 없다면 **tbboot** 명령어를 사용하여 Tibero를 기동한다. tbboot 명령어를 사용하는 방법에 대한 내용은 “2.5.1. tbboot”를 참고한다.

– Tibero가 기동되어 있는 경우

다음과 같이 Tibero 프로세스는 실행되고 있으나, 데이터베이스에 접속되지 않는 경우는 대체로 서버와 클라이언트의 환경설정이 서로 맞지 않을 때 발생한다.

```
$ ps -ef | grep tbsvr
tibero   32036 1      0 20:00 pts/7    00:00:00 tbsvr         -t ...
tibero   32038 32036  0 20:00 pts/7    00:00:00 tbsvr_MGWP    -t ...
tibero   32039 32036  0 20:00 pts/7    00:00:00 tbsvr_FGWP000   -t ...
tibero   32040 32036  0 20:00 pts/7    00:00:00 tbsvr_FGWP001   -t ...
tibero   32041 32036  0 20:00 pts/7    00:00:00 tbsvr_PEWP000   -t ...
tibero   32042 32036  0 20:00 pts/7    00:00:00 tbsvr_PEWP001    -t ...
tibero   32043 32036  0 20:00 pts/7    00:00:00 tbsvr_PEWP002    -t ...
tibero   32044 32036  0 20:00 pts/7    00:00:00 tbsvr_PEWP003    -t ...
tibero   32045 32036  0 20:00 pts/7    00:00:00 tbsvr_AGNT    -t ...
tibero   32046 32036  0 20:00 pts/7    00:00:00 tbsvr_DBWR    -t ...
tibero   32047 32036  0 20:00 pts/7    00:00:00 tbsvr_RCWP    -t ...
tibero   32057 30048  0 20:00 pts/7    00:00:00 grep tbsvr
```

$TB_HOME/client/config 디렉터리에 있는 **tbdsn.tbr** 파일을 확인하여 클라이언트의 설정 부분을 확인한다.

<<tbdsn.tbr>>

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

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 초기화 파라미터
#
LISTENER_PORT=6666
......
```

$TB_SID.tip 파일을 보면 클라이언트에 설정된 포트 번호와 일치하지 않다는 것을 확인할 수 있다. 이로 인해 Tibero의 데이터베이스에 접속할 수 없는 원인이 된다.

포트 번호가 설정되지 않으면 Tibero 내부에 테스트 용인 임의의 포트 번호가 설정된다. 그러면 포트 번호의 충돌 등의 문제를 유발할 수 있다. **이러한 경우 Tibero의 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 Tibero는 기동된다. 반면에 원격으로 접속할 때에는 반드시 서버 쪽의 포트 번호와 일치해야 Tibero가 기동된다.

Appendix D. 클라이언트 환경변수

다음은 클라이언트에서 설정할 수 있는 환경변수에 대한 설명이다.

환경변수	설명
TB_NLS_LANG	클라이언트의 캐릭터 셋이다. 자세한 내용은 "Tibero 설치 안내서"의 "Appendix C. Tibero 지원 문자 집합"을 참고한다.
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 – RR/MM/DD HH24:MISSXFF(기본값)
TB_NLS_NCHAR	클라이언트의 국가별 캐릭터 셋이다. 다음의 값 중에 하나를 선택한다. – JA16SJISFIXED
TBCLI_LOG_LVL	tbCLI의 로그 레벨을 설정한다. 다음의 값 중에 하나를 설정한다. (기본값: DEBUG) – FATAL: 심각한 오류로 인해 응용 프로그램의 중단이 필요한 경우 사용한다. – ERROR: 응용 프로그램 작업이 계속 수행된다. – WARN: 오류는 아니지만, 문제가 될 소지가 있다.

환경변수	설명
	– 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) – TRACE: 매우 자세한 정보, DEBUG보다 자세한 로깅 정보를 제공한다. (예: 정형화되지 않은 이진 데이터 덤프, 메모리 덤프 등 과 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. 로그파일이 생성되는 위치는 다음과 같다. – WINDOWS: C:\tbxa_*날짜시간*.log – UNIX: /tmp/tbxa_*날짜시간*.log 설정값은 TBCLI_LOG_LVL와 동일하다.
TB_DSN_FILE	DSN 파일의 위치를 설정한다. 서버없이 클라이언트만 설치되어 TB_HOME 환경변수를 통해서 DSN 파일의 위치를 참조할 수 없을 때 사용한다. TB_HOME이 함께 설정된 경우, TB_DSN_FILE이 우선 적용된다.

색인

Symbols

A

B

C

D

Q

R

S

T

U

V

W

X

ㄱ

ㄴ

ㄷ

ㄹ

ㅁ

ㅂ

ㅅ

ㅇ

표

ㅎ