Tibero Database Administration

Contents

TmaxTibero

0. 교육소개

교육과정

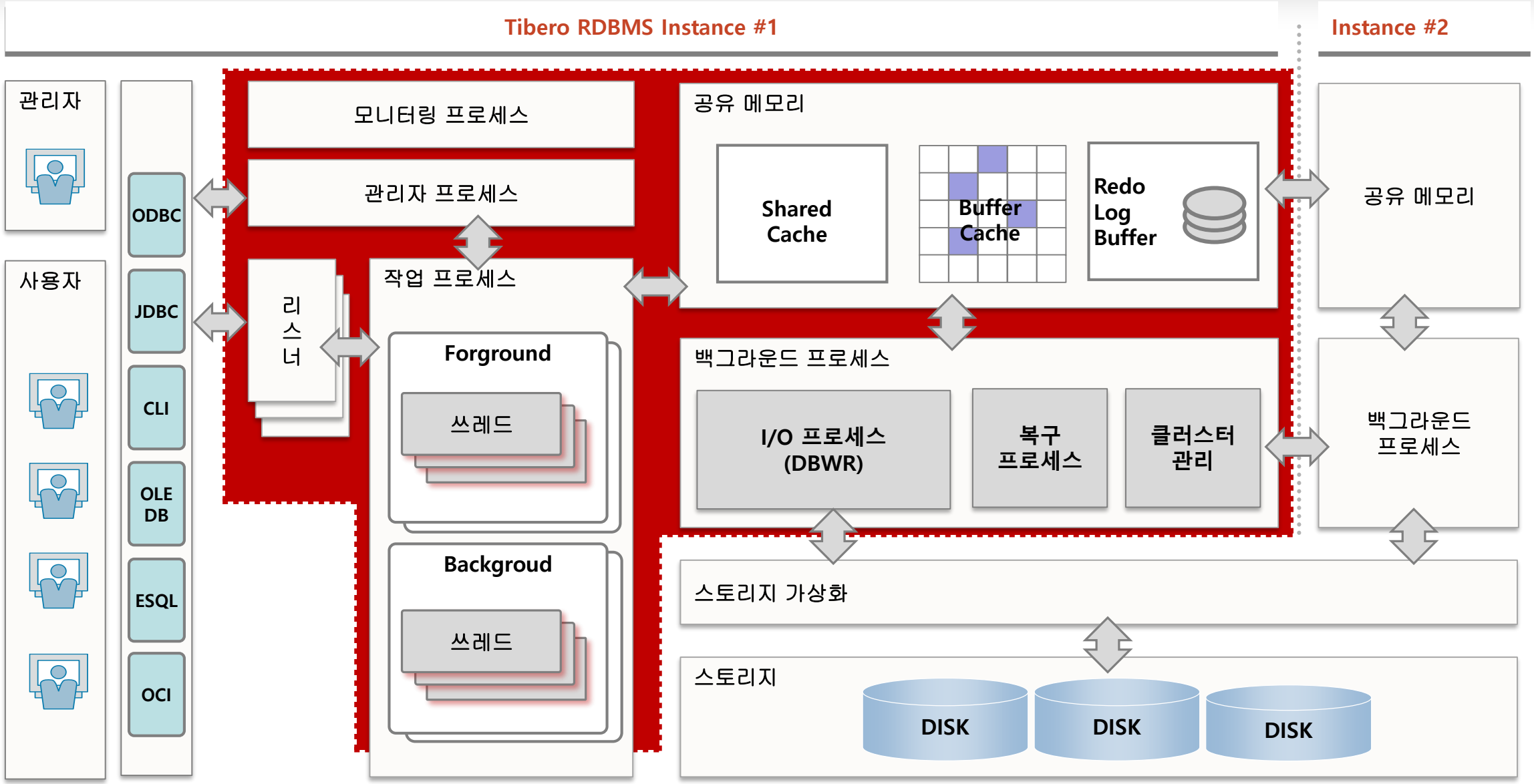
과정	내용
Tibero Database Administration	**티베로 관리자를 위한 교육** • 교육기간 : 3일 • 교육대상 : Tibero 관리자

TmaxTibero

1. 데이터베이스 유지관리

- Tibero 구조
- Tibero 기동 환경
- tbSQL 유틸리티
- Tibero 기동 종료
- Parameter File
- Controlfile 관리
- Redo Log 관리
- Tablespace 와 Datafile 관리

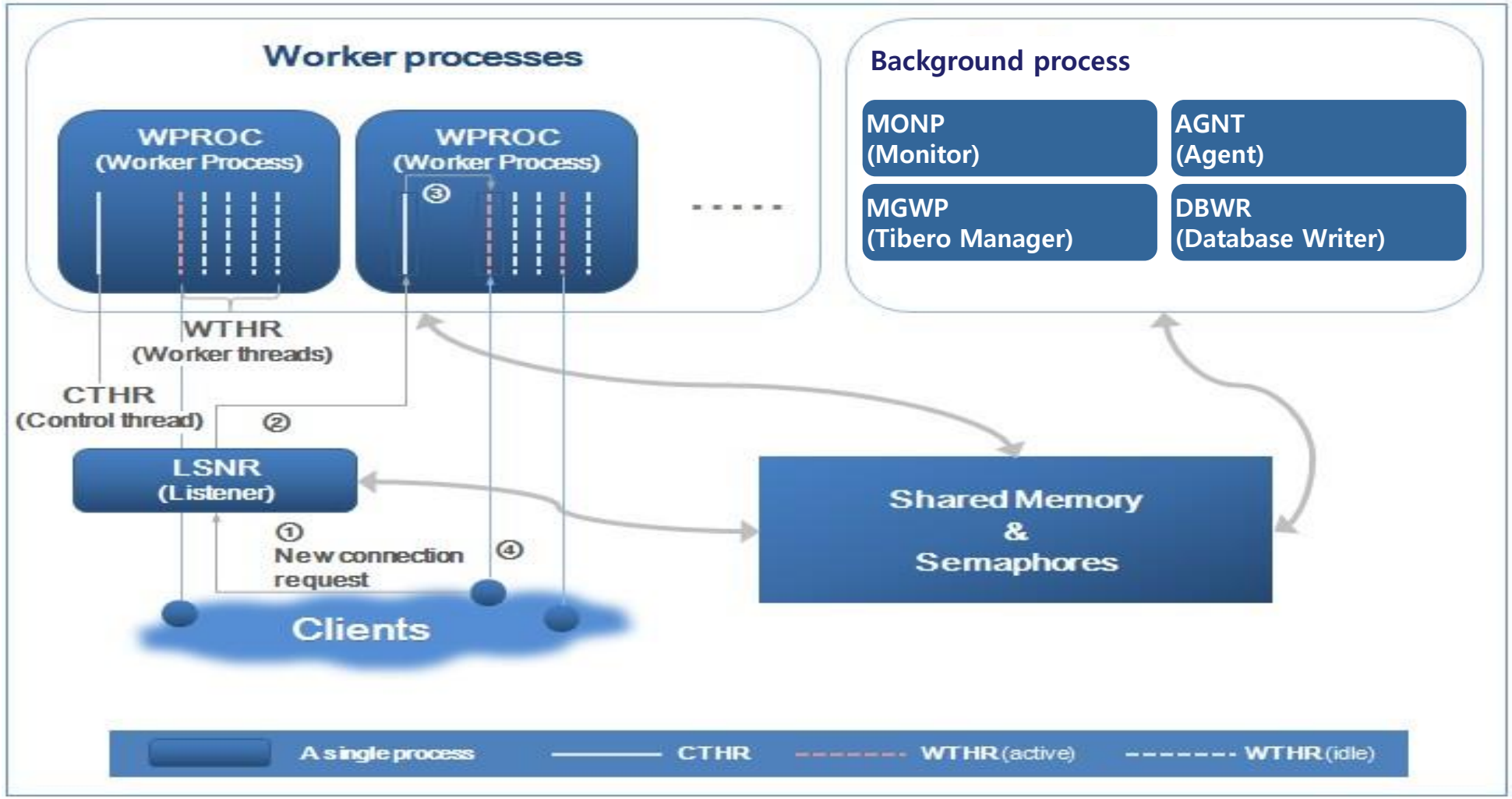
Tibero 구조



TmaxTibero

Tibero 구조 - Process

- Tibero 프로세스 구조
  - 대규모 사용자 접속을 수용하는 다중 프로세스 및 다중 스레드 기반의 아키텍쳐 구조
    - Listener, 워커프로세스 (Working Process 또는 Foregroud Process) , Background Process



TmaxTibero

Tibero 구조 - Process

- **Listener**
  - 클라이언트의 새로운 접속 요청을 받아 이를 유휴한 워커 프로세스에 할당
  - 클라이언트와 워커 프로세스간에 중계 역할을 담당하며, 별도의 실행 파일인 tblistener를 사용하여 작업
  - 모니터링 프로세스에 의해서 생성되며 외부에서 강제 종료하더라도 자동으로 재 시작 됨
  - Listener Port

Listener port
< $TB_SID.tip > LISTENER_PORT=8629

Extra listener port	
정적	< $TB_SID.tip > LISTENER_PORT=8629 EXTRA_LISTENER_PORTS=8639;8640;
동적	alter system listener add port 8799; alter system listener delete port 8799;

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

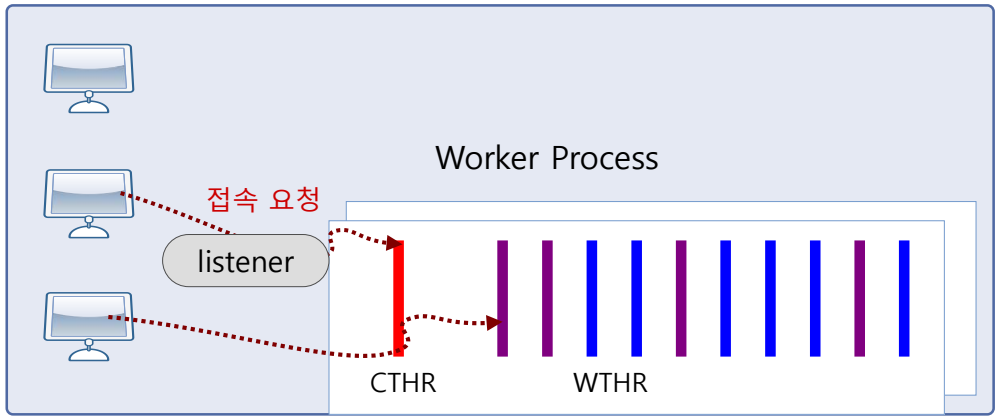
Tibero 구조 - Process

- 워커프로세스 (Worker Processes 또는 Foregroud Process)

  - 클라이언트와 실제 통신을 하며, 사용자 요구 사항을 처리 하는 프로세스
  - Foregroud Worker Process는 리스너를 통해 들어온 온라인 요청 처리
  - Backgroud Worker Process는 Internal Task나 Job Scheduler에 등록된 배치 작업을 수행

  - CTHR (Control Thread)
    - 각 Working Process마다 하나씩 생성. 서버 시작 시에 지정된 개수의 Worker Thread를 생성.
    - 시그널 처리 담당.
    - I/O Multiplexing을 지원하며, 필요한 경우 워커 스레드 대신 메시지 송/수신 역할 수행.

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



Tibero 구조 - Process

- **Background Processes**

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

  - 독립된 프로세스로서, 사용자의 요청과 비동기적으로 동작.

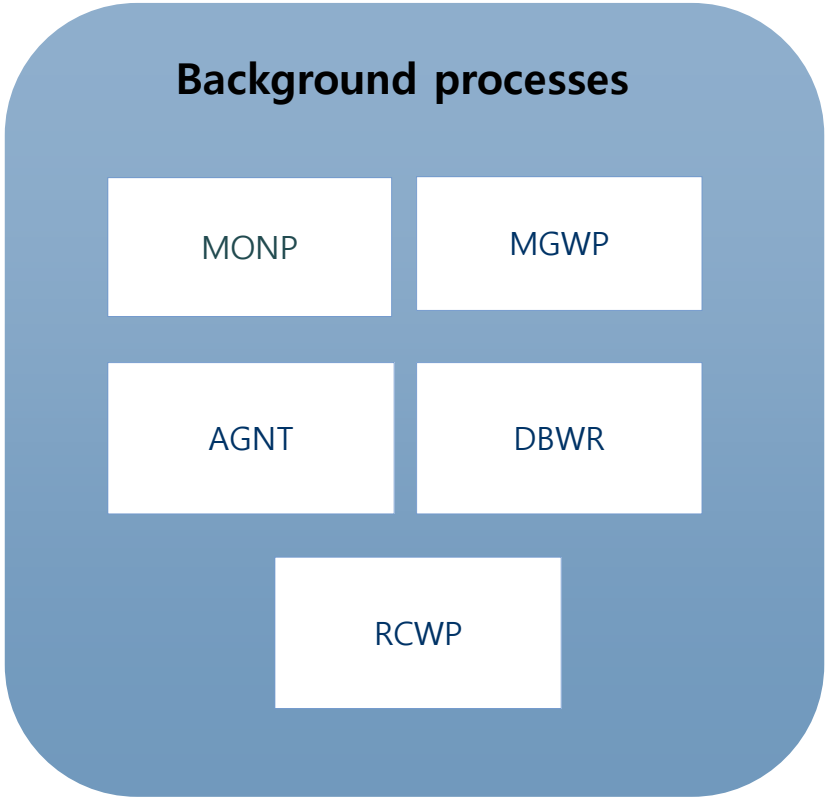
  - 감시 프로세스(MONP: monitor process)

  - 매니저 프로세스(MGWP: manager worker process)

  - 에이전트 프로세스(AGNT : agent process)

  - 데이터베이스 쓰기 프로세스(DBWR : database writer)

  - 복구 프로세스(RCWP : recover worker process)



TmaxTibero

Tibero 구조 - Process

Background Processes

감시 프로세스(MONP: monitor process)

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

매니저 프로세스(MGWP)

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

Tibero 구조 - Process

Background Processes

에이전트 프로세스(AGNT : agent process)

- 시스템 유지를 위해 주기적으로 처리해야 하는 Tibero 내부의 작업을 담당
  - Internal Task나 Batch Job이 언제 수행되어야 하는지 판단은 AGENT 프로세스가 담당 하지만, 실제 수행은 포어그라운드 혹은 백그라운드 워커 프로세스에게 의뢰하는 구조임
- 다중 스레드(multi-threaded) 기반 구조로 동작하며, 서로 다른 용도의 업무를 스레드별로 나누어 수행

데이터베이스 쓰기 프로세스(DBWR)

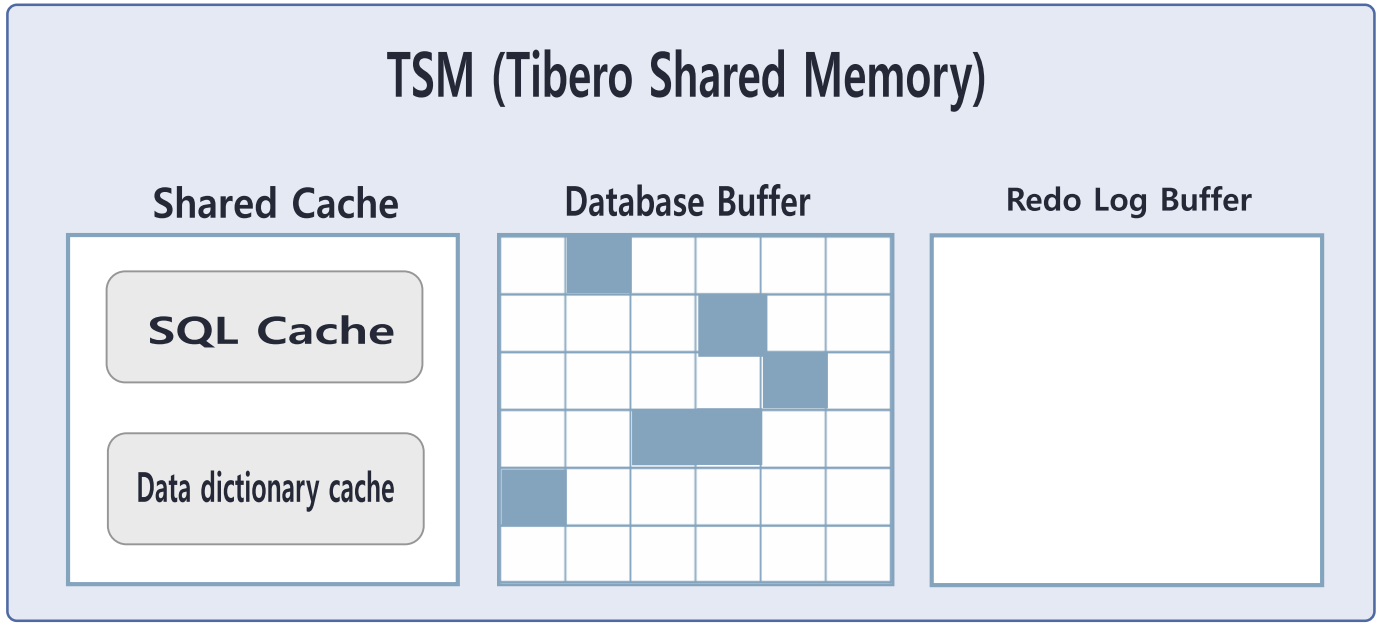
- 데이터베이스에서 변경된 내용을 디스크에 기록하는 일과 연관된 스레드들이 모여 있는 프로세스
- 사용자가 변경한 블록을 디스크에 주기적으로 기록하는 스레드
- 리두 로그를 디스크에 기록하는 스레드
- 두 스레드를 통해 데이터베이스의 체크포인트 과정을 관할하는 체크포인트 스레드

복구 프로세스(REWP)

- 복구 전용 프로세스
- Crash / Instance Recovery 수행

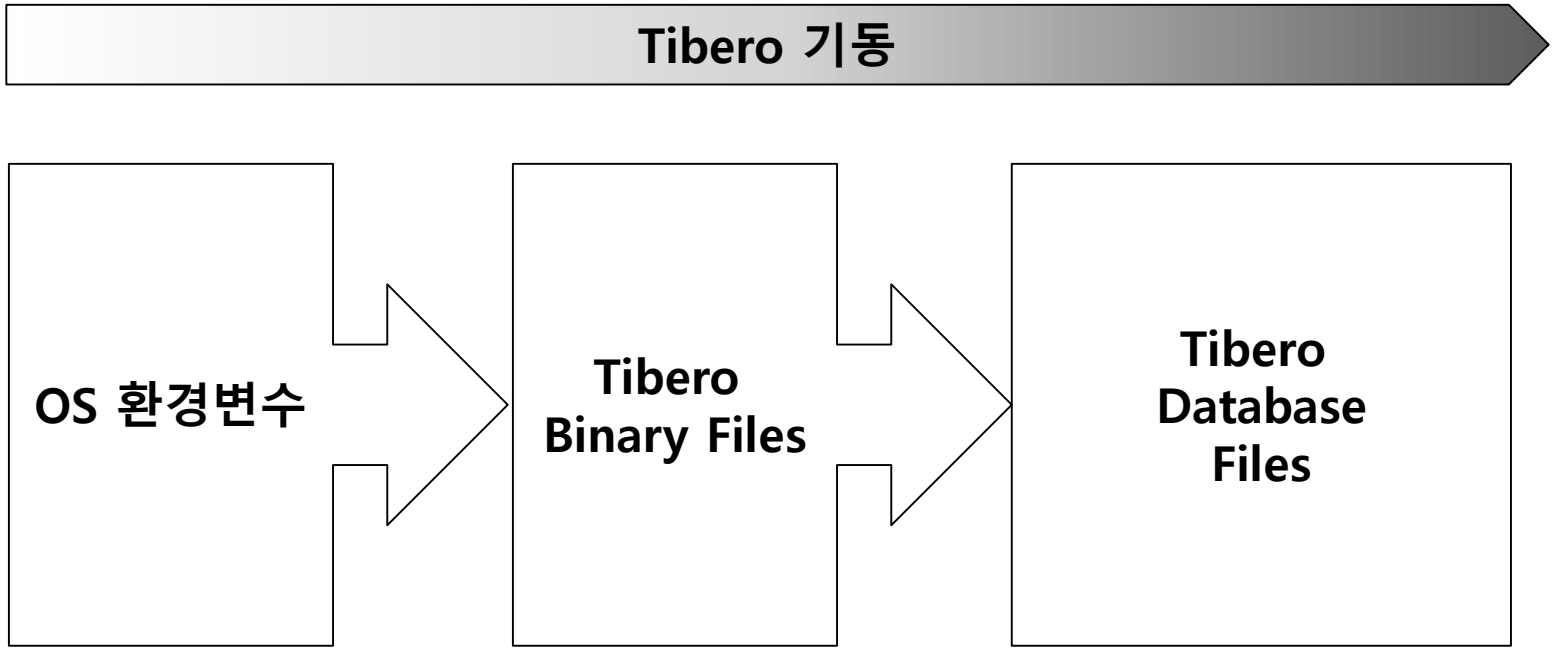
Tibero 구조 - Memory

- Tibero Shared Memory (TSM)
  - 인스턴스에 대한 데이터와 제어 정보를 가지는 공유 메모리 영역
  - 사용자가 동시에 데이터를 공유
  - Database Buffer, Redo Log Buffer, SQL Cache, Data Dictionary Cache 로 구성됨.
  - Backgroud Process는 인스턴스가 시작될 때 TSM 영역을 할당하고, 인스턴스가 종료하면 할당 해제
  - TSM의 전체 크기는 인스턴스가 시작될 때 생성되어 고정.



Tibero 기동 환경

Tibero 기동을 위해서는 OS 커널파라미터 및 환경변수 그리고 Tibero 바이너리, 데이터베이스 파일이 정상적으로 설정 및 설치가 되어 있어야 함



Tibero 기동 환경

Tibero 는 OS 환경변수를 참조하여 구동 되므로, 적절한 값이 설정되어 있어야 한다

환경변수	설명
TB_HOME	Tibero가 설치된 디렉터리이다. **예: /tibero/tibero6**
TB_SID	Tibero 인스턴스 식별자 이며, 파라미터 파일의 이름, 각종 로그 생성 경로 등에 영향을 준다 **예: tibero**
LD_LIBRARY_PATH	Tibero를 사용할 때 필요한 공유 라이브러리가 위치한 경로이다. 필요한 라이브러리는 모두 $TB_HOME/lib와 $TB_HOME/client/lib 안에 있고, OS별로 환경변수가 다르게 지정된다. SunOS, Linux : LD_LIBRARY_PATH HP-UX : SHLIB_PATH AIX : LIBPATH
PATH	Tibero 의 실행파일 디렉터리 경로이다. OS 기본 $PATH 에 $TB_HOME/bin 와 $TB_HOME/client/bin 을 추가하여 설정한다 **예: /tibero/tibero6/bin , /tibero/tibero6/client/bin**

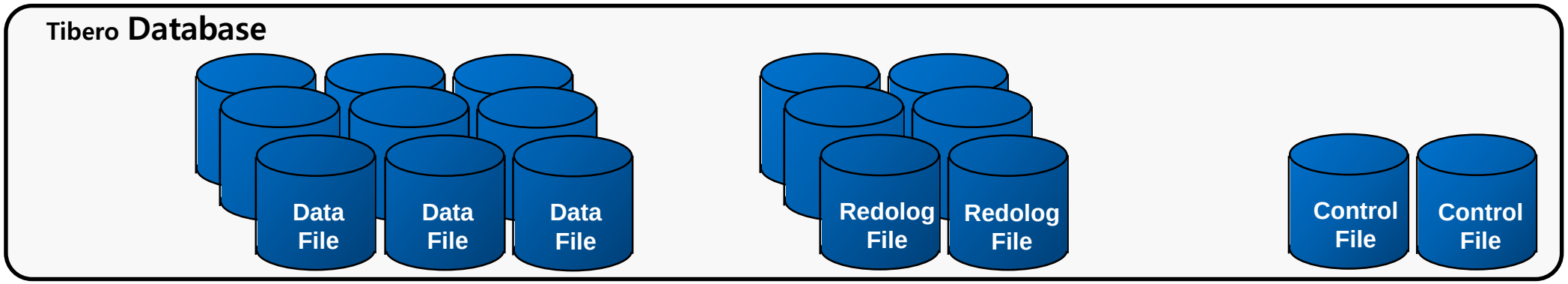
Tibero 기동 환경

Tibero 서버는 다음과 같은 디렉터리로 구성되며, 구동을 위해서는 Tibero 바이너리 파일이 정상적으로 설치되어 있어야 한다

디렉토리	설명
bin	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.
client	클라이언트 실행 파일이 있는 디렉터리이다. 이 디렉터리에는 다음과 같은 유틸리티가 있다.(tbSQL, T-Up, tbExport, tbImport, tbLoader, tbpc)
config	환경설정 파일이 위치하는 디렉터리이다. 이 위치에 존재하는 $TB_SID.tip 파일이 Tibero의 환경설정을 결정한다.
database	데이터베이스 저장 디렉터리 기본 경로 (데이터베이스 정보를 별도로 설정하지 않는 한 모든 데이터베이스 정보가 이 디렉터리와 그 하위 디렉터리에 저장된다)
instance	인스턴스 동작시 발생하는 각종 로그를 기록하는 디렉터리이다
lib	공간(Spatial) 쿼리와 관련된 함수를 사용하기 위한 라이브러리 파일이 있는 디렉터리이다
license	라이선스 파일(license.xml)이 있는 디렉터리이다. XML 형식이므로 일반 텍스트 편집기로도 라이선스의 내용을 확인할 수 있다.
nls	시간대 파일이 있는 디렉터리이다
scripts	데이터베이스를 생성할 때 사용하는 각종 SQL script 가 있는 디렉터리이다

Tibero 기동 환경

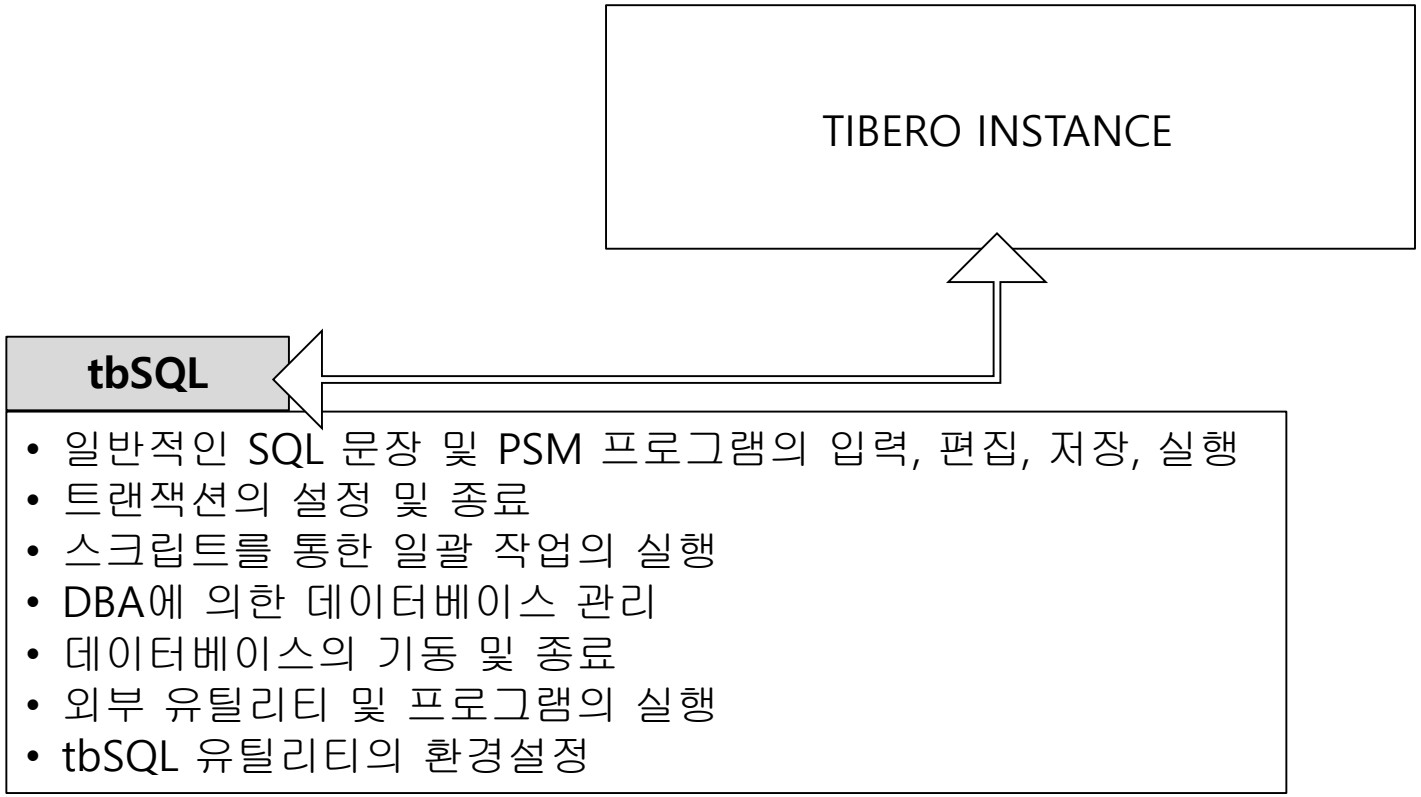
Tibero 서버 기동을 위해서는 데이터베이스 파일이 정상적으로 생성되어 있어야 한다



- Datafiles
  - Tables & Indexes 들과 같은 데이터 저장 객체를 물리적으로 저장하고 있는 파일.
- Redo Log Files
  - 복구를 위하여 Database에서 변경된 모든 것을 기록 하는 파일.
- Control Files
  - Database의 물리적 구조와 상태를 기록 하는 파일.

tbSQL 유틸리티

Tibero 관리를 위해서 사용하는 기본 도구로서 tbSQL 를 제공하고 있다. tbSQL은 TEXT 모드의 대화형 SQL 명령어 처리를 지원하는 유틸리티이다. SQL 질의, 데이터 정의어, 트랜잭션과 관련된 SQL 문장 등을 실행할 수 있다.



Tibero 기동 종료

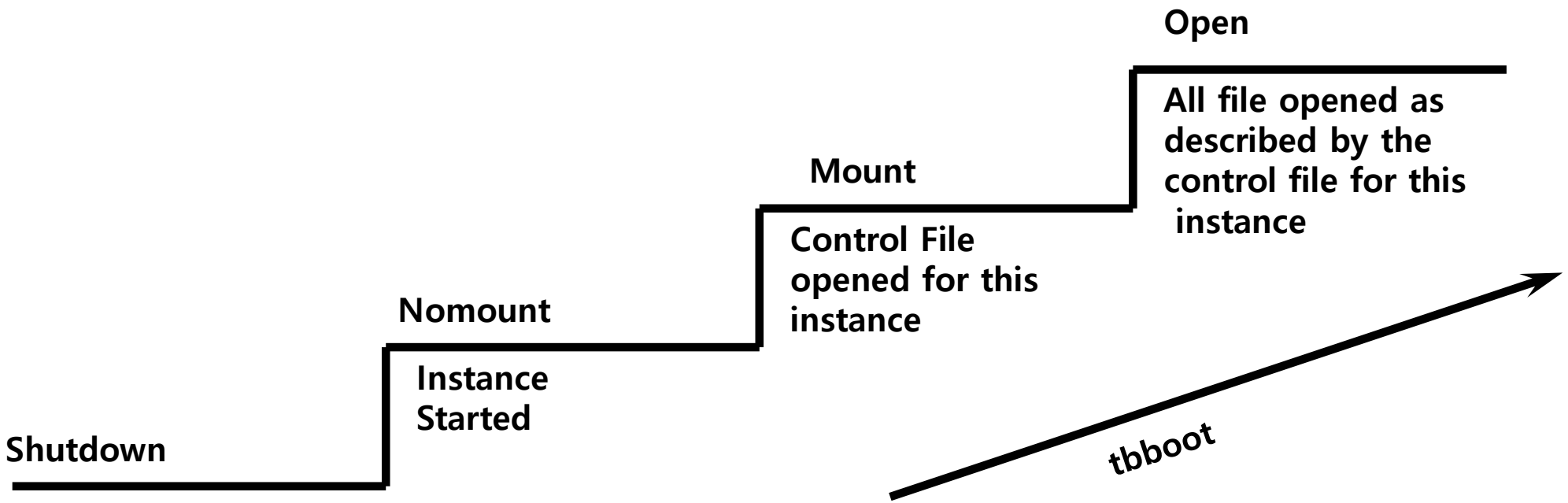
tbboot

- tbboot는 Tibero가 설치된 머신에서 실행해야 한다.
- tbboot를 실행하기 전에 반드시 환경변수가 제대로 설정되어 있어야 한다.
- 이 명령어와 관련된 환경변수는 $TB_HOME과 $TB_SID이다.
- tbboot는 실행 파일을 실행할 수 있는 권한이 있는 사용자라면 어느 누구든 Tibero를 기동할 수 있으므로, Tibero를 설치한 사용자만 Tibero의 실행 파일에 접근할 수 있도록 권한을 설정할 것을 권장한다.

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.

Tibero 기동 종료

Tibero Instance 기동



TmaxTibero

Tibero 기동 종료

Tibero Instance 종료

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

옵션	설명
	옵션이 없는 경우 Tibero를 정상 모드로 종료하는 옵션.
-h	tbdown 을 사용하기 위한 간단한 도움말을 보여주는 옵션.
-t	Tibero 서버를 종료할 수 있는 옵션이다. 이 옵션은 생략이 가능.

다운 모드	설명
NORMAL	모든 세션이 끝날 때까지 기다린 후 종료하는 모드.
POST_TX	모든 트랜잭션이 끝날 때까지 기다린 후 Tibero 를 종료하는 모드.
IMMEDIATE	현재 수행 중인 모든 작업을 강제로 중단시키며, 진행중인 트랜잭션을 롤백하고 Tibero 를 종료하는 모드.
ABORT	Tibero 의 프로세스를 강제로 종료하는 모드.

Tibero 기동 종료

Tibero Instance 종료 과정

- **Database Close**
  - TSM에 있는 모든 Database Data와 Recovery Data를 Data File과 Redo Log File에 각각 기록.
- **Dismounting Database**
  - Control File을 Close.
- **Instance Shutdown**
  - TSM에 있는 Memory가 제거되고, Process가 종료.
  - Abnormal Shutdown이 되었을 경우 TSM에 Memory가 상주해 있거나 Process가 종료되지 않을 수가 있다. 따라서 남아 있는 이전 Instance를 제거하거나, tbdown abort 명령어를 이용하여 새로운 Instance를 강제로 종료하고, tbdown clean.

Tibero 기동 종료

- **tbdown abort**
  - Tibero의 프로세스를 강제로 종료하는 모드.
  - ABORT는 Tibero에 SYS 사용자로 접속한 다음 Tibero의 MTHR 프로세스가 모든 프로세스를 OS의 강제 종료 시그널을 전달하여 강제로 종료시키는 모드. 따라서 이 모드는 비상 시에 사용하며. 다음 번에 TiberoRDBMS를 기동할 때 파손 복구 과정이 필요.
  - ABORT는 Tibero가 강제로 종료시키므로 사용하던 시스템 리소스를 해제할 기회 미존재. 따라서 서버가 종료된 다음에도 공유 메모리(Shared Memory), 세마포어(Semaphore) 등의 시스템 리소스가 남아있을 수 있으며, 로그 파일이나 데이터 파일도 동일.
  - tbdown abort 이후에 Tibero 재 기동 시 파손복구에 많은 시간이 소요 가능성 존재.
  - ABORT 는 다음과 같은 상황에서만 사용하도록 권장.

✓Tibero 의 내부 에러로 인한 정상적인 종료가 불가능한 경우.
(예를 들어, 모든 세션이 사용 중인 상태에서 클라이언트의 접속이 증가하는 경우.)

✓H/W에 문제가 발생하여 Tibero 를 즉시 종료해야 하는 경우.

✓해킹 등의 비상 상태가 발생하여 Tibero 를 즉시 종료해야 하는 경우.

Tibero 기동 종료

- **tbdown clean**

옵션	설명
clean	Tibero 서버가 비정상 종료된 상태에서 운영중에 사용하였던 공유 메모리나 세마포어 자원들을 해제하는 옵션. Tibero 서버가 운영 중일 때는 사용 할 수 없는 옵션.

- **kill 명령으로 종료한 경우, tbdown clean 를 실행.**
  - ➢ 만약 Tibero 서버가 Kill 과 같은 시스템 내부 명령어에 의해서 비정상적으로 종료된 경우, 운영 중에 사용하였던 공유 메모리나 세마포어 자원들이 해제가 않 될 수 있다. 이런 경우 Tibero 가 정상적으로 재 기동 하지않게 되므로, tbdown clean 으로 기존 자원을 해제시킨 후
    tbboot 를 실행.

BOOT_WITH_AUTO_DOWN_CLEAN 파라미터

비정상적으로 종료되더라도 초기화 파라미터 BOOT_WITH_AUTO_DOWN_CLEAN를 Y로 설정하면 자동으로 이전 운영 중에 사용하였던 자원을 해제시켜 부팅 가능.

관리자가 서버의 비정상 종료 상황을 제대로 인지하지 못하고 서버 운영을 할 수 있으며, 기존 자원이나 프로세스가 제대로 정리가 안 되는 예외적이 상황이 발생하여 충돌이 날 수 있으므로 BOOT_WITH_AUTO_DOWN_CLEAN 옵션을 켜는 것을 비 권장.

Tibero 기동 종료

- **tbdown POST_TX**
  - ➢ 모든 트랜잭션이 끝날 때까지 기다리고 나서 Tibero를 종료하는 모드.
  - ➢ POST_TX는 Tibero에 SYS 사용자로 접속한 다음 모든 트랜잭션이 끝날 때까지 기다림.
    그 다음 Tibero RDBMS를 종료.
  - ➢ tbdown 실행이 시작되면 더는 데이터베이스에 접속할 수 없고, 이미 열려 있던 세션에서도 새로운 트랜잭션을 시작할 수 없음. 다만, 현재 수행 중인 트랜잭션은 Commit 혹은 Rollback 할 때까지 제한 없이 수행할 수 있으며, Commit이나 Rollback을 하는 순간 자동으로 데이터베이스 접속을 종료.
  - ➢ tbdown 실행이 시작되면 데이터베이스에 접속한 클라이언트에게 서버 종료를 알리는 메시지를 보내지 않음
  - ➢ tbSQL 유틸리티 등에서는 클라이언트가 서버 종료를 즉시 알지 못하고, 그 다음 명령을 실행할 때
    비로소 Tibero RDBMS가 종료되었음을 알리도 됨

Tibero 기동 종료

- **tbdown POST_TX**
  - 사용 예제

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

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

SQL> SELECT * FROM T1;
      COL1
----------
        10

1 row selected.

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

SQL> COMMIT;
Commit succeeded.

        ...이 시점에서 실제로 서버가 종료되고 tbdown 실행이 끝난다.

SQL> SELECT * FROM T1;
TBR-2069: I/O read error

        ...서버가 종료되었으므로 tbSQL 유틸리티의 프롬프트에서는 TBR-2069 에러가 발생된다.
```

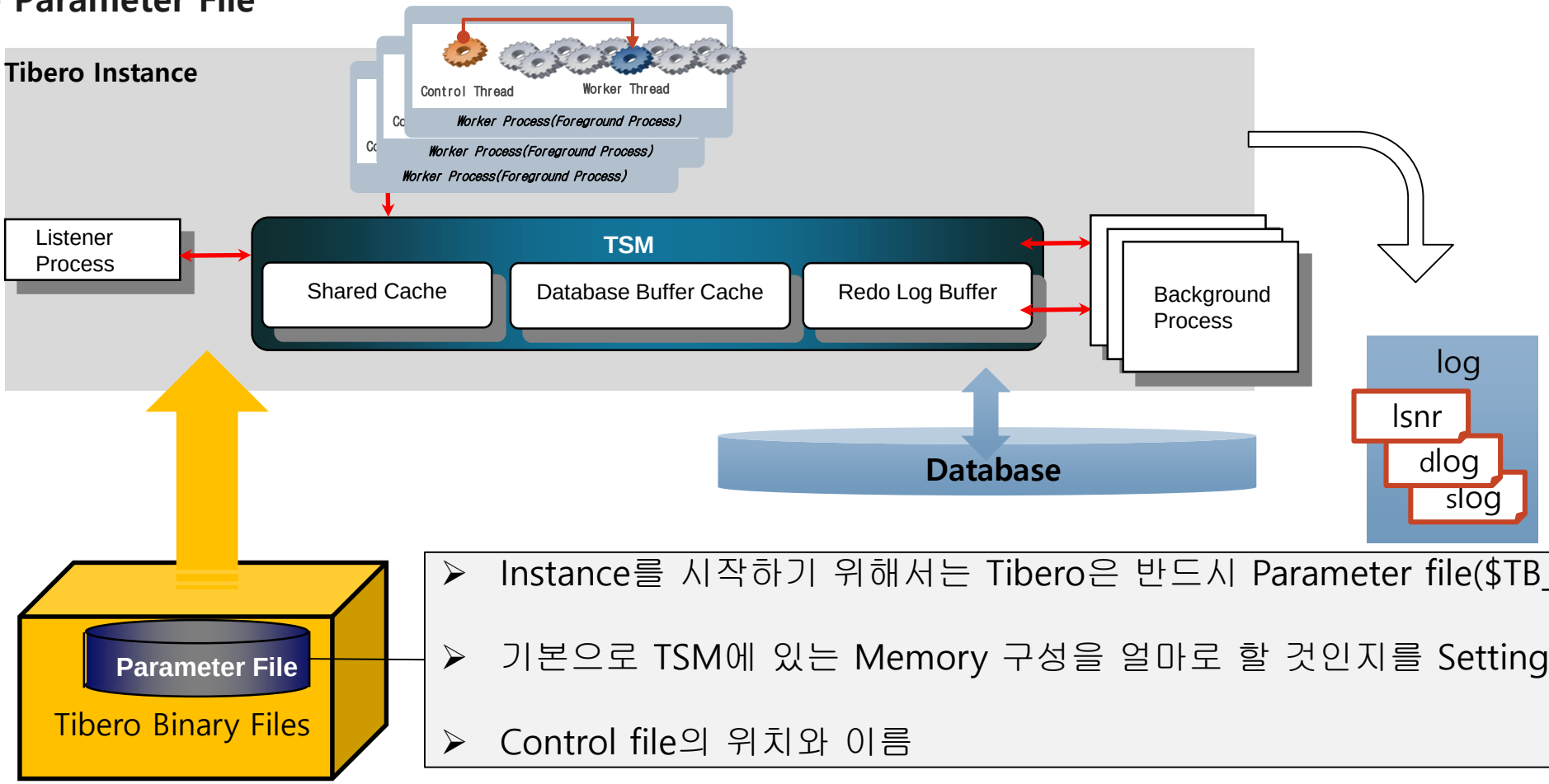
Tibero 기동 종료

- **tbdown IMMEDIATE**
  - 현재 수행 중인 모든 작업을 강제로 중단시키며, 진행 중인 모든 트랜잭션을 롤백하고 Tibero를 종료하는 모드.
  - IMMEDIATE는 Tibero에 SYS 사용자로 접속한 다음 현재 수행 중인 모든 작업을 강제로 종료하고, 진행 중이던 모든 트랜잭션을 롤백한 후, Tibero를 종료. 클라이언트에서 Tibero 종료를 알지 못하는 것은 POST_TX 모드와 동일.
  - 트랜잭션이 오래 걸리는 작업 중에 있었다면, 이를 모두 롤백하기 위해서 다소 시간이 걸림.
  - 사용 예

```
$ tbdown immediate

Tibero instance terminated (IMMEDIATE mode).
```

Parameter File

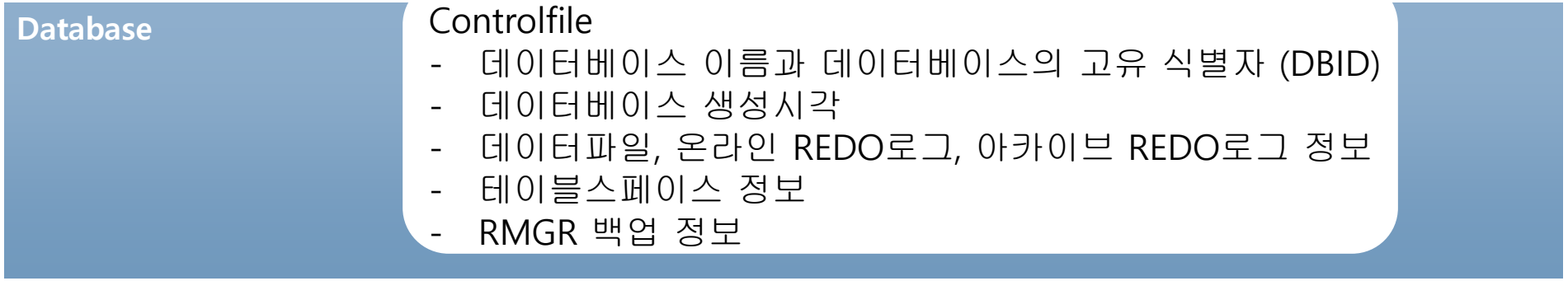
- **The Parameter File**



- Instance를 시작하기 위해서는 Tibero은 반드시 Parameter file($TB_SID.tip)을 Read.
- 기본으로 TSM에 있는 Memory 구성을 얼마로 할 것인지를 Setting 필요.
- Control file의 위치와 이름

Controlfile 관리

- **Control Files**



- Control File은 작은 Binary File로 Database의 구조.
- 모든 Database Files & log files들은 Control file에서 식별 가능.
- Database name은 Control file에 저장.
- Control File은 Mount, Open시에 필요.
- Recovery시 필요한 동기화된 정보를 저장.
- 최소 다른 Disk상에 2개 이상의 Control Files를 가지도록 권장.
- Control File은 Database가 Open되어 졌을 때 반드시 Tibero Server가 Writing 가능하도록 설정.

TmaxTibero

Controlfile 관리

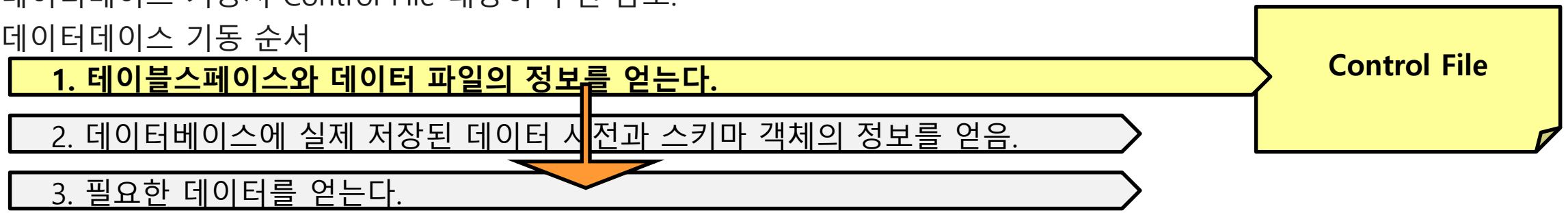
- **Control File 생성 지침**
  - **제어 파일 이름 지정**
    - Parameter file의 CONTROL_FILES에서 지정
    - CONTROL_FILES는 쉼표로 구분된 하나이상의 파일 이름 명시.
    - 지정된 모든 파일에 데이터베이스 작업 내용이 기록.
  - **서로 다른 디스크 상의 제어 파일 다중화**
    - 모든 Tibero 데이터베이스는 최소한 두 개의 제어 파일을 보유.
    - 데이터베이스 작업 중 Tibero 서버는 CONTROL_FILES 변수에 나열된 첫 번째 파일만 Read.
    - 데이타베이스 작업 중 사용할 수 없는 제어 파일이 생기면 인스턴스는 더 이상 작동할 수 없으며 중지.
  - **제어 파일의 올바른 배치**
    - 제어 파일의 각 복사본을 각기 다른 디스크상에 저장.
    - 온라인 로그 파일의 멤버와 같은 디스크에 두어서 디스크 손실 시 위험을 최소화
  - **제어 파일의 크기**
    - 제어 파일 크기는 CREATE DATABASE문에서 지정한 MAXDATAFILES, MAXLOGFILES,MAXLOGMEMBERS,MAXLOGHISTORY및 MAXINSTANCE의 설정 값과 운영체제마다 상이.

Controlfile 관리

- **제어 파일(Control File) 관리**
  - 제어 파일은 데이터베이스 자체에 대한 메타 데이터를 보관하고 있는 바이너리 파일
  - 제어 파일은 Tibero 시스템에 의해서만 생성 및 갱신 가능
  - 제어 파일에 포함된 내용.

정보	설명
데이터베이스	데이터베이스 이름, $TB_SID.tip 파일의 이름 또는 생성되었거나 변경된 타임스탬프 등이 존재.
테이블스페이스	테이블스페이스를 구성하는 데이터 파일 또는 생성되었거나 변경된 타임스탬프 등이 존재.
데이터 파일	데이터 파일의 이름과 위치 또는 생성되었거나 변경된 타임스탬프 등이 존재.
Redo 로그	로그 그룹의 개수 및 이를 구성하는 로그 멤버(로그 파일)의 이름과 위치 또는 생성되었거나 변경된 타임스탬프 등이 존재.
체크포인트	최근 체크포인트를 수행한 타임스탬프 등이 존재.

- 데이터베이스 기동시 Control File 내용이 우선 참조.
- 데이터데이스 기동 순서



TmaxTibero

Controlfile 관리

- **컨트롤 파일에 대한 정보**
  - Tibero에서는 여러가지 시스템 뷰를 정의하여 컨트롤 파일의 관리

뷰	설명
V$DATABASE	ARCHIVELOG 모드인지 여부, 체크포인트 등의 정보
V$CONTROLFILE	컨트롤 파일의 상태 및 파일이름 정보

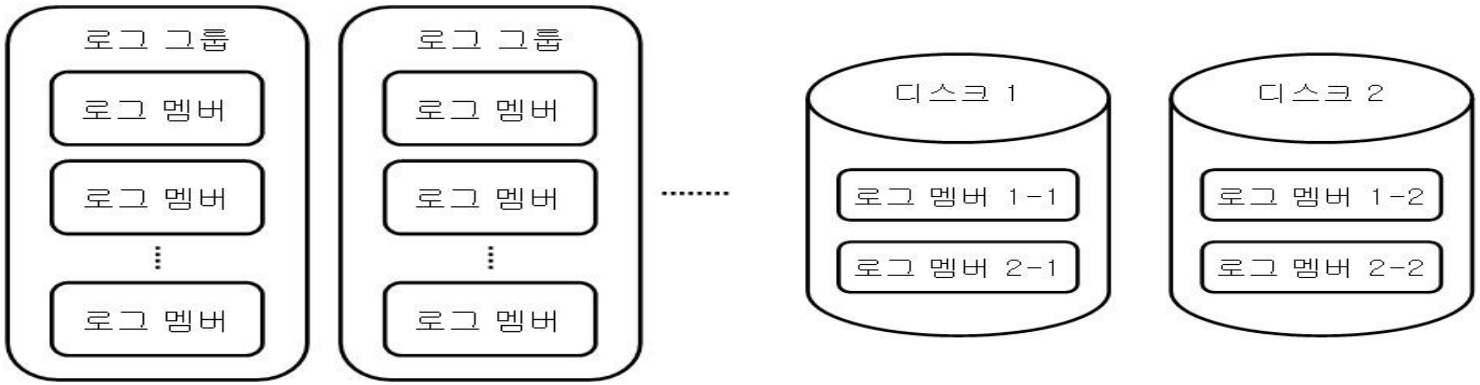
```
SQL> SELECT NAME, CREATE_DATE, CURRENT_TSN, RESETLOG_TSN, LOG_MODE, CKPT_TSN
         FROM V$DATABASE;

NAME     CREATE_DATE  CURRENT_TSN   RESETLOG_TSN    LOG_MODE    CKPT_TSN
------- ------------ ------------ -------------- ------------ ----------
tibero    2009/05/20        86723              0 NOARCHIVELOG       86596
```

```
SQL> SELECT STATUS,  NAME FROM V$CONTROLFILE;

STATUS   NAME
------- ----------------------------
      0  /home/tibero6/tbdata/c1.ctl
```

Redo Log 관리

- **리두 로그 파일 (Redo Log File)의 관리**
  - 리두 로그 버퍼에 기록된 내용을 디스크에 기록하는 데이터베이스 구성 요소.
  - 리두 로그 버퍼엔 데이터베이스 장애시 복구를 수행하기 위해 모든 변경내역을 기록.

  - 리두 로그 파일의 구성 요소
    - 리두 로그 멤버 (Redo Log Member)
    - 리두 로그 그룹 (Redo Log Group)



TmaxTibero

Redo Log 관리

- **리두 로그 파일의 관리**
  - 로그 그룹 추가

```
SQL> ALTER DATABASE ADD LOGFILE GROUP 3
('/home/tibero6/tbdata/redo31.log', '/home/tibero6/tbdata/redo32.log') SIZE 512K;
```

  - 로그 그룹 삭제

```
SQL> ALTER DATABASE DROP LOGFILE GROUP 3 ;
```

  - 로그 멤버 추가

```
SQL> ALTER DATABASE ADD LOGFILE MEMBER  '/home/tibero6/tbdata/redo33.log'  TO GROUP 1;
```

  - 로그 멤버 삭제

```
SQL> ALTER DATABASE DROP LOGFILE MEMBER '/home/tibero6/tbdata/redo33.log';
```

Redo Log 관리

- **리두 로그 파일 정보 조회**
  - Tibero는 리두 로그의 관리를 위해 Redo 로그에 대한 정보를 제공하는 여러가지 뷰를 정의.

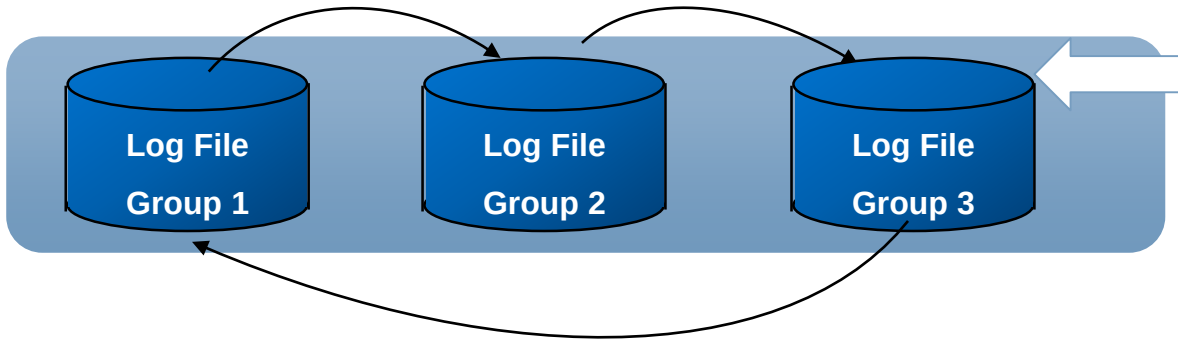
뷰	설명
V$LOG	로그 그룹에 대한 정보
V$LOGFILE	로그 파일에 대한 정보

```
SQL> SELECT * FROM V$LOG;

 GROUP#  SEQUENCE# BYTES        MEMBERS ARCHIVED STATUS   FIRST_CHANGE# FIRST_TIME
-------- --------- --------- ---------- -------- -------- ------------- ----------
       0         5  10485760          2 NO       INACTIVE         18210 2007/01/01
       1         6  10485760          2 NO       CURRENT          23247 2007/01/01
       2         4  10485760          2 NO       INACTIVE         12100 2007/01/01
```

```
SQL> SELECT GROUP#, TYPE, MEMBER  FROM V$LOGFILE;

 GROUP#     TYPE    MEMBER
-------- ------- --------------------------------
            0   ONLINE /home/tibero6/tbdata/log001.log
            0   ONLINE /home/tibero6/tbdata/log002.log
            1   ONLINE /home/tibero6/tbdata/log011.log
            1   ONLINE /home/tibero6/tbdata/log012.log
            2   ONLINE /home/tibero6/tbdata/log021.log
            2   ONLINE /home/tibero6/tbdata/log022.log
```

Redo Log 관리

- **Online Redo Log Files**



- 데이터 변경 내용
- TSN 과 타임 스탬프
- 트랜잭션 ID
- 트랜잭션이 커밋 될 때의 TSN과 타임 스탬프 (트랜잭션이 커밋 된 경우)
- 변경을 수행 한 작업의 유형
- 변경된 데이터 세그먼트의 이름 및 유형

➢ 복구를 위해 사용되며, 데이터베이스 변경이 발생하면 변경내용이 기록됨

➢ 리두 로그 파일들은 순환하며 Write

➢ 리두 로그 그룹은 적어도 2개 이상

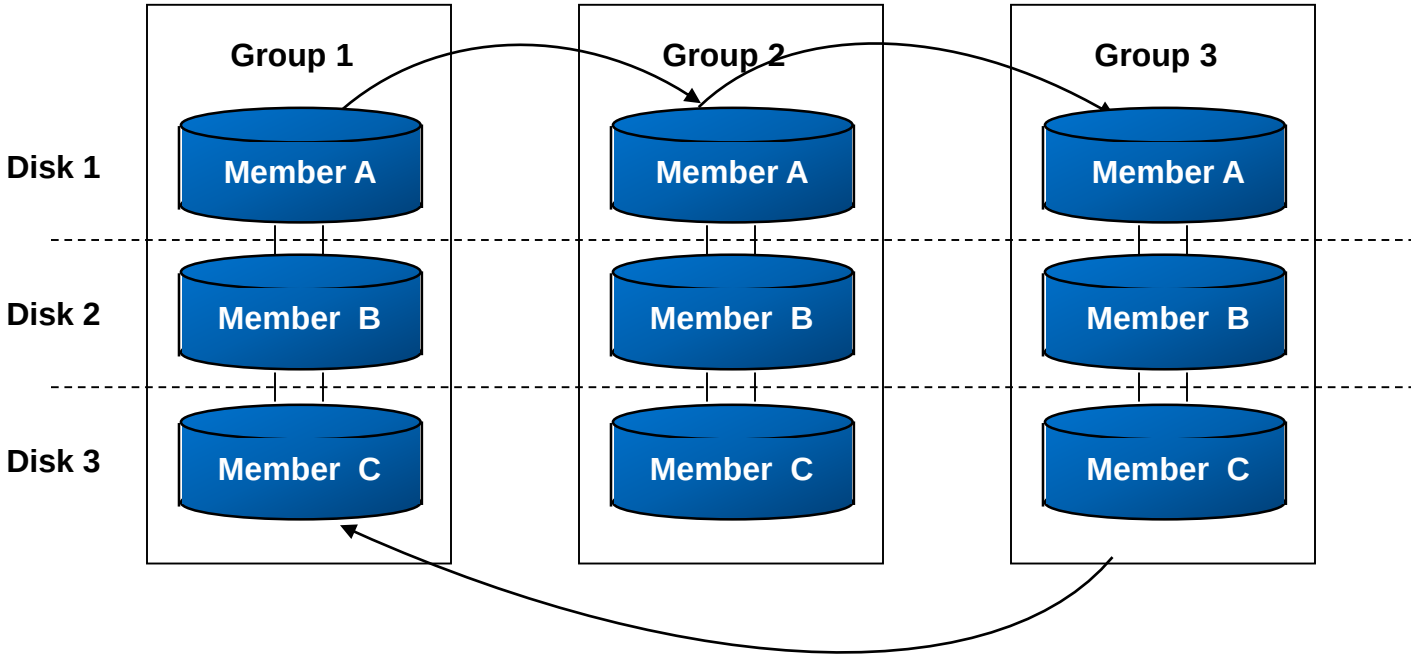
➢ 만약 Database운영 Mode가 ARCHIVELOG Mode일 경우 Log Switch가 일어날 때 log_archive_dest로 Copy되어져 향후 복구 시 사용됨

➢ (인스턴스 복구시) 온라인 리두로그의 내용을 이용하여 아직 데이터 파일에 기록되지 않은 커밋된 데이터를 복구함

➢ 리두로그 버퍼의 내용(커밋되지 않은 트랜잭션, 커밋된 트랜잭션, 언두 데이터, 스키마 객체 관리를 위해 사용한 쿼리 문 등) 이 기록됨.

Redo Log 관리

- **Multiplexed Redo Log Files**

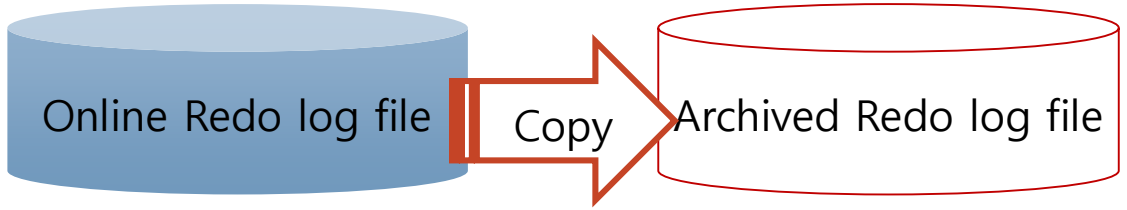


- Redo log File을 구성하는데 1 Group에 2개 이상의 Member를 동일 Machine상에서 다른 Disk에 분리하여 구성하기 권장.
- Multiplexed Redo Log Files는 특정 그룹에서 하나의 파일을 손실 시에 특이 사항 없음.
- 그룹의 모든 Member들은 동일한 정보와 Size 를 갖음

TmaxTibero

Redo Log 관리

● Archived Redo Log Files



- 아카이브 리두 로그 파일은 온라인 리두로그 그룹 맴버의 복사본임
- 아카이브 리두 로그 파일은 데이터베이스 파일에 속하지는 않지만 복구르 위해 사용되는 중요한 파일임
- 아카이브 리두 로그의 용도
  - 데이터베이스 백업 복구
  - TSC(Tibero Standby Cluster) 의 Standby DB 의 데이터 업데이트
  - Tmax 의 데이터복제 솔루션인 ProSync 에서 사용

Tablespace 와 Datafile 관리

테이블스페이스(Tablespace)

- **개요**
  - 하나 이상의 논리적 저장 영역으로 구성되어 있으며 Tibero의 모든 데이터를 저장
  - 각 테이블스페이스는 물리적인 저장 영역인 데이터파일이라는 하나 이상의 파일로 구성되며, 이 데이터 파일은 운영체제의 물리적인 저장영역

- **TABLESPACE 관리 지침**
  - Using Multiple Tablespaces
    - Data Dictionary Data 와 유저 Data의 분리
    - 어플리케이션별 분리
    - I/O 경합을 줄이기 위하여 데이터 파일의 분산(디스크)
    - 유저 Data 와 Rollback Segment의 분리
    - 개별 테이블스페이스를 오프라인 할 수 있도록 할 것.
    - 업데이트가 많은것, 읽기 전용, 임시용 등 특별한 형태별 지정
    - 개별 테이블스페이스의 백업 고려

Tablespace 와 Datafile 관리

● 테이블스페이스(Tablespace) 개요

➢ 테이블스페이스 유형

구성요소	설명
SYSTEM TABLESPACE	데이터베이스 작업을 위해 항상 있어야 함. (항상 ON-LINE 상태 유지) 포함 내용 - 데이터 딕셔너리 정보와 저장 프로시저, 패키지, 그리고 데이터베이스 트리거의 정의. SYSTEM 롤백 세그먼트를 포함. 주의 : 사용자 데이터를 포함 할 수 있지만 그렇지 않도록 할 것.
NON SYSTEM TABLESPACE	응용프로그램 데이터/인덱스, 임시 세그먼트, 롤백 세그먼트

➢ 테이블스페이스 영역 관리 방법

- Locally Managed Tablespace 방식

구성요소	설명
EXTENT_MANAGEMENT_LOCAL	EXTENT_MANAGEMENT_LOCAL의 뒷 부분에 테이블스페이스의 익스텐트가 어떻게 관리될 것인지를 명시.
AUTOALLOCAT	익스텐트의 크기를 시스템이 자동으로 설정.
UNIFORM	익스텐트의 크기를 사용자가 설정한다. 사용자가 설정한 크기대로 익스텐트가 항상 일정한 크기로 생성.
size	익스텐트의 크기를 명시. 기본값은 16이며, 8~8388608까지 명시 가능.

Tablespace 와 Datafile 관리

● 테이블스페이스 관리

➢ 테이블스페이스 생성

- System Tablespace는 데이터베이스가 생성될 때 생성
- Non System Tablespace 생성하기 (일반 유저가 사용하는 Tablespace)

```
SQL> CREATE TABLESPACE my_space
     DATAFILE '/home/tibero6/tbdata/my_file001.dtf' SIZE 50M
     AUTOEXTEND ON NEXT 10M MAXSIZE 3G
     EXTENT MANAGEMENT LOCAL
     UNIFORM SIZE 256K;
```

➢ 테이블스페이스 크기 변경

- 테이블스페이스의 크기는 물리적인 데이터파일의 크기 변경과 데이터파일 추가로 변경 가능
- 데이터파일의 크기 변경

```
SQL> ALTER DATABASE DATAFILE 'my_file001.dtf' RESIZE 100M;
```

- 데이터파일 추가

```
SQL> ALTER TABLESPACE my_space ADD DATAFILE 'my_file002.dtf' SIZE 20M;
```

➢ 테이블스페이스 삭제

- 테이블스페이스를 구성하는 데이터 파일을 함께 제거시 INCLUDING 절을 삽입

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

Tablespace 와 Datafile 관리

● 테이블스페이스 관리

➢ 테이블스페이스 가용성 변경

- 특정 테이블스페이스에 읽고 쓰는 모든 접근을 허용하지 않으려면 ALTER TABLESPACE 문에서 OFFLINE 절을 이용하여, 테이블스페이스를 오프라인 상태로 변경.

- 예제) 테이블스페이스를 오프라인 상태로 만들고 미디어복구 수행 후 온라인으로 변경.

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

➢ 테이블스페이스 오프라인은 NORMAL과 IMMEDIDATE 두 가지 모드를 지원.

모드	설명
NORMAL	체크포인트를 수행한 후 테이블스페이스 오프라인을 수행. 향후 테이블스페이스 온라인을 수행할 때 미디어 복구 불필요.
IMMEDIATE	체크포인트를 수행하지 않고 테이블스페이스 오프라인을 수행. 향후 테이블스페이스 온라인을 수행할 때 미디어 복구가 필요. 따라서 ARCHIVELOG 모드에서만 가능.

Tablespace 와 Datafile 관리

- **테이블스페이스 Read-Only**

  - **Making a Tablespace Read-Only**
    - 테이블스페이스를 읽기 전용으로 만들면 파일의 쓰기 작업 방지.
    - ALTER TABLESPACE의 SYSTEM 권한 필요.
    - 필요조건
      - 테이블 스페이스는 ONLINE 상태
      - SYSTEM 테이블스페이스는 지정 불가
      - 테이블스페이스에 활성 롤백 세그먼트가 없어야 함.
      - 테이블스페이스가 현재 온라인 백업이 없어야 함.
    - Making a Read-Only Tablespace Writeable
      - ALTER TABLESPACE tablespace명 READ WRITE;
      - 모든 데이터파일의 상태가 ONLINE,
      - dba_tablespaces 를 조회하면 데이터파일의 상태 조회 가능.

  - **Dropping Tablespace**
    - DROP TABLESPACE tablespace명 INCLUDING CONTENTS;
    - 테이블스페이스가 비어 있으면 INCLUDING CONTENTS는 필요 없음
    - CASCADE CONSTRAINTS는 하위 테이블의 FOREIGN KEY는 단계적으로 삭제됨.

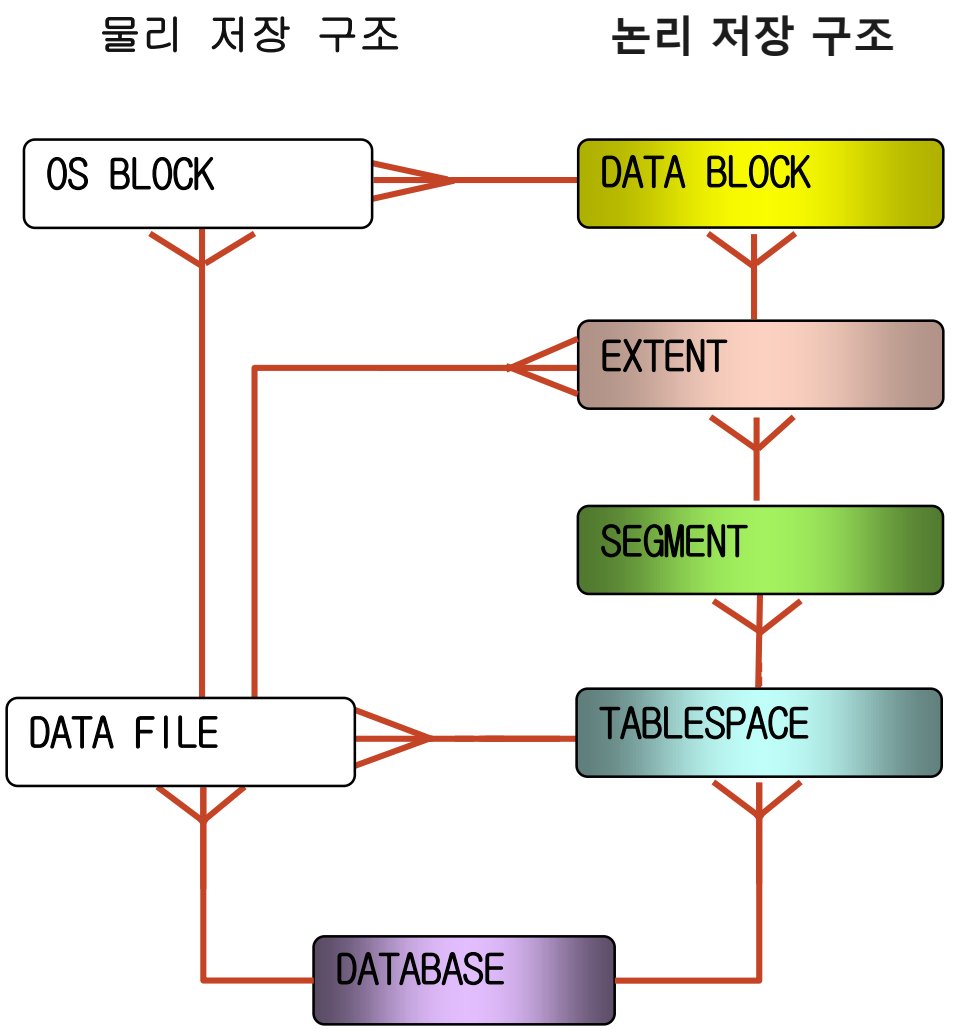
Tablespace 와 Datafile 관리

- **테이블스페이스 정보 조회**
  - Tibero에서는 테이블스페이스에 대한 관리를 용이하게 하기 위하여, 여러가지 뷰를 정의하여 테이블스페이스에 대한 정보를 제공

뷰	설명
DBA_TABLESPACES	티베로내의 모든 테이블스페이스에 대한 정보
USER_TABLESPACES	현재 유저에 속한 테이블스페이스에 대한 정보
V$TABLESPACE	티베로내의 모든 테이블스페이스에 대한 간략한 정보

```
SQL> SELECT TABLESPACE_NAME, TS_ID, DATAFILE_COUNT, BLOCK_SIZE, STATUS, CONTENTS
       FROM DBA_TABLESPACES;

TABLESPACE_NAME      TS_ID DATAFILE_COUNT BLOCK_SIZE STATUS    CONTENTS
--------------- ---------- -------------- ---------- --------- ---------
SYSTEM                    0              1       8192 ONLINE    PERMANENT
UNDO                      1              1       8192 ONLINE    UNDO
TEMP                      2              1       8192 ONLINE    TEMPORARY
USR                       3              1       8192 ONLINE    PERMANENT
SYSSUB                    4              1       8192 ONLINE    PERMANENT
```

Tablespace 와 Datafile 관리

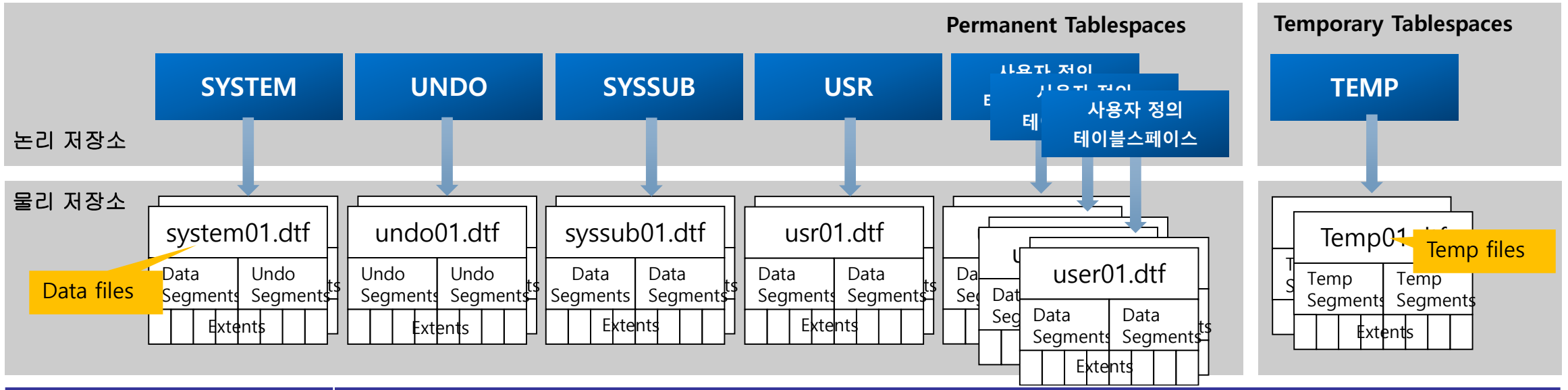


- **Data Block**
  - 데이터 파일에 할당되는 물리적 파일 블럭
  - Data가 저장되는 최소의 구조단위.
  - 2k, 4k, 8k, 16k, 32k
- **Extent**
  - 연속된 데이터베이스 블럭의 집합
- **Segment**
  - tablespace내 특정 구조에 대한 모든 데이터를 갖고 있는 하나 혹은 하나 이상의 익스텐트의 집합.
  - table, index segment
- **Tablespace**
  - 물리적으로 그룹화된 데이터를 위한 논리적 저장 단위.
- **Database**
  - 테이블스페이스가 저장되어서 공유되는 논리적 집합체.

Tablespace 와 Datafile 관리

● 테이블스페이스

- 테이블스페이스는 세그먼트를 저장하는 논리 저장소, 한개 이상의 데이터 파일이나 임시파일을 사용하여 데이터를 저장함



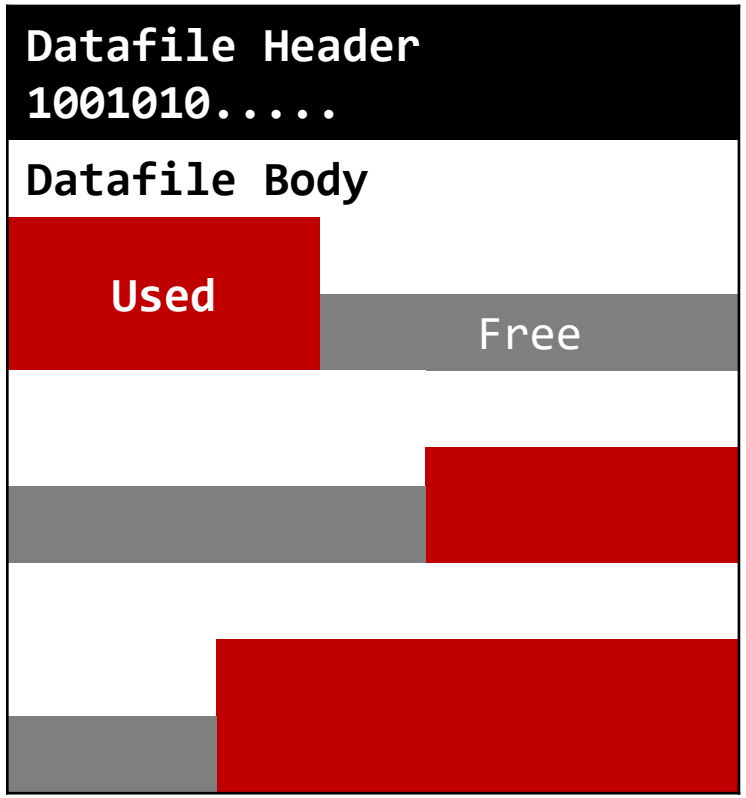
Tablespace	포함하고 있는 데이터
SYSTEM	- 데이터베이스 관리 정보를 포함한 테이블, 뷰(데이터 사전) - 프로시저, 패키지, 트리거 등 컴파일된 오브젝트
UNDO	- 언두 데이터(언두 세그먼트)
SYSSUB	- TPR 의 스냅샷 데이터
USR	- TIBERO, TIBERO1 유저의 default tablespace
TEMP	- 정렬 작업용 데이터, 세션의 유지 기간만 저장되는 임시 데이터

TmaxTibero

Tablespace 와 Datafile 관리

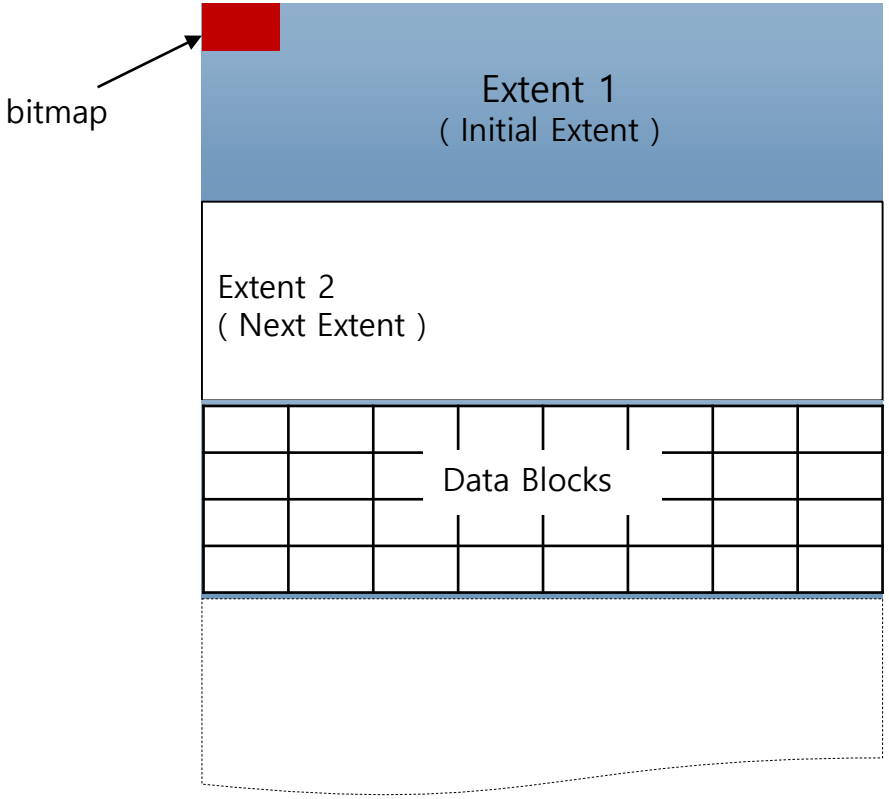
● 테이블스페이스 공간 관리

- 티베로는 로컬 관리 방식으로 데이터 파일 헤더에 데이터파일의 공간을 관리하기 위한 비트맵을 생성하여 사용함



세그먼트 공간 관리

- 자동 세그먼트 공간방식(ASSM: Automatic Segment Space Management)으로 초기할당 익스텐트의 비트맵을 사용하여 관리함.
- 공간관리에 사용하는 유일한 파라미터는 PCTFREE 로써 블록의 빈 공간의 비율을 설정함

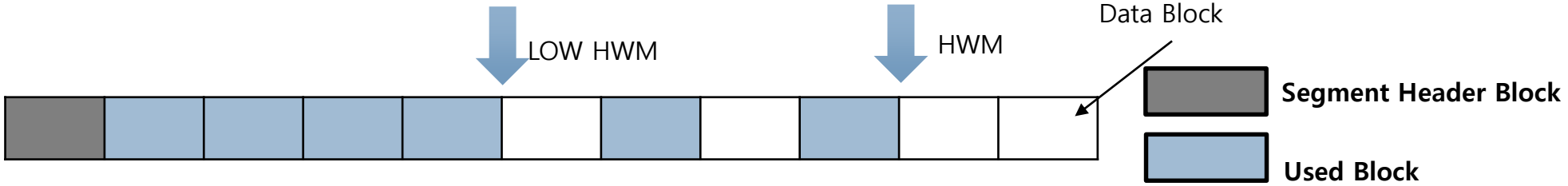


TmaxTibero

Tablespace 와 Datafile 관리

● 세그먼트 공간관리 와 HWM(High Water Mark)

- 자동 세그먼트 공간 관리 방식으로써 BITMAP 을 이용하여 공간을 관리함
- PCTFREE 파라미터를 이용하여 블록 속성 설정
  (다른 파라미터는 허용은 되나 무시됨)
- 데이터 입력시 블록그룹을 할당하며 세그먼트 헤더블록의 비트맵에 블록 정보를 기록합니다.



- 데이터 입력시 LOW HWM 하위의 여유공간이 있는 블록에 쓰거나, LOW HWM 과 HWM 사이의 블록을 사용함
- FULL SCAN 시에 LOW HWM 하위의 모든 블록과 LOW HWM 과 HWM 사이의 경우는 세그먼트 헤더블록의 BIT MAP 정보를 이용하여 액세스 함

TmaxTibero

Tablespace 와 Datafile 관리

● 세그먼트 생성

- 테이블, 인덱스 객체 생성시 세그먼트가 할당됩니다.
- 파티션 되어 있는 객체는 파티션마다 세그먼트가 할당됩니다.

```
CREATE TABLE MY_SEGEMENT(C1 NUMBER);
```

스키마 오브젝트	세그먼트	세그먼트 타입
MY_SEGEMENT	MY_SEGEMENT	TABLE

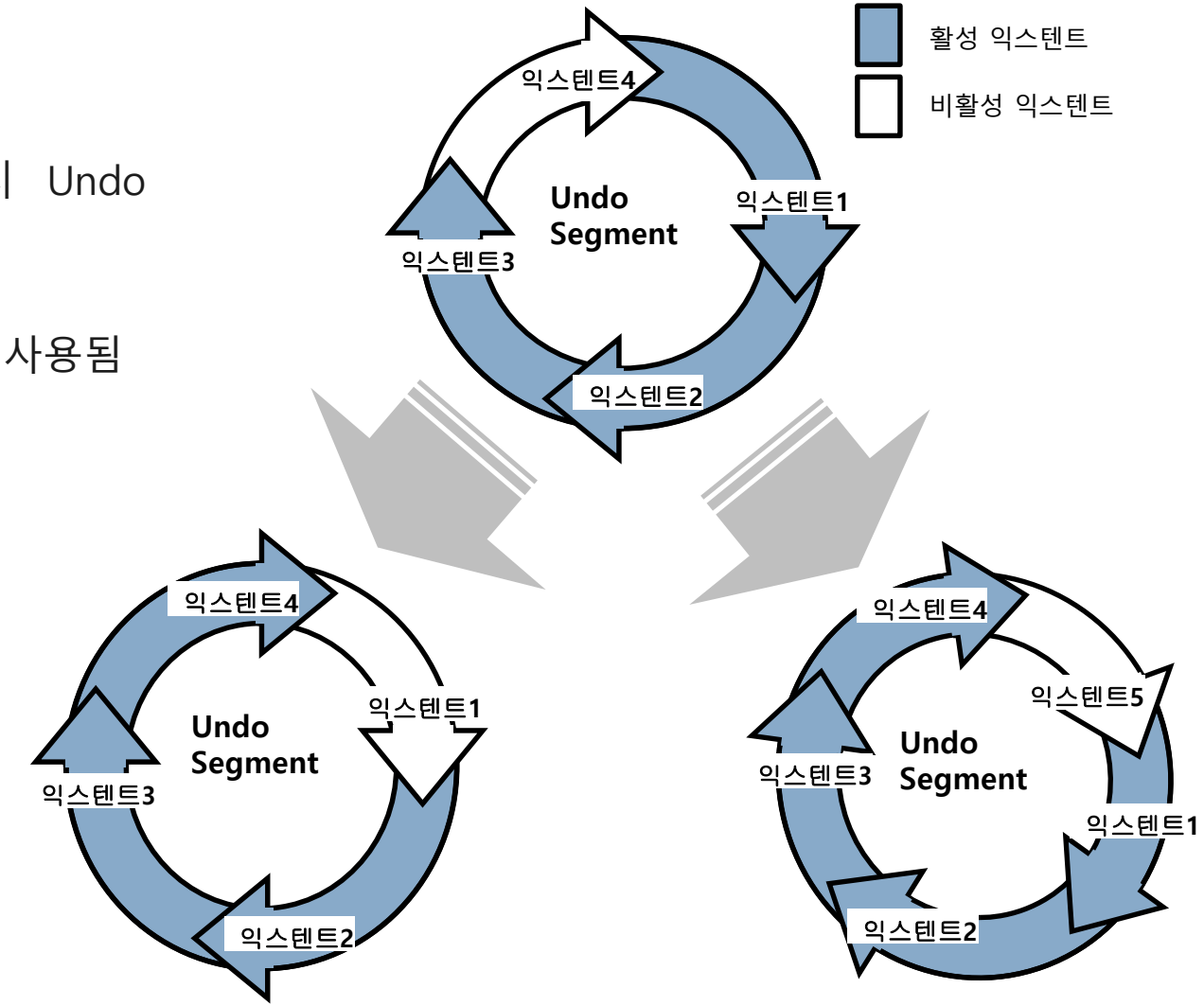
```
CREATE TABLE MY_SEGEMENT(C1 NUMBER PRIMARY KEY
                        ,C2 CLOB                );
```

스키마 오브젝트	세그먼트	세그먼트 타입
MY_SEGEMENT	MY_SEGEMENT	TABLE
TIBERO_CON32500741	TIBERO_CON32500741	INDEX
TIBERO_LOB272700	TIBERO_LOB272700	LOB
TIBERO_LIDX272800	TIBERO_LIDX272800	INDEX

Tablespace 와 Datafile 관리

● UNDO 테이블스페이스

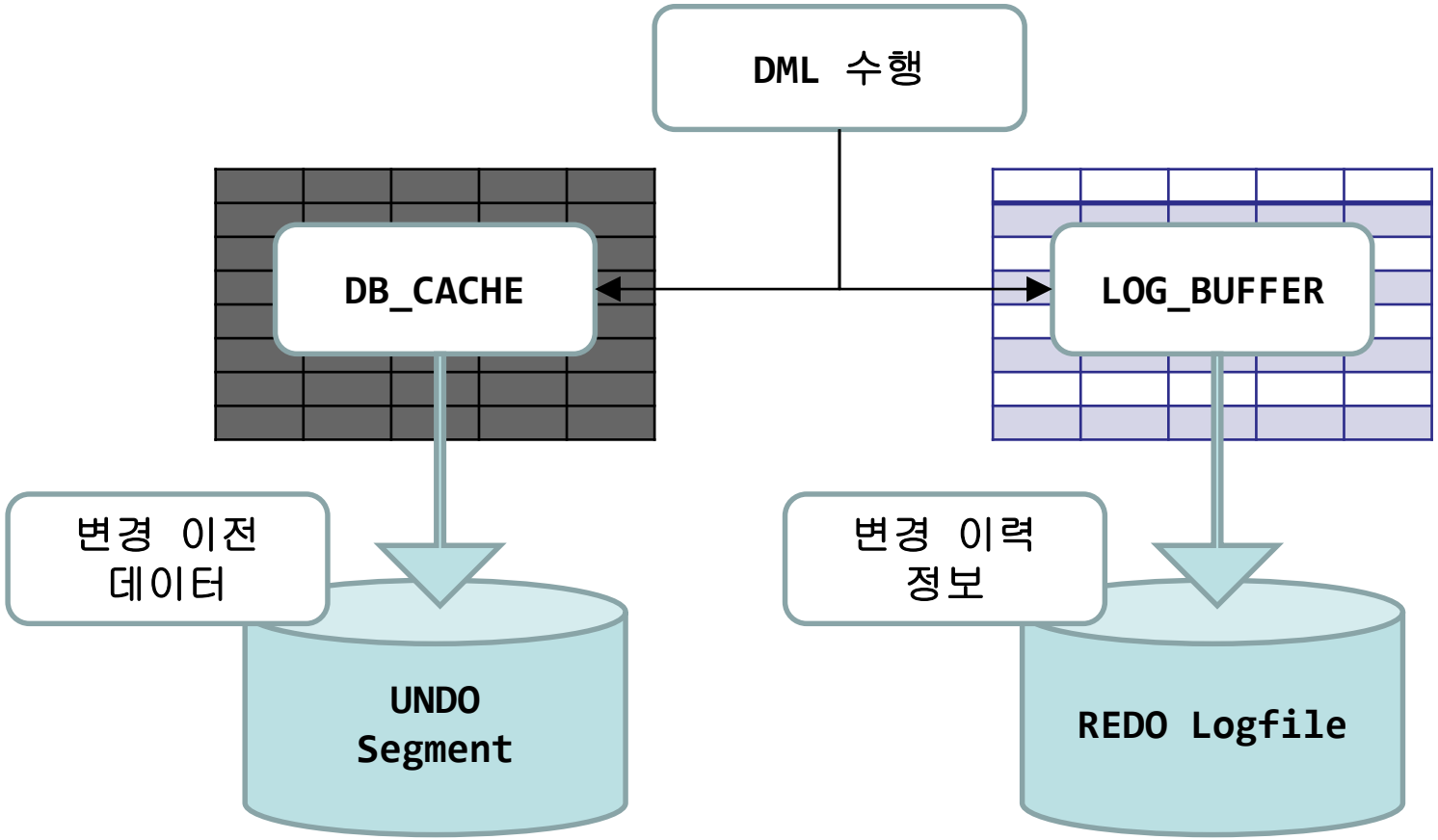
- 모든 Transaction 으로 부터 Undo 데이터 발생
- Transaction 종료시 까지, 또는 그 이상 시점까지 Undo 데이터는 보존됨
- 언두 데이터는 트랜잭션 기록을 보관하며, 다음의 경우 사용됨
  - 트랜잭션 롤백
  - 트랜잭션 복구
  - 읽기 일관성 작업
  - 플래시백 작업



TmaxTibero

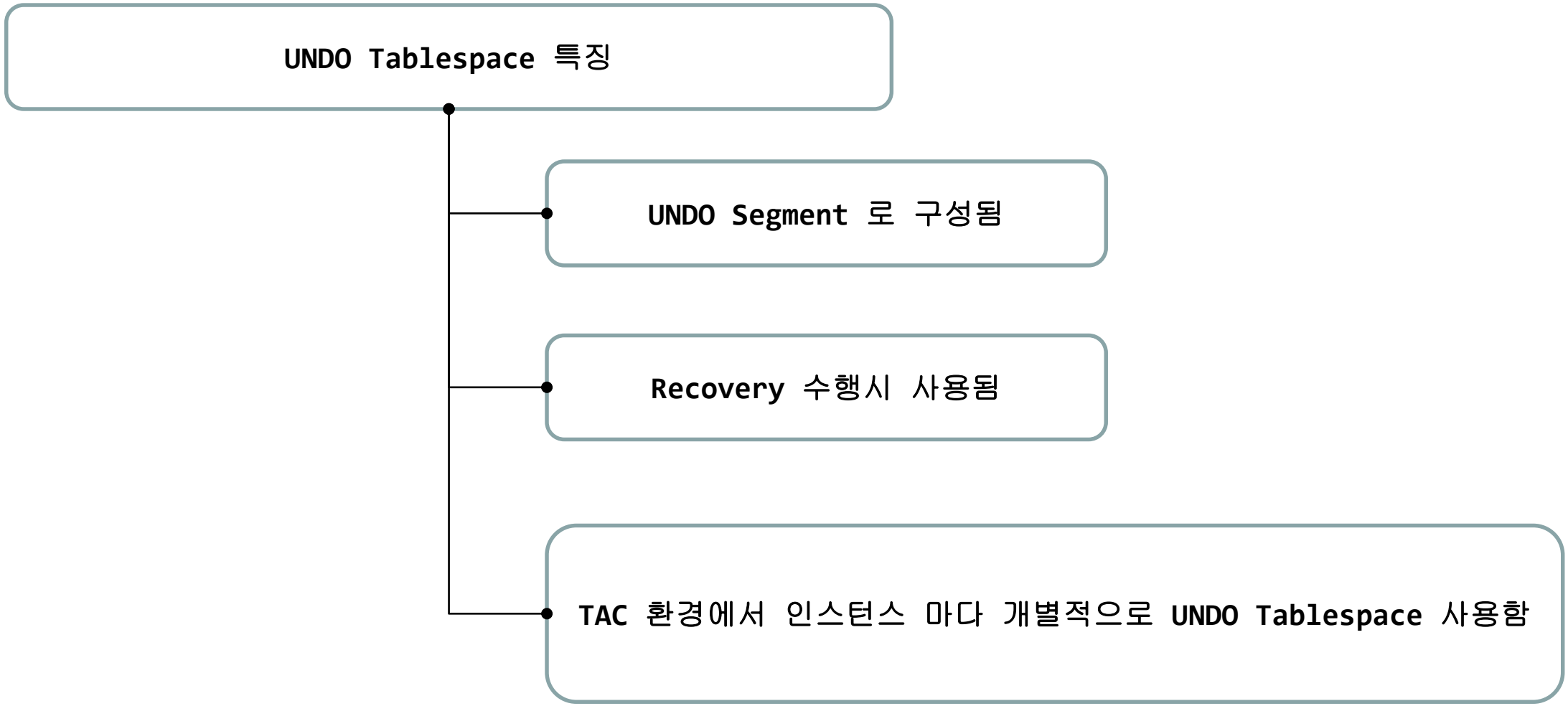
Tablespace 와 Datafile 관리

- UNDO 테이블스페이스
  - ✓트랜잭션은 하나의 언두 세그먼트를 할당받음
  - ✓한개의 언두 세그먼트에 여러 트랜잭션이 할당 될 수 있음

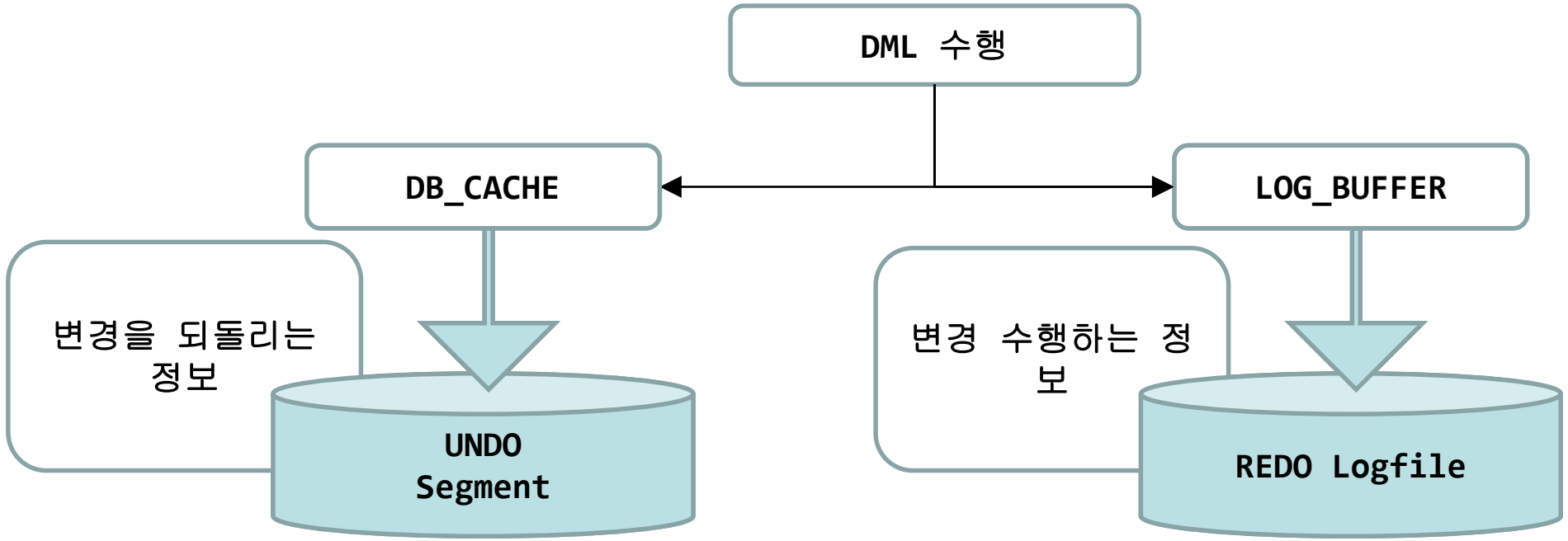


Tablespace 와 Datafile 관리

- UNDO Segment 는 UNDO Tablespace 에 저장



Tablespace 와 Datafile 관리



UNDO 역할

- 롤백 기능
- 플래시백 기능
- 동시 사용자에 대한 읽기 일관성

REDO 역할

- Instance Crash 발생시 롤포워드
- 데이터파일 복구시 사용

Tablespace 와 Datafile 관리

UNDO 자동 관리

- 언두 테이블스페이스에서 언두 데이터 저정하며, 공간을 자동으로 관리함
- 공간 필요시 자동으로 언두 세그먼트에 익스텐트 추가 할당됨
- 읽기 일관성 보장을 위해 **UNDO_RETENTION** 시간 동안 언두데이터 유지

UNDO_RETENTION

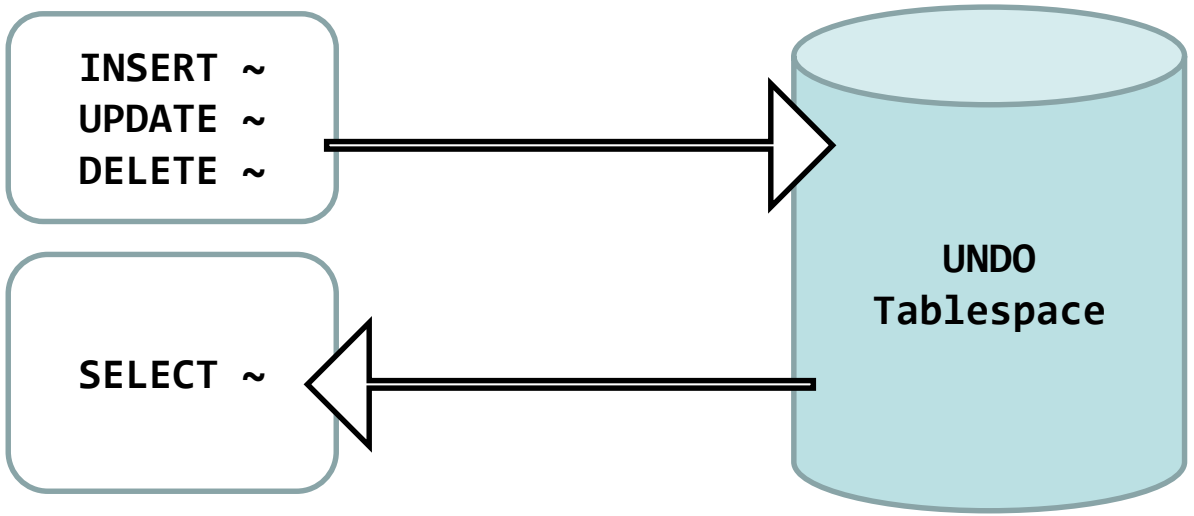
- **COMMIT** 언두 데이터의 보존하는 기간을 설정하는 파라미터(단위: 초)
- 기본값 **900초(15분)** 으로, 해당 언두 익스텐트의 데이터 보장
- **UNDO_RETENTION** 기간이 지난 익스텐트는 진행 중인 트랜잭션을 위해 재할당 될 수 있음 **RETENTION** 기간 이내인 경우 익스텐트는 재활용 될 수 없음

UNDO EXTENT 의 상태

- **Active, Unexpired, Expired**

Tablespace 와 Datafile 관리

- UNDO TABLESPACE 에서 가용한 UNDO Extents 가 있을 경우, DML 쿼리 수행 정상 동작.
- SELECT 문장이 UNDO_RETENTION 이하의 수행시간일때 정상 동작을 보장(기본값 900 Seconds (15Min.))

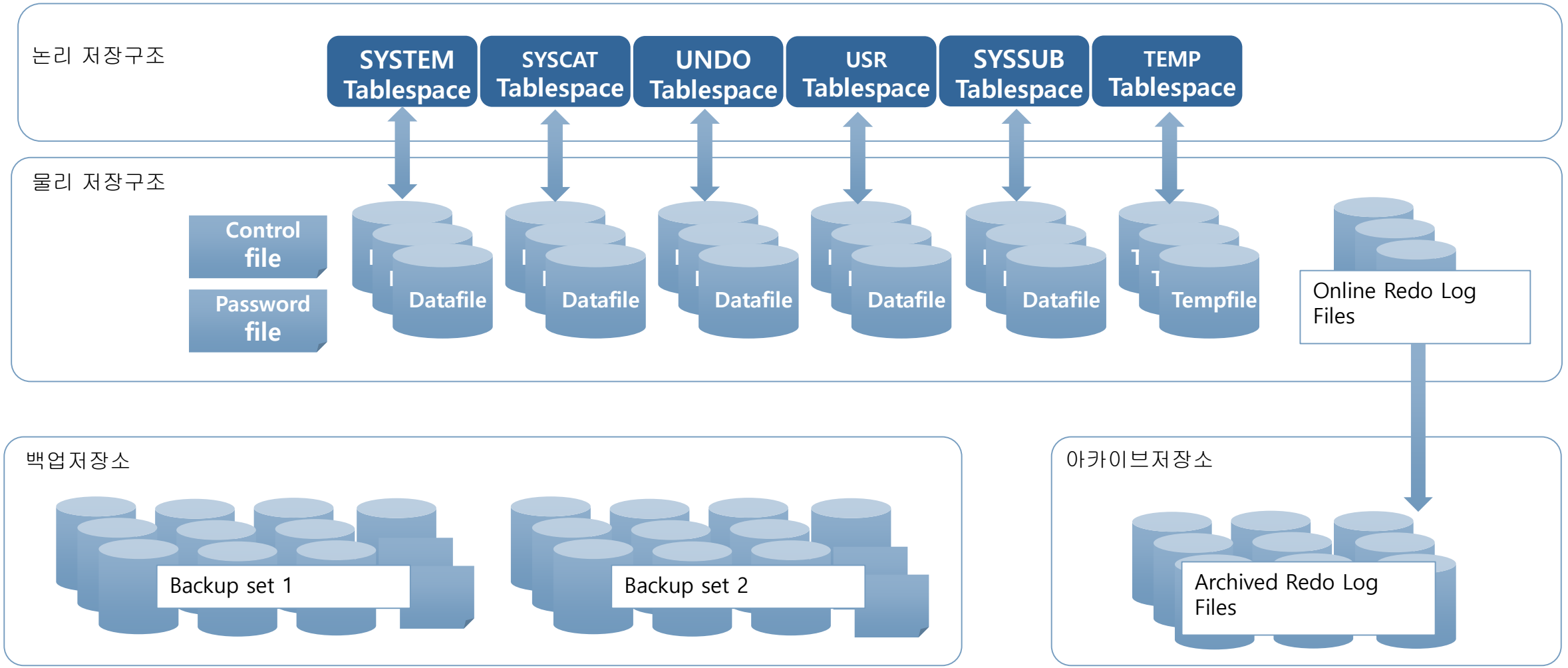


- UNDO_RETENTION 측정시점
  - COMMIT 시점 이후부터 측정함

- UNDO Tablespace 크기 증가
  - 기존 Expired 상태의 UNDO Extent 를 소모하고, 추가적으로 Active 상태의 UNDO Extent 가 많아지는 경우임

Tablespace 와 Datafile 관리

- **Tibero Database 저장소 구조**



TmaxTibero

Tablespace 와 Datafile 관리

Tibero Database 파일 저장 방식

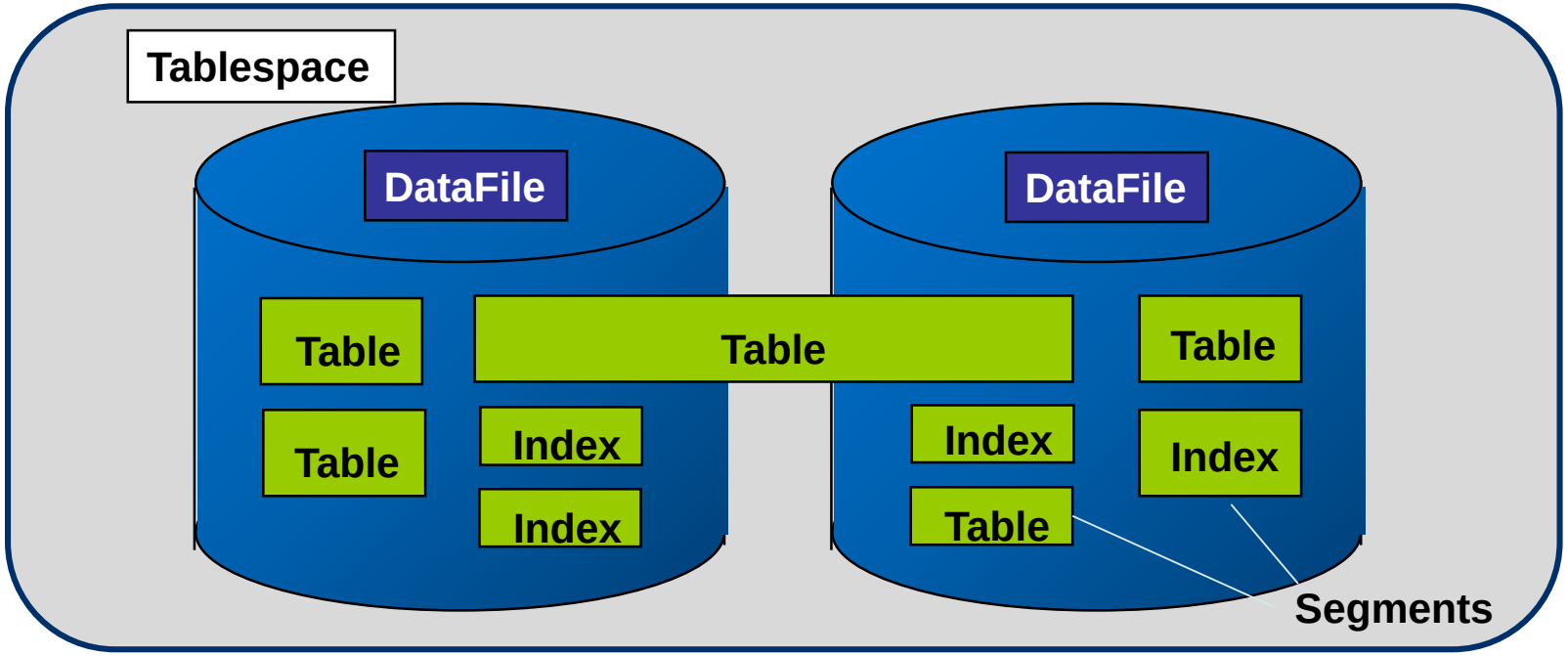
- Tibero Active Storage (TAS)
  - Tibero Database 에서 사용하는 전용 파일시스템
  - Tibero TAC 환경에서 TAS 를 기반으로 데이터베이스를 생성하여 운영할 수 있음

- 운영체제의 파일 시스템
  - 일반적으로 파일시스템은 논리볼륨관리자(LVM) 에 의해 생성된 논리볼륨을 기반으로 구축되어 있고, LVM 은 서로 다른 물리적 디스크 영역을 하나의 연속된 주소 공간으로 결합하여 제공함

- RAW 장치
  - Raw Device 는 파일시스템으로 포맷되어 있지 않은 디스크 파티션 혹은 논리 볼륨으로, 파일시스템과 달리 캐시를 거치지 않고 직접 I/O 를 수행할 수 있음
  - Tibero TAC 환경에서 TAS 를 기반으로 데이터베이스를 생성하여 운영할 수 있음

- 클러스터 파일 시스템
  - 클러스터 파일 시스템은 여러 컴퓨터에서 파일의 저장소를 공유 하는 기능을 제공함
  - Tibero TAC 환경에서 클러스터 파일시스템을 기반으로 데이터베이스를 생성하여 운영할 수 있음

Tablespace 와 Datafile 관리



- **Datafiles**
  - Tibero Database 는 데이터 파일 이라는 구조에 데이터베이스 데이터를 저장합니다.
  - Tablespace는 하나이상의 물리적인 datafile을 가지며, 하나의 datafile은 오직 하나의 Tablespace에 포함
  - 세그먼트는 하나 이상의 데이터 파일에 걸쳐 있을 수 있지만, 여러 테이블스페이스에 걸쳐져 있는 것은 아님

Tablespace 와 Datafile 관리

- **Datafile 관리 지침**

  - **DATAFILE**
    - Tablespace는 하나 이상의 물리적인 datafile을 가지며, 하나의 datafile은 오직 하나의 Tablespace에 포함.

  - **DATAFILE 관리 지침**
    - 사용자는 datafile 을 추가할 수 있으며 다음 제한을 갖음.
      - Operating system limit
      - Tibero system limit
      - Control file upper bound : MAXDATAFILES
    - Set the Size of Datafiles
      - 데이터 딕셔너리를 저장하기 위한 SYSTEM 테이블스페이스 datafile은 최소한 2M 이상을 지정.
    - Store Datafiles Separately From Redo Log Files
      - 데이터 파일과 리두 로그 파일의 디스크를 분리하여 지정.

Tablespace 와 Datafile 관리

- **Tablespace에 Datafile 작성 및 추가**

➢ **Creating and Adding Datafiles to a Tablespace**

– CREATE TABLESPACE 문에서 데이터 파일을 생성

– 생성된 테이블스페이스에 데이타 파일을 추가

```
ALTER TABLESPACE rb_segs
  ADD DATAFILE 'filename.dtf' SIZE 10M;
```

➢ **Changing a Datafile's Size**

– Enabling and Disabling Automatic Extension for a Datafile

- CREATE DATABASE, CREATE TABLESPACE, ALTER TABLESPACE 명령으로 크기를 자동으로 증가할 수 있는 데이터 파일 생성 가능.

```
ALTER TABLESPACE users
  ADD DATAFILE 'filename2.dtf' size 10M
    AUTOEXTEND ON
    NEXT 512 K
    MAXSIZE 250M;
```

- ALTER DATABASE DATAFILE 'filename2.dtf' AUTOEXTEND OFF;
- ALTER DATABASE DATAFILE 'filename2.dtf' RESIZE 100M;

Tablespace 와 Datafile 관리

- **DATAFILE 가용성 변경**
  - **Altering Datafile Availability**
    - Bringing Datafiles Online in ARCHIVELOG Mode
      - ALTER DATABASE DATAFILE 'filename' ONLINE;
    - Taking Datafiles Offline in NORACHIVELOG Mode
      - ALTER DATABASE DATAFILE 'filename' OFFLINE DROP;
- **DATAFILE 정보 조회**

뷰	설명
DBA_EXTENTS	테이블 스페이스 내에 포함된 Segment의 Extent 조회
DBA_SEGMENTS	테이블 스페이스 내에 포함된 Segment 조회
DBA_FREE_SPACE	테이블 스페이스 내에 사용 가능한 Extents의 정보 조회
DBA_TABLESPACE	테이블 스페이스 조회
DBA_DATA_FILES	Datafile에 대한 정보 조회
DBA_EXTENTS	테이블 스페이스 내에 포함된 Segment의 Extent 조회
V$DATAFILE	데이터 베이스 내에 Datafile에 대한 정보

Tablespace 와 Datafile 관리

- **DATAFILE 이름 변경 및 위치 재 지정**
  - **데이터파일 이름 변경 및 위치 재지정**
    - Renaming and Relocating Datafiles for a Single Tablespace
      - Datafiles 이 오프라인 된 non-SYSTEM 테이블스페이스 선택
      - Datafiles 복사 혹은 이름 변경
      - ALTER TABLESPACE 명령으로 Datafiles 이름 변경
        - ALTER TABLESPACE users
          RENAME DATAFILE 'filename1', 'filename2' TO 'filename3', 'filename4';
      - 테이블스페이스를 온라인으로 전환
      - OLD 와 NEW Datafiles이 존재 해야 함. 또한 ALTER DATABASE BACKUP CONTROLFILE TO 'filename' 으로 컨트롤 파일을 백업 필요.

  - **Renaming and Relocating Datafiles for a Multiple Tablespace**
    - 한 개 이상의 테이블스페이스의 datafile 혹은 SYSTEM 테이블스페이스의 datafile을 이름변경 혹은 위치변경.
    - 데이터 베이스를 마운트 상태로 기동 (OPEN 상태가 아님)
    - datafile 을 새로운 위치로 복사
    - ALTER DATABASE RENAME 명령을 사용하여 이름 변경
      - ALTER DATABASE
        RENAME FILE 'filename1', 'filename2' TO 'filename3', 'filename4';

Tablespace 와 Datafile 관리

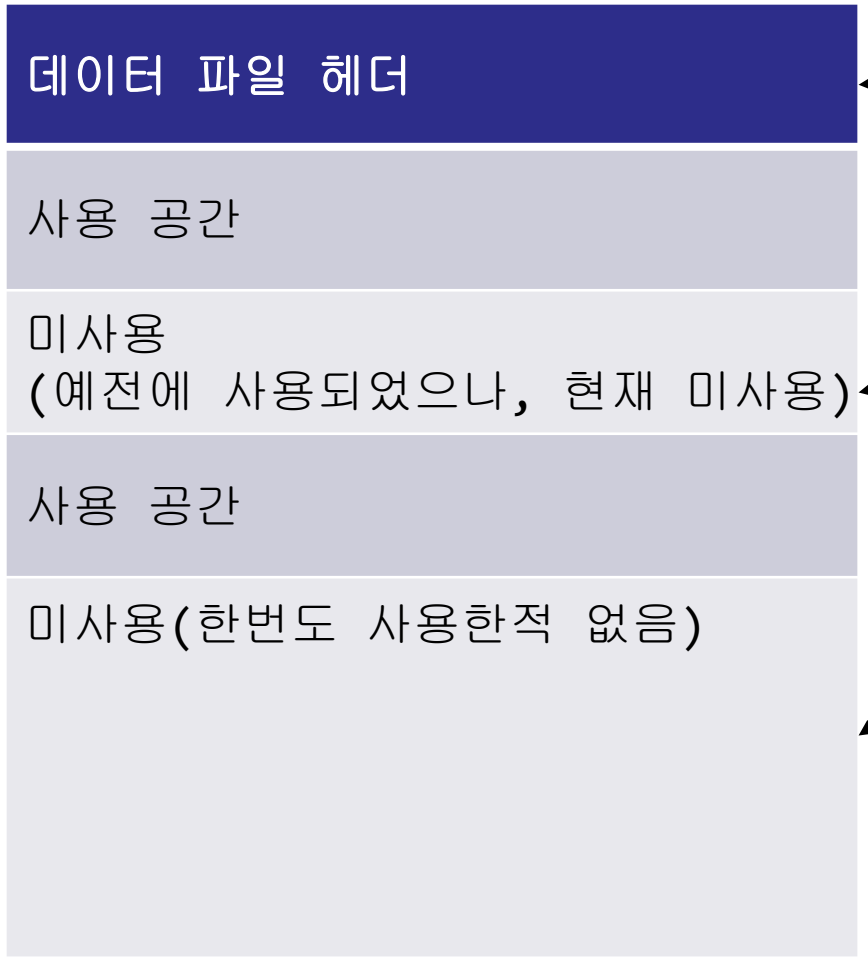
- **일반 테이블스페이스의 영속적인 데이터 파일**
  - 테이블, 인덱스와 같은 영구적인 스키마 객체가 포함되며, 데이터 파일에 저장 관리됨

- **임시 테이블스페이스의 임시파일(Tempfile)**
  - 세션의 또는 트랜잭션 지속 시간 동안 임시 테이블과 같은 스키마 객체의 데이터가 존재하며, 메모리에서의 해시 및 정렬 등의 작업 메모리 공간이 부족했을때의 저장소로 사용됨
  - 일반테이블의 영구적인 데이터는 임시파일에 저장되지 않음
  - 임시파일을 사용하는 스키마 오브젝트는 NOLOGGING 모드로 설정되며, 미디어 복구가 않됨
  - 임시 파일은 DBA_TEMP_FILES, V$TEMPFILE 에서 모니터링 할수 있고, 일반 데이터 파일을 조회하는 DBA_DATA_FILES, V$DATAFILE 뷰에서는 표시되지 않음

- **온라인 데이터 파일 및 오프라인 데이터 파일**
  - 모든 데이터 파일은 온라인 (사용 가능) 또는 오프라인 (사용 불가능) 상태에 해당됨
  - MOUNT 모드에서 데이터파일을 오프라인 변경후, 테이블스페이스를 삭제할 수 있음
  - 테이블스페이스를 오프라인 변경하면 연결된 데이터파일 또한 오프라인 상태가 됨
  - 데이터파일의 이름/경로를 바꾸기 위해서는 해당 테이블 스페이스를 오프라인 상태로 설정후 변경가능

Tablespace 와 Datafile 관리

● 데이터 파일의 구조



데이터 사이즈에 대한 정보, 체크 포인트 TSN, 데이터베이스에서 고유하게 식별하기 위한 파일 절대번호, 테이블스페이스에서 데이터파일을 식별하기 위한 파일 상대번호등의 메타 정보 포함

데이터의 갱신 또는 삭제가 반복되면 중간 사용하지 않는 공간이 존재하게 되며, 재사용하기에는 작은 크기일 경우, 이를 조각난 공간이라 함

데이터파일을 생성시 초기 포맷된 공간으로 테이블스페이스의 데이터 증가시 데이터파일의 여유공간을 사용하여 세그먼트에 익스텐트를 할당하게됨

TmaxTibero

2. 사용자 관리

사용자 관리

- **사용자 관리**
  - Tibero가 설치되면 아래와 같은 사용자 계정이 자동으로 생성됨
  - 데이터베이스 관리자는 기본 계정 이외에 업무에 필요한 사용자 계정을 추가 생성하고, 관리하여야 함

계정	설명
SYS	데이터베이스 관리를 위한 계정으로서 시스템 패키지, 동의어, 사용자, 역할, 가상 테이블, 시퀀스, 동적 뷰 등을 생성하고 관리한다.
SYSCAT	데이터베이스 관리를 위한 정적 카탈로그 뷰를 생성하고 관리하는 계정이다.
OUTLN	동일한 SQL을 수행할 때 항상 같은 질의 플랜(plan)으로 수행될 수 있게 관련 힌트(hint)를 저장하는 등의 일을 하는 계정이다.
SYSGIS	GIS(Geographic Information System)와 관련된 테이블 생성 및 관리를 하는 계정이다.
TIBERO	CONNECT, RESOURCE, DBA 역할이 부여된 샘플 사용자 계정이다.
TIBERO1	CONNECT, RESOURCE, DBA 역할이 부여된 샘플 사용자 계정이다.

사용자 관리

- **사용자 계정**
  - Tibero 내부의 데이터에 접근하기 위해서는 사용자 계정(Account)이 필요하다.

  - 각 계정은 패스워드(Password)를 통해 보안이 유지된다
    - 패스워드는 사용자 계정을 생성할 때 설정됨
    - 사용자 계정이 생성된 이후에 변경할 수 있음
    - 패스워드는 데이터 사전(Data Dictionary)에 암호화된 형태로 저장됨

  - 하나의 사용자 계정은 하나의 스키마를 가지며 스키마의 이름은 사용자의 이름과 같다.
    (** 스키마(Schema) : 테이블, 뷰, 인덱스 등의 스키마 객체(Schema object)의 묶음)

사용자 관리 (계속)

- **사용자 생성**
  - 사용자를 새로 생성하거나 변경 혹은 삭제하기 위해서는 DBA권한을 가진 사용자로 접속해야 함.
  - Tibero에서는 기본적으로 SYS라는 DBA 사용자를 제공
  - 사용자 생성은 아래와 같이 CREATE USER 명령을 사용하여 할 수 있다.

```
SQL> CREATE USER tibero
       IDENTIFIED BY tmax
       DEFAULT TABLESPACE MY_SPACE;
```

  - 사용자를 생성 후 DB에 접속할 수 있는 CONNECT 권한을 부여하지 않으면 DB에 접속할 수 없다.
  - 만약 Default tablespace 를 지정하지 않으면 database 의 default tablespace 로 object 가 저장이 되고, Database 의 default tablespace 가 지정되지 않았다면 USR tablespace 로 저장이 됩니다.

예) 현재 유저의 Default tablespace 확인.
→ select default_tablespace from user_users;

예) 'tiberouser01' 유저의 default tablespace 를 ts_tiberouser 로 변경.
→ alter user tiberouser01 default tablespace ts_tiberouser;

예) emp 테이블의 tablespace 를 ts_tiberouser 로 변경.
→ alter table emp move tablespace ts_tiberouser;

사용자 관리 (계속)

- **사용자 정보 변경**
  - 비밀번호 변경

```
SQL> ALTER USER tibero
       IDENTIFIED BY tmax123;
```

  - 사용자 기본 테이블스페이스 변경

```
SQL> ALTER USER tibero
       DEFAULT TABLESPACE TMAX_DAT;
```

- **사용자 삭제**

```
SQL>DROP USER park CASCADE;
```

- **사용자 정보 조회**
  - 사용자에 대한 일반적인 관리에 도움을 주기 위해 아래와 같은 시스템 뷰 제공.

뷰	설명
ALL_USERS	•데이터베이스 내의 모든 사용자에 대한 기본적인 정보
DBA_USERS	•데이터베이스 내의 모든 사용자에 대한 자세한 정보
USER_USERS	•현재 사용자에 대한 정보

사용자 관리 (계속)

- **프로파일**
  - 데이터베이스 사용자의 패스워드 관리 정책을 지정할 수 있다.
  - 이 처럼, 프로파일은 사용자 패스워드 관리 정책을 다양하게 생성하고 각각의 사용자에게 특정 정책을 사용하도록 지정함으로써 사용자별로 그룹화된 패스워드 정책을 관리할 수 있는 기능을 제공한다.

```
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.
```

사용자 관리 (계속)

- **프로파일 종류**
  - 프로파일을 생성, 변경할 때는 세부적인 파라미터를 지정할 수 있다. 아래 표는 각 파라미터에 대한 상세 설명이다. 참고로 각 파라미터의 기본 값은 데이터베이스를 생성할 때 함께 생성된 'DEFAULT' 프로파일에 의해 결정되며 이 기본값들은 DBA_PROFILES 뷰를 통해 확인할 수 있다.

구성요소	설명
FAILED_LOGIN_ATTEMPTS	잘못된 패스워드로 로그인 시도할 경우 사용자 계정를 잠글때까지 로그인 시도 허용 횟수를 지정한다.
LOGIN_PERIOD	마지막 로그인 후 지정된 시간이 지나면 계정인 잠긴다.
PASSWORD_LIFE_TIME	패스워드 만료 기간을 설정한다. 숫자와 수식 두가지 형태로 지정할 수 있다. 숫자로 지정할 경우 단위는 일(day)이다. 예를 들어 30으로 지정하면 패스워드 만료 일자가 30일 후가 된다. 해당 값을 1일 미만으로 지정하는 등 특별한 용도로 사용 하기 위해 수식을 사용할 수 있다. 예를 들어 1/1440으로 지정하면 패스워드 만료 일자가 1분 후가 된다.
PASSWORD_REUSE_TIME	패스워드 재사용 금지 기간을 설정함. 숫자와 수식 두가지 형태로 지정할 수 있다. 예) 해당 값을 30으로 설정하면 30일동안 동일한 패스워드로 다시 변경할 수 없다.
PASSWORD_REUSE_MAX	설정된 갯수만큼 최근 변경한 패스워드는 재사용할 수 없다. 예) 10으로 지정하면 현재 패스워드를 재사용하기 위해서는 10회 이상 다른 값으로 먼저 패스워드를 변경해야 한다.

사용자 관리 (계속)

- **프로파일 종류(계속)**

구성요소	설명
PASSWORD_LOCK_TIME	패스워드 오류 횟수 초과로 계정이 잠 금 상태가 되었을 때 자동으로 잠금 상태를 해제하는 기간을 설정한다. 숫자와 수식 두가지 형태로 지정할 수 있다. 예) 1/1440으로 지정하면 1분후 자동으로 잠금 상태가 해제된다.
PASSWORD_GRACE_TIME	패스워드 사용기간(PASSWORD_LIFE_TIME) 만료 후 패스워드 만료 경고를 보내는 기간을 설정한다. 숫자와 수식 두가지 형태로 지정할 수 있다. 경고를 보내는 기간은 패스워드 사용기간 만료후 첫 접속한 시점부터 계산한다. 예) PASSWORD_LIFE_TIME을 30, PASSWORD_GRACE_TIME을 3으로 설정하면 30일이 지난 후 첫 접속부터 3일간 패스워드 만료 경고가 나타나고 다시 3일이 지나면 계정이 잠금상태가 된다. 패스워드 만료 경고는 tbsql에서만 동작하며 다른 방식으로 접속하는 경우(OCI, JDBC등) 패스워드 만료 에러를 반환한다.
PASSWORD_VERIFY_FUNCTION	패스워드를 변경할 때 패스워드 문자열을 검사하 여 유효성을 확인하는 PSM 함수를 지정한다. PSM 함수는 기본으로 들어있는 함수를 지정하거나 다른 함수를 정의해서 지정할 수 있다. 아무것도 지정하지 않으면 데이터베이스를 생성할 때 미리 만들어진 NULL_VERIFY_FUNCTION이라는 함수를 기본으로 사용한

사용자 관리 (계속)

- **프로파일 적용**
  - 프로파일을 변경 : PROF1이라는 profile의 패스워드 오류 횟수 초과시 계정 잠금 상태가 1일간 지속되며 한 번 사용한 패스워드는 30개의 다른 패스워드를 사용한 이후에 다시 사용할 수 있다.

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

  - 프로파일 삭제

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.

```
SQL> DROP PROFILE prof CASCADE;
```

  - 프로파일 지정

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

  - 프로파일 변경

```
SQL> ALTER USER Peter PROFILE default;
```

권한(Privilege) 관리

- **권한(Privilege)**
  - 데이터베이스나 데이터베이스 오브젝트에 접근하여 SQL문을 실행하기 위한 권리
  - 권한 종류
    - 스키마 오브젝트 권한
    - SYSTEM 권한
  - 특권을 부여하는 예

```
SQL> conn Peter/abcdef ... ① ... Connected.
SQL> CREATE TABLE EMPLOYEE
                    (ID NUMBER,
                      EMPLOYEE_NAME VARCHAR(20),
                     ADDRESS VARCHAR(50)); ... ② ... Created.
SQL> GRANT SELECT ON EMPLOYEE TO Smith; ... ③ ... Granted.
```

① Peter 라는 사용자 계정으로 데이터베이스에 접속한다.

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

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

이제 사용자 Smith는 Peter의 EMPLOYEE 테이블을 조회할 수 있다.

권한(Privilege) 관리 (계속)

- **스키마 오브젝트 권한 (Schema Object Privilege)**
  - 오브젝트에 대해 실행할 수 있는 명령어 사용을 위한 권한
  - WITH GRANT OPTION을 이용한 권한 부여
    - 권한을 부여 받은 사용자가 부여 받은 스키마 오브젝트 권한을 다른 사용자에게 다시 부여가능

```
SQL> GRANT SELECT, UPDATE (PROD_NAME , PROD_DATE ) ON PRODUCT
        TO tibero WITH GRANT OPTION;
```

  - 스키마 오브젝트 권한 종류

권한	설명
ALTER	• 오브젝트 특성을 변경
DELETE	• 테이블에서 row를 삭제
EXECUTE	• 함수, 프로시저를 실행
INDEX	• 테이블에 대한 인덱스를 생성
INSERT	• 테이블에 row를 삽입
REFERENCES	• 테이블을 참조하는 제약조건을 생성
SELECT	• 테이블에 대하여 질의
UPDATE	• 테이블의 row를 갱신

권한(Privilege) 관리 (계속)

- **시스템 권한 (System Privilege)**
  - 데이터베이스를 관리하는 데 필요한 시스템 명령어 사용을 위한 권한
  - 시스템 권한은 기본적으로 SYS 사용자가 소유하고 있으며, 다른 사용자에게 부여 가능
  - WITH ADMIN OPTION을 이용 권한 부여
    - 권한을 부여 받은 사용자가 부여 받은 시스템 권한을 다른 사용자에게 다시 부여 가능

```
SQL> GRANT SELECT ANY TABLE TO tibero
       WITH ADMIN OPTION;
```

  - 스키마 오브젝트 권한 종류

권한	설명
ALTER SYSTEM	• ALTER SYSTEM 명령을 사용 수 있다.
CREATE SESSION	• DB에 session을 생성. 즉 로그인을 할 수 있다.
CREATE TABLESPACE	• TABLESPACE를 생성
ALTER TABLESPACE	• TABLESPACE를 변경
DROP TABLESPACE	• TABLESPACE를 삭제
CREATE USER	• USER를 생성
ALTER USER	• USER 정보를 변경
DROP USER	• USER를 삭제

권한(Privilege) 관리 (계속)

권한	설명
CREATE TABLE	• 자신의 스키마에 TABLE을 생성
CREATE ANY TABLE	• 임의의 스키마에 TABLE을 생성
ALTER ANY TABLE	• 임의의 스키마에 있는 TABLE을 변경
DROP ANY TABLE	• 임의의 스키마에 있는 TABLE을 삭제
COMMENT ANY TABLE	• 임의의 스키마에 있는 TABLE에 주석 추가
SELECT ANY TABLE	• 임의의 스키마에 있는 TABLE에 질의
INSERT ANY TABLE	• 임의의 스키마에 있는 TABLE에 row를 삽입
UPDATE ANY TABLE	• 임의의 스키마에 있는 TABLE의 row를 갱신
DELETE ANY TABLE	• 임의의 스키마에 있는 TABLE의 row를 삭제
CREATE ANY INDEX	• 임의의 스키마에 있는 TABLE에 대한 인덱스를 생성
ALTER ANY INDEX	• 임의의 스키마에 있는 TABLE에 있는 인덱스를 수정
DROP ANY INDEX	• 임의의 스키마에 있는 TABLE에 있는 인덱스를 삭제
CREATE SYNONYM	• 자신의 스키마에 동의어를 생성
CREATE ANY SYNONYM	• 임의의 스키마에 동의어를 생성
DROP ANY SYNONYM	• 임의의 스키마에 있는 동의어를 삭제

권한(Privilege) 관리 (계속)

권한	설명
SYSDBA	• SHUTDOWN, ALTER DATABASE, CREATE DATABASE, ARCHIVELOG, RECOVERY 명령을 내릴 수 있다
CREATE PUBLIC SYNONYM	• PUBLIC 스키마에 동의어를 생성
DROP PUBLIC SYNONYM	• PUBLIC 스키마에 동의어를 삭제
CREATE VIEW	• 자신의 스키마에 VIEW를 생성
CREATE ANY VIEW	• 임의의 스키마에 VIEW를 생성
DROP ANY VIEW	• 임의의 스키마에 있는 VIEW를 삭제
CREATE SEQUENCE	• 자신의 스키마에 SEQUENCE를 생성
CREATE ANY SEQUENCE	• 임의의 스키마에 SEQUENCE를 생성
ALTER ANY SEQUENCE	• 임의의 스키마에 있는 SEQUENCE를 변경
DROP ANY SEQUENCE	• 임의의 스키마에 있는 SEQUENCE를 삭제
SELECT ANY SEQUENCE	• 임의의 스키마에 있는 SEQUENCE를 질의
CREATE ROLE	• 롤을 생성
DROP ANY ROLE	• 롤을 삭제
GRANT ANY ROLE	• 임의의 롤을 부여

권한(Privilege) 관리 (계속)

권한	설명
ALTER ANY ROLE	• 롤을 수정
ALTER DATABASE	• DATABASE를 변경
CREATE PROCEDURE	• 자신의 스키마에 psm procedure를 생성
CREATE ANY PROCEDURE	• 임의의 스키마에 psm procedure를 생성
ALTER ANY PROCEDURE	• 임의의 스키마에 있는 psm procedure를 변경
DROP ANY PROCEDURE	• 임의의 스키마에 있는 psm procedure를 삭제
EXECUTE ANY PROCEDURE	• 임의의 스키마에 있는 psm procedure를 실행
CREATE TRIGGER	• 자신의 스키마에 있는 trigger를 생성
CREATE ANY TRIGGER	• 임의의 스키마에 있는 trigger를 생성
ALTER ANY TRIGGER	• 임의의 스키마에 있는 trigger를 변경
DROP ANY TRIGGER	• 임의의 스키마에 있는 trigger를 삭제
GRANT ANY OBJECT PRIVILEGE	• 모든 스키마 객체에 대한 특권을 가진다.
GRANT ANY PRIVILEGE	• 모든 특권을 다 부여할 수 있다.

권한(Privilege) 관리 (계속)

- **권한 회수(REVOKE)**
  - 스키마 오브젝트 권한 회수

```
SQL> REVOKE SELECT, UPDATE (EMPLOYEE_NAME, ADDRESS) ON EMPLOYEE
        FROM tibero;
```

  - 시스템 권한 회수

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

롤(Role) 관리

- **롤 (Role)**
  - 여러 권한을 모아놓은 집합이며, 하나의 단위로서 사용자에게 부여
  - 롤을 생성하거나 수정, 부여하기 위해서는 그에 맞는 권한이 필요

```
SQL>  CREATE ROLE APP_USER;
SQL>  GRANT CREATE SESSION TO APP_USER;
```

```
SQL>  CREATE ROLE APP_USER2;
SQL>  GRANT SELECT, INSERT ON tibero.prduct TO APP_USER2;
```

  - 롤 부여

```
SQL> GRANT APP_USER TO tibero;
```

```
SQL> GRANT APP_USER2 TO tibero;
```

  - WITH ADMIN OPTION
    - 롤을 부여 받은 사용자가 부여 받은 롤을 다른 사용자에게 다시 부여 가능

```
SQL> GRANT APP_USER TO tibero WITH ADMIN OPTION;
```

  - 롤 취소

```
SQL> REVOKE APP_USER FROM tibero;
```

롤(Role) 관리 (계속)

- **Tibero 에서 미리 정의된 롤(ROLE)**

롤	설명
CONNECT	• CREATE SESSION
RESOURCE	• CREATE PROCEDURE • CREATE SEQUENCE • CREATE TABLE • CREATE TRIGGER
DBA	• 모든 시스템 권한이 WITH ADMIN OPTION으로 부여됨

- **SET ROLE 명령**
  - 동적으로 변경 가능 롤 변경 가능

```
SET ROLE APP_USER, TMAX;          /* APP_USER, TMAX 롤을 켬 */
SET ROLE ALL EXCEPT TMAX;         /* TMAX 를 제외한 모든 롤을 켬 */
SET ROLE ALL;                     /* 모든 롤을 켬 */
SET ROLE NONE;                    /* 모든 롤을 끔 */
```

롤(Role) 관리 (계속)

- **권한과 롤(ROLE) 정보 조회**

뷰	설명
DBA_ROLES	•모든 롤에 대한 정보
DBA_ROLE_PRIVS	•사용자나 다른 롤에 부여된 모든 롤의 정보
USER_ROLE_PRIVS	•현재 사용자나 PUBLIC 사용자에 부여된 롤의 정보
DBA_SYS_PRIVS	•사용자에게 부여된 모든 권한의 정보
USER_SYS_PRIVS	•현재 사용자에게 부여된 권한의 정보
DBA_TBL_PRIVS	•데이터베이스 내의 모든 스키마 오브젝트 권한의 정보
USER_TBL_PRIVS	•현재 사용자가 가지고 있는 모든 스키마 오브젝트 권한의 정보
ALL_TBL_PRIVS	•현재 사용자가 가지고 있는 모든 스키마 오브젝트 권한과 모든 사용자가 사용할 수 있게 공개된 모든 스키마 오브젝트 권한의 정보
DBA_COL_PRIVS	•데이터베이스 내의 모든 스키마 오브젝트 권한중 스키마 오브젝트의 특정 컬럼에 부여된 권한의 정보
USER_COL_PRIVS	•현재 사용자가 가지고 있는 스키마 오브젝트 권한중 스키마 오브젝트의 특정 컬럼에 부여된 권한의 정보
ALL_COL_PRIVS	•현재 사용자가 가지고 있는 스키마 오브젝트 권한 혹은 모든 사용자가 사용할 수 있게 공개된 모든 스키마 오브젝트 권한중 스키마 오브젝트의 특정 컬럼에 부여된 권한의 정보.

롤(Role) 관리 (계속)

뷰	설명
USER_COL_PRIVS_MADE ALL_COL_PRIVS_MADE	•현재 사용자 소유의 오브젝트중 특정 컬럼에 부여된 권한의 정보 •현재 사용자가 만든 권한중 스키마 오브젝트의 특정 컬럼에 부여된 권한의 정보
USER_COL_PRIVS_RECD ALL_COL_PRIVS_RECD	•현재 사용자에게 부여된 모든 스키마 오브젝트 권한중 스키마 오브젝트의 특정 컬럼에 부여된 권한의 정보 •현재 사용자또는 PUBLIC 사용자에게 부여된 모든 스키마 오브젝트 권한중 스키마 오브젝트의 특정 컬럼에 부여된 권한의 정보

예) Role 조회

```
/*=================================================================
 User에게 부여된 System Privileges 조회
=================================================================*/

SET PAGESIZE 200
COL GRANTEE FOR A15
COL PRIVILEGE FOR A40

SELECT * FROM DBA_SYS_PRIVS WHERE GRANTEE='SYS';

GRANTEE         PRIVILEGE                                ADMIN_OPTION
--------------- ---------------------------------------- ------------
SYS             ALTER SYSTEM                             YES
SYS             CREATE SESSION                           YES
SYS             ALTER SESSION                            YES
SYS             CREATE TABLESPACE                        YES
SYS             ALTER TABLESPACE                         YES
SYS             DROP TABLESPACE                          YES
SYS             CREATE USER                              YES
SYS             ALTER USER                               YES
SYS             DROP USER                                YES
SYS             ALTER ROLLBACK SEGMENT                   YES
SYS             CREATE TABLE                             YES
SYS             CREATE ANY TABLE                         YES
..... 내용 중략 ...

105 rows selected.
```

```
/*=================================================================
 User 에게 부여된 Role 조회
=================================================================*/

SET PAGESIZE 200
COL GRANTEE FOR A20
COL GRANTED_ROLE FOR A30

SELECT GRANTEE, GRANTED_ROLE
  FROM DBA_ROLE_PRIVS
 WHERE GRANTEE IN ('SYS');

GRANTEE              GRANTED_ROLE
-------------------- ------------------------------
SYS                  DBA
SYS                  CONNECT
SYS                  RESOURCE
SYS                  SELECT_CATALOG_ROLE
SYS                  HS_ADMIN_ROLE

5 rows selected.
```


TmaxTibero

예) Role 조회

```
/*=============================================================================
 Role 에 부여된 Privileges 조회
=============================================================================*/

SET PAGESIZE 200
COL GRANTEE FOR A20
COL PRIVILEGE FOR A40

SELECT GRANTEE, PRIVILEGE
  FROM DBA_SYS_PRIVS
 WHERE GRANTEE IN ('RESOURCE');

GRANTEE              PRIVILEGE
-------------------- ----------------------------------------
RESOURCE             CREATE TABLE
RESOURCE             CREATE SEQUENCE
RESOURCE             CREATE PROCEDURE
RESOURCE             CREATE TRIGGER

4 rows selected.
```

```
/*=============================================================================
 Role 에 부여된 Privileges 조회
=============================================================================*/

SET PAGESIZE 200
COL GRANTEE FOR A20
COL PRIVILEGE FOR A40

SELECT GRANTEE, PRIVILEGE
  FROM DBA_SYS_PRIVS
 WHERE GRANTEE IN ('CONNECT');

GRANTEE              PRIVILEGE
-------------------- ----------------------------------------
CONNECT              CREATE SESSION

1 row selected.
```

예) 사용자 생성 및 권한 부여

```
SQL> /* TABLESPACE 생성 */
SQL> CREATE TABLESPACE TBS_EDU1
  2 DATAFILE
  3   'TBS_EDU1_01.dtf' SIZE 10M AUTOEXTEND OFF,
  4   'TBS_EDU1_02.dtf' SIZE 10M AUTOEXTEND ON NEXT 640K MAXSIZE UNLIMITED
  5 NOLOGGING
  6 ONLINE
  7 EXTENT MANAGEMENT LOCAL AUTOALLOCATE
  8 --BLOCKSIZE 8K
  9 SEGMENT SPACE MANAGEMENT AUTO
 10 --FLASHBACK ON;

Tablespace 'TBS_EDU1' created.

SQL> CREATE TABLESPACE TBS_EDU2
  2 DATAFILE
  3   'TBS_EDU2_01.dtf' SIZE 10M AUTOEXTEND OFF,
  4   'TBS_EDU2_02.dtf' SIZE 10M AUTOEXTEND ON NEXT 640K MAXSIZE UNLIMITED
  5 NOLOGGING
  6 ONLINE
  7 EXTENT MANAGEMENT LOCAL AUTOALLOCATE
  8 --BLOCKSIZE 8K
  9 SEGMENT SPACE MANAGEMENT AUTO
 10 --FLASHBACK ON;

Tablespace 'TBS_EDU2' created.
```

예) 사용자 생성 및 권한 부여

```
SQL> CREATE TEMPORARY TABLESPACE TEMP1
   2 TEMPFILE
   3   'TEMP1_01.dtf' SIZE 10M AUTOEXTEND ON NEXT 640K MAXSIZE UNLIMITED,
   4   'TEMP1_02.dtf' SIZE 10M AUTOEXTEND ON NEXT 128K MAXSIZE UNLIMITED
   5 --TABLESPACE GROUP ''
   6 EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M
   7 --FLASHBACK ON;

SQL> /* 유저 생성 */
SQL> CREATE USER EDU1 IDENTIFIED BY "EDU1"
   2 DEFAULT TABLESPACE TBS_EDU1
   3 TEMPORARY TABLESPACE TEMP1
   4 PROFILE DEFAULT
   5 ACCOUNT UNLOCK;

User 'EDU1' created.
```

```
SQL> /* 유저 조회 */
SQL> SET LINESIZE 100
SQL> COL USERNAME FOR A10
SQL> COL DEFAULT_TEMP_TABLESPACE FOR A25
SQL> COL PROFILE FOR A10
SQL> COL DEFAULT_TABLESPACE FOR A20
SQL> COL ACCOUNT_STATUS FOR A15
SQL>
SQL> SELECT USERNAME, PROFILE, ACCOUNT_STATUS
   2       , DEFAULT_TABLESPACE, DEFAULT_TEMP_TABLESPACE
   3 FROM DBA_USERS
   4 WHERE USERNAME = 'EDU1';

USERNAME   PROFILE    ACCOUNT_STATUS  DEFAULT_TABLESPACE   DEFAULT_TEMP_TABLESPACE
---------- ---------- --------------- -------------------- -------------------------
EDU1       DEFAULT    OPEN            TBS_EDU1             TEMP1

1 row selected.
```

예) 사용자 생성 및 권한 부여

```
SQL> /* ROLL 부여 */
SQL> -- 2 Roles for EDU1
SQL> GRANT CONNECT TO EDU1;

Granted.

SQL> GRANT RESOURCE TO EDU1;

Granted.

SQL> ALTER USER EDU1 DEFAULT ROLE ALL;

User 'EDU1' altered.

SQL>
SQL> SET PAGESIZE 200
SQL> COL GRANTEE FOR A20
SQL> COL GRANTED_ROLE FOR A30
SQL>
SQL> SELECT GRANTEE, GRANTED_ROLE
   2   FROM DBA_ROLE_PRIVS
   3  WHERE GRANTEE = 'EDU1';

GRANTEE              GRANTED_ROLE
-------------------- ------------------------------
EDU1                 CONNECT
EDU1                 RESOURCE

2 rows selected.
```

```
SQL>
SQL>
SQL>
SQL>
SQL>
SQL> /* TABLESPACE QUOTA for EDU1 */
SQL> ALTER USER EDU1 QUOTA 5M        ON TBS_EDU1;

User 'EDU1' altered.

SQL> ALTER USER EDU1 QUOTA UNLIMITED ON TBS_EDU2;

User 'EDU1' altered.

SQL>
SQL> COL TABLESPACE_NAME FOR A15
SQL> SELECT * FROM DBA_TS_QUOTAS;

TABLESPACE_NAME USERNAME        BYTES  MAX_BYTES     BLOCKS MAX_BLOCKS DROPPED
--------------- ---------- ---------- ---------- ---------- ---------- -------
TBS_EDU1        EDU1                0    5242880          0        640 NO
TBS_EDU2        EDU1                0         -1          0         -1 NO

2 rows selected.
```

3. 오브젝트(Obejct) 관리

1. 스키마 객체(Schema Objects)

스키마 객체

▪스키마는 집합체로써 각종 오브젝트를 포함하며, 동일한 이름의 유저에 의해 소유된다. 본 문서에서는 다음과 같은 오브젝트에 대해 학습하고자 한다.

- TABLE
- SYNONYM
- INDEX
- VIEW
- SEQUENCE
- JOB

▪스키마 오브젝트는 논리적인 데이터 저장 구조로써 디스크의 물리적 파일과 1대1 대응을 하지 않고, 논리적 구조인 테이블스페이스에 저장된다.

스키마 객체의 이름

- SQL 문장에서 스키마 객체의 이름을 명시할 때는 '따옴표 있는 식별자' 또는 '따옴표 없는 식별자'를 사용한다.

- 따옴표 없는 식별자는 대소문자를 구별하지 않으며, 모두 대문자로 간주되어 처리된다. 따옴표 있는 식별자는 대소문자를 구분한다.

- 예를 들어 다음의 경우는 모두 서로 다른 식별자이다.

서로 다른 식별자	서로 같은 식별자
department "department" "Department"	department DEPARTMENT "DEPARTMENT"

스키마 객체의 이름

▪식별자 기술 규칙은 다음과 같다

구분	내용
예약어	예약어는 따옴표 없는 식별자가 될 수 없다. 따옴표를 적용하여 예약어를 식별자로 만들 수 있으나, 권장하지는 않는다. 예약어는 SQL참조 안내서의 “Appendix A. 예약어”를 참고한다. SQL> create table table(c1 number); TBR-7207: New identifier required: 'TABLE' is a reserved word. at line 1, column 14 of null: create table table(c1 number) SQL> CREATE TABLE "TABLE"(C1 NUMBER); Table 'TABLE' created.

스키마 객체의 이름

▪식별자 기술 규칙은 다음과 같다

구분	내용
숫자,한글,기호	따옴표 없는 식별자는 알파벳, 한글, 숫자, 밑줄(_), $, #만 사용할 수 있다. 다만, 숫자, '$', '#'는 첫 글자로 올 수 없다. SQL> create table 다(ㄱ number); SQL> desc 다 COLUMN_NAME TYPE CONSTRAINT ---------- ---------- ---------- ㄱ NUMBER SQL> insert into 다(ㄱ) values(10); SQL> select ㄱ from 다; ㄱ ---------- 10

스키마 객체의 이름

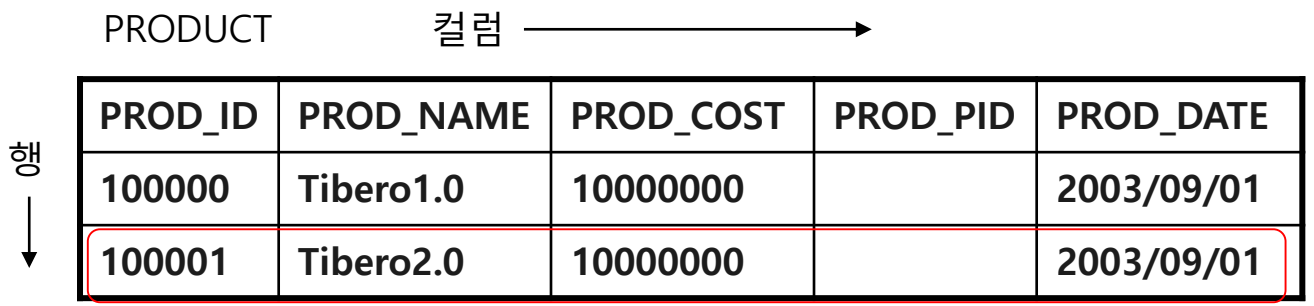
- 하나의 테이블 내의 서로 다른 컬럼은 동일한 이름을 가질 수 없다.
- 서로 다른 테이블에 속해 있는 컬럼은 동일한 이름을 가질 수 있다.
- 한 패키지 내의 서로 다른 프러시저나 함수는 인자의 개수나 데이터 타입이 다른 경우에 한해서 동일한 이름을 가질 수 있다.
- 하나의 네임스페이스 안에서 서로 다른 두 객체가 동일한 이름을 가질 수 없다.

구분	네임스페이스	종류
스키마 객체	하나의 네임스페이스	테이블, 뷰, 실체화 뷰, 시퀀스, 동의어, 패키지, 패키지에 포함되지 않은 함수 및 프러시저
	독립적인 네임스페이스	인덱스 , 트리거, 서로 다른 스키마의 네임스페이스는 공유되지 않는다. 그러므로 서로 다른 스키마에 있는 두 개의 테이블은 같은 이름을 가질 수 있다.
일반 객체	독립적인 네임스페이스	사용자 역할, 공유 동의어, 테이블 스페이스

2. 테이블(TABLE)

테이블

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



PROD_ID	PROD_NAME	PROD_COST	PROD_PID	PROD_DATE
100000	Tibero1.0	10000000		2003/09/01
100001	Tibero2.0	10000000		2003/09/01

테이블

- 테이블에 사용되는 컬럼 타입
  - 문자형
    - CHAR (n) : 고정길이 문자 데이터. 최대 2000byte 까지 선언.
    - VARCHAR(n) : 가변길이 문자 데이터. 최대 65,532byte 까지 선언.
    - NCHAR (n) : 유니코드 문자열을 저장, 고정길이 문자 데이터. 최대 2000byte 까지 선언.
    - NVARCHAR (n) : 유니코드 문자열을 저장, 가변길이 문자 데이터. 최대 65,532byte 까지 선언.
    - RAW : 임의의 바이너리 데이터를 저장하기 위한 타입. 최대 2000byte 까지 선언.
    - LONG : VARCHAR와 RAW 타입의 최대 길이를 2G까지 늘린 타입. 일반 문자열 저장
    - LONG RAW: VARCHAR와 RAW 타입의 최대 길이를 2G까지 늘린 타입. 바이너리 데이터 저장
  - 숫자형
    - NUMBER (p,s) : 가변길이 숫자 데이터. p는 정밀도 자리수, s는 소수점 이하 자리수
  - 날짜형
    - DATE : 날짜 및 초 단위까지의 시간을 선언.
    - TIME : 초 단위 소수점 9자리까지 시간을 선언.
    - TIMESTAMP: 날짜와 초 단위 소수점 9자리까지의 시간을 선언.
  - 대용량 객체형
    - CLOB : 최대 4G까지 선언. 일반 문자열 저장
    - BLOB : LONG과 LONG RAW타입을 확장한 데이터 타입. 최대 4G까지 선언. 바이너리 저장.
    - XMLTYPE : W3C 국제표준 형식으로, XML 데이터를 저장하며 내부적으로 CLOB 형식으로 저장.

테이블

▪ 테이블의 생성

➢ Creating Tables

- CREATE TABLE 권한 필요.
- 다른 사용자의 스키마에서 생성시 CREATE ANY TABLE 권한이 필요

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

- **TABLESPACE : 테이블을 생성하면서 이 테이블의 데이터가 저장될 저장소 지정.**
- **PCTFREE : 테이블의 각 데이터 블록에 있는 공간 비율로 행을 갱신하기 위해 예약해 두는 공간의 비율(%.)**
- **INITRANS : 테이블에 할당된 데이터 블록에 동시에 엑세스 가능한 트랜잭션의 초기값**

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

테이블

▪ Temporary 테이블의 생성

➢ Creating TEMPORARY Tables

- Transaction or Session 동안에만 Data가 유지
- Data는 Session상에서만 Display
- DML작업 시 Redo를 생성하지 않음
- DML 작업 시 ROLLBACK을 생성하지 않음.
- dba_tables.(table_name, temporary, duration)

```
CREATE GLOBAL TEMPORARY TABLE emp_temp
      (eno NUMBER, ename VARCHAR(20), sal NUMBER)
ON COMMIT DELETE ROWS;
```

→ **Transaction 종료시**

```
CREATE GLOBAL TEMPORARY TABLE emp_temp
      (eno NUMBER, ename VARCHAR(20), sal NUMBER)
ON COMMIT PRESERVE ROWS;
```

→ **Session 종료시**

테이블

테이블의 변경

- 다음 경우에 테이블을 변경한다.
  - 컬럼 추가, 삭제
  - Online중 Table이동 및 Reorganization
  - 무결성 제약 조건 추가
  - 기존 컬럼의 정의(데이타 유형, 길이, 디폴트, NOT NULL 제약조건) 수정
  - 데이타 블럭의 영역 사용 매개변수 (PCTFREE) 수정
  - 트랜잭션 엔트리 항목(INITRANS) 수정
  - 테이블에 관련된 무결성 제약조건 혹은 트리거의 활성화/비활성화
  - 테이블에 관련된 무결성 제약조건 삭제

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

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

- 컬럼 명 변경

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

- 저장 영역 변경

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

테이블

▪테이블 삭제

➢ DROP 구문 사용시 테이블의 모든 행이 삭제되고 사용된 공간이 해제.

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

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

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

➢ 테이블 삭제 구문은 자동 커밋이므로 ROLEBACK이 불가능.

테이블

테이블 정보 조회

뷰	설명
DBA_TABLES USER_TABLES ALL_TABLES	• 티베로내의 모든 테이블에 대한 정보 • 현재 유저에 속한 테이블에 대한 정보 • 유저가 접근 가능한 테이블에 대한 정보
DBA_TBL_COLUMNS USER_TBL_COLUMNS ALL_TBL_COLUMNS	• 티베로내의 모든 테이블, 뷰에 속한 컬럼에 대한 정보 • 현재 유저에 속한 테이블, 뷰에 속한 컬럼에 대한 정보 • 유저가 접근 가능한 테이블, 뷰에 속한 컬럼에 대한 정보

테이블

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

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

```
CREATE TABLE TEMP_PROD (
                prod_id number(6),
                prod_name varchar(50) constraint prod_name_nn not null,
                prod_cost  varchar(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 종류 :**
    - **Not Null : 해당 컬럼은 NULL값을 가질 수 없음. 테이블 레벨의 제약 조건 지정은 불가능.**
    - **Unique : 한 테이블 내에서 해당 컬럼은 동일한 값을 가질 수 없음. NULL 값은 여러 행에 입력가능.**
    - **Primary Key : 무결성 제약조건과 고유키 무결성 제약조건을 결합한 개념. NULL 사용 불가**
    - **Foreign Key : 다른 테이블이나 자신 테이블의 Primary key나 Unique key를 참조할 때 사용**
    - **Check : 입력 또는 수정 값이 만족해야 할 조건 지정. 한 컬럼에 대하여 여러 개의 제약조건 지능 가능**

테이블

제약조건의 상태

- Enable : 제약조건이 활성화 되어 제약 조건을 적용.
- Disable : 제약조건이 비활성화 되어 적용되지 않음.
- Validate : 기존 저장되어 있는 데이터의 무결성을 보장.
- Novalidate : 기존 저장되어 있는 데이터의 무결성을 보장하지 않음

제약조건 상태 변경 문

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

제약조건 이름 변경

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

제약조건의 추가

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

제약조건의 삭제

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

테이블

▪ 제약조건에 대한 정보

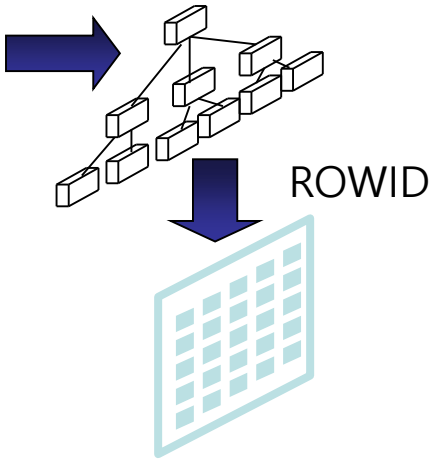
뷰	설명
DBA_CONSTRAINTS	• 티베로내의 모든 제약조건에 대한 정보
USER_CONSTRAINTS	• 현재 유저에 속한 제약조건에 대한 정보
ALL_CONSTRAINTS	• 유저가 접근 가능한 제약조건에 대한 정보
DBA_CONS_COLUMNS	• 티베로내의 모든 제약조건에 걸린 컬럼에 대한 정보
USER_CONS_COLUMNS	• 현재 유저에 속한 제약조건에 걸린 컬럼에 대한 정보
ALL_CONS_COLUMNS	• 유저가 접근 가능한 제약조건에 걸린 컬럼에 대한 정보

테이블(Index-Organized Tables)

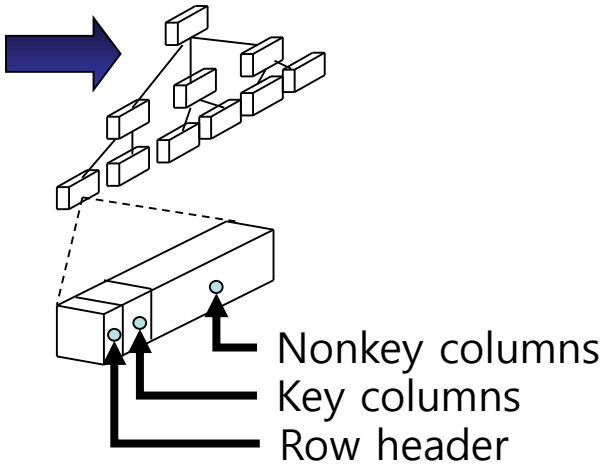
Index-Organized Tables

- IOT는 테이블의 기본 키와 다른 열 값을 포함한 하나의 B* Tree만을 유지 관리
- Index와 non-index가 같은 Segment에 있기에 Fewer I/Os
- Reduced storage requirements
- ROWID 의사 컬럼은 Primary Key기준의 Logical ROWID를 참조
- Secondary Key 생성가능

Accessing table using separate index



Accessing IOT



TmaxTibero

테이블(Index-Organized Tables)

- **Index-Organized Tables**

일반테이블	인덱스구성테이블(IOT)
ROWID는 행을 고유하게 식별	기본 키(PK)가 행을 고유하게 식별
Primary Key 선택	Primary Key 필수
Hash 에 저장 가능	Hash 에 저장 불가능

- 일반테이블은 ROWID로 행을 구별하지만 IOT는 PK로 행을 구별.
- 일반테이블의 FULL SCAN은 행이 Return되는 순서를 예측할 수 없지만, IOT는 PK값의 순서에 따라 출력.
- IOT는 Unique 제약조건을 설정 불가능.
- IOT는 일반테이블보다 저장공간이 감소.
- IOT의 Secondary 인덱스는 Primary Key값과 그것을 기반으로 하는 'Universal Rowid' 즉, Urowid를 가지고 인덱스가 생성.
- IOT는 일반 인덱스와 달리 물리적인 ROWID 정보를 가지고 있지 않고, 논리적인 유니버설 ROWID(UROWID)를 가지고 있어 빠르게 검색 가능.
- IOT는 LONG 컬럼 보유 불가능

테이블(Index-Organized Tables)

- **Index-Organized Tables**

장점	단점
같은 값을 가진 레코드들이 100% 정렬된 상태로 모여 있기 때문에 Random 액세스가 아닌 Sequential 방식으로 데이터를 액세스 할 수 있고, 이 때문에 넓은 범위를 액세스 할 때 유리하다.	데이터 입력 시 성능이 느리다
PK 컬럼 기준으로 데이터가 모여 있더라도 선행 컬럼이 '=' 조건이 아니면 조회 대상 레코드들이 서로 흩어져 많은 스캔을 유발하지만, 적어도 테이블 Random 액세스는 발생하지 않아 빠른 성능을 낼 수 있다.	**인덱스 분할(Split) 발생량의 차이**로 일반 힙 구조 테이블과 성능 차이가 클 수 있다. IOT는 인덱스 구조이므로 중간에 꽉 찬 블록에 새로운 값을 입력할 일이 종종 생기고 그럴 때 인덱스 분할(Split)이 발생한다. 그런데 **IOT가 PK 이외에 많은 컬럼을 갖는다면 리프 블록에 저장해야 할 데이터량이 늘어나 그만큼 인덱스 분할 발생빈도도 높아** 진다. 컬럼 수가 많은 테이블이라면 인덱스 스캔 효율 때문이라도 IOT 대상으로 부적합하다.
PK 인덱스를 위한 별도의 세그먼트를 생성하지 않아도 돼 저장공간을 절약하는 부수적인 이점도 있다.	**IOT에 Direct Path Insert가 작동하지 않는다.**

테이블(Index-Organized Tables)

Index-Organized Tables

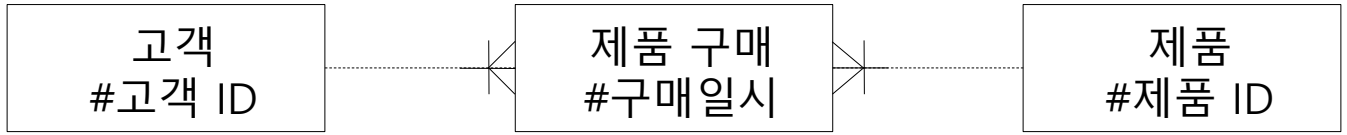
IOT 적용 대상 : 소량 데이터를 NL 조인으로 반복 Lookup 하는 테이블

- 코드성 테이블
- NL 조인에서 Inner 쪽 Lookup 테이블로서 액세스되는 동안 건건이 인덱스와 테이블 블록을 다 읽는다면 비효율적이므로 IOT로 구성하면 테이블은 반복 액세스하지 않아도 된다.
- IOT 구성 시 PK이외 속성의 크기 때문에 인덱스 높이가 증가한다면 역효과가 날 수 있으므로 이를 반드시 확인해야 한다.
- 폭이 좁고 긴(=로우 수가 많은), PK 이외의 컬럼이 거의 없는 테이블
- Between, Like 같은 조건으로 넓은 범위를 검색하는 테이블

테이블(Index-Organized Tables)

Index-Organized Tables

IOT 적용 대상 : 컬럼 수가 적고 건수가 많은 테이블



- 제품 구매 테이블은 PK(고객ID, 제품ID, 구매일시) 이외의 컬럼은 거의 없음,
- PK인덱스는 생성해야 하는데, 테이블과 거의 중복된 데이터를 갖게 되므로, IOT 로 테이블을 생성하면 데이터 중복 제거 할 수 있음.

IOT 적용 대상 : 넓은 범위를 검색하는 테이블

- 넓은 범위 검색(between, like 등으로) 테이블로써 PK 이외의 컬럼이 별로 없는 테이블
  - Non key 컬럼이 거의 없는 통계성 테이블
  - 등호(=) 조건 컬럼, 범위 조건 컬럼 순서로 PK 구성하는 것이 효과적임.

TmaxTibero

테이블(Index-Organized Tables)

Index-Organized Tables

IOT 적용 대상 : 조회 유형이 입력과 상이한 테이블

- 일자 별 실적 등록, 사업장별 실적조회를 수행하는 경우
  - 전체 사업장100곳의 일별 실적이 데이터 블록 1개의 담기는 경우, 1년 365개 블록이 사용되며, 일자별 입력, 조회는 사업장별로 이루어 지게 됨
  - 적은 블록을 액세스 하기 위해, 사업장ID 를 KEY 로 하는 IOT 를 생성하여 성능을 개선할 수 있음 (100건/1블록 일 경우, 365 개 블록 --> 4개 블록 으로 감소)

```
CREATE TABLE 영업실적(매장ID VARCHAR(6), 일자 VARCHAR(8), ...
CONSTRAINT 영업실적_PK PRIMARY KEY (매장ID, 일자) ) ORGANIZATION INDEX;
```

```
SELECT SUBSTR(일자, 1, 10) 월도, SUM(판매금액) 총판매금액, AVG(판매금액) 평균판매금액
FROM   영업실적
WHERE  매장ID = 'A00001'
AND    일자 BETWEEN '20180101' AND '20181231'
GROUP BY SUBSTR(일자, 1, 10)
```

테이블(Index-Organized Tables)

Index-Organized Tables

➢ IOT Table 생성

```
SQL> CREATE TABLE IOTA
      ( TOKEN CHAR(20),
        DOC_OID NUMBER,
        TOKEN_FREQUENCY NUMBER,
        TOKEN_OCCURCENCE_DATA VARCHAR(512),
        CONSTRAINT PK_IOTA PRIMARY KEY (TOKEN, DOC_OID)
      ) ORGANIZATION INDEX TABLESPACE MY_SPACE;


SQL> SELECT SEGMENT_NAME,SEGMENT_TYPE
   2   FROM USER_SEGMENTS
   3  WHERE SEGMENT_NAME LIKE '%IOTA%';

SEGMENT_NAME         SEGMENT_TYPE
-------------------- --------------------
PK_IOTA              INDEX
```

테이블(Index-Organized Tables)

Index-Organized Tables

➢ Dictionary Views

– DBA_TABLES

```
SQL> SELECT *
       FROM DBA_TABLES
      WHERE IOT_TYPE  IN ('IOT', 'IOT_OVERFLOW');
```

– DBA_INDEXES

```
SQL> SELECT *
       FROM DBA_INDEXES
      WHERE INDEX_NAME = 'INDEX_NAME';
```

3. 인덱스(INDEX)

인덱스

인덱스

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

인덱스

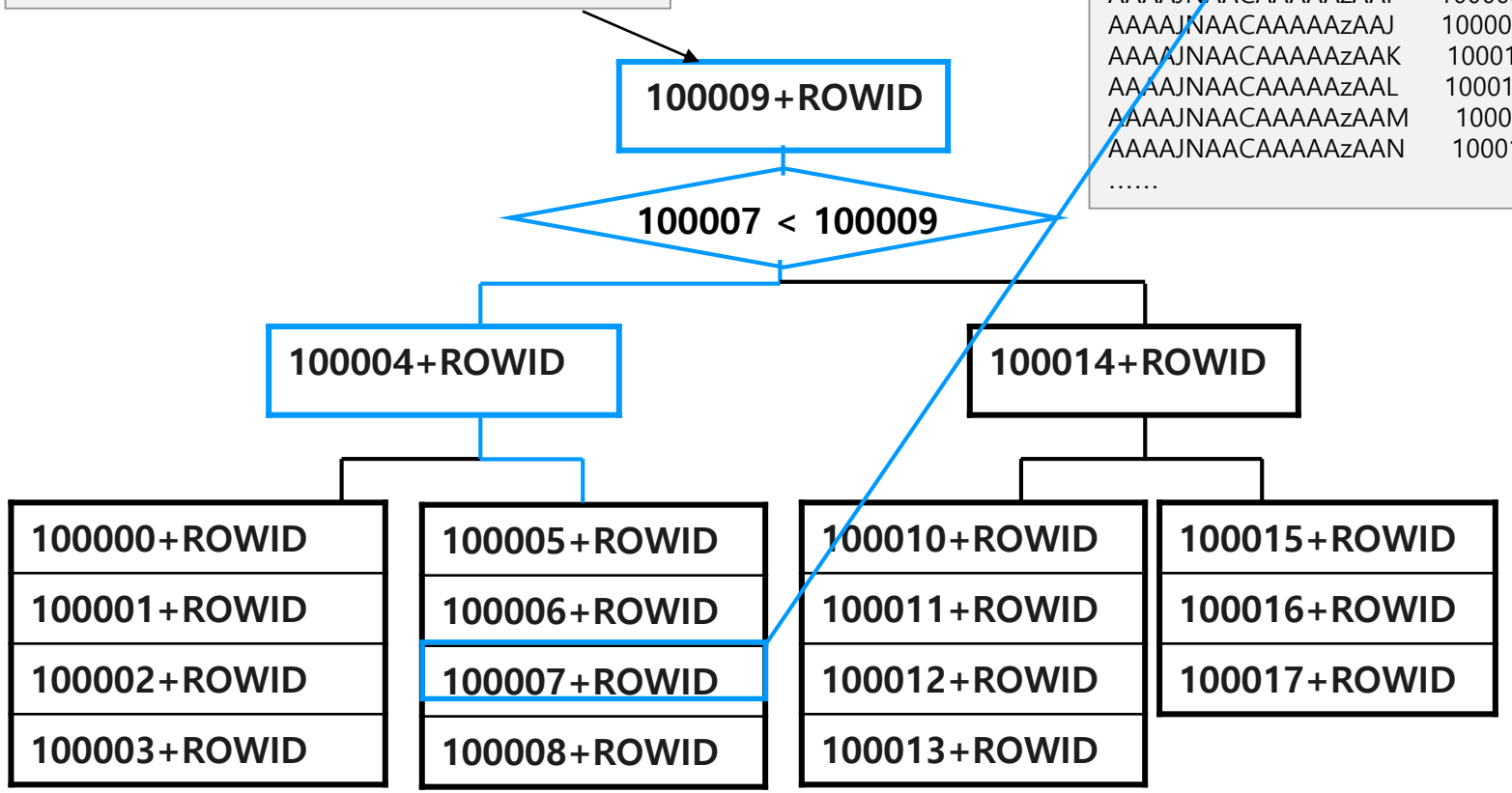
- **Guidelines for Indexes**
  - 테이블에 관련된 구조.
  - 인덱스의 존재여부는 SQL 명령문의 문법에 변화를 주지 않음.
  - 인덱스는 관련된 테이블의 데이터에 논리적, 물리적으로 독립적.
  - 인덱스를 삭제하여도 응용프로그램은 계속 수행.
  - Create Indexes After Inserting Table Data
    - tbLoader, tbImport 로 데이터를 삽입한 후 인덱스를 생성.
    - 데이터를 가진 테이블에 인덱스를 생성하려면 sort 공간을 사용.
  - 큰 인덱스를 관리하는 방법
    - 새로운 temporary segment 테이블스페이스를 생성
    - 생성자의 temporary segment 테이블스페이스를 변경
    - 인덱스 생성
    - temporary segment 테이블스페이스 삭제 및 변경 (원 위치)
  - 인덱스가 많으면 테이블 수정이 오버헤드 발생

인덱스

- 인덱스 기본 구조(Balance-Tree)

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

ROWID	PROD_ID	PROD_NAME
AAAAJNAACAAAAAzAAA	100000	Tibero1.0
AAAAJNAACAAAAAzAAB	100001	Tibero2.0
AAAAJNAACAAAAAzAAC	100002	Tibrto3.0
AAAAJNAACAAAAAzAAD	100003	AnyLink
AAAAJNAACAAAAAzAAE	100004	AnyLink2.0
AAAAJNAACAAAAAzAAF	100005	AnyLink3.0
AAAAJNAACAAAAAzAAG	100006	ProFrame
AAAAJNAACAAAAAzAAH	100007	ProFrame3.0
AAAAJNAACAAAAAzAAI	100008	ProFrame4.0
AAAAJNAACAAAAAzAAJ	100009	JEUS
AAAAJNAACAAAAAzAAK	100010	JEUS4.0
AAAAJNAACAAAAAzAAL	100011	JEUS5.0
AAAAJNAACAAAAAzAAM	100012	JEUS6.0
AAAAJNAACAAAAAzAAN	100013	WebtoB
……		



TmaxTibero

인덱스

인덱스의 생성

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

- 비유일 인덱스 생성 구문

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

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

기존 인덱스의 재생성

- Block 내부의 Fragmentation을 줄이는 것이 가능

```
SQL>ALTER INDEX prod_id_pk REBUILD;
```

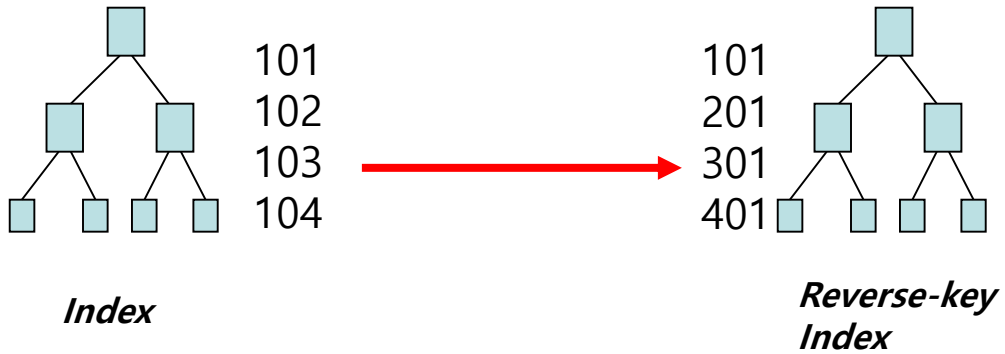
인덱스

Reverse-Key Indexes

- Reverse-key indexes
  - Reverse Key는 인덱스 할 각 컬럼의 값을 역순으로 삽입.
  - Bitmapped index or Index-organized table은 REVERSE 불가능.
  - Range Scan이 불가능 .
  - Parallel Server 환경이거나 유저가 Ascending Values를 Insert할 때 사용.
  - Delete가 적은 테이블에 적용.
  - Regular Index에 Alter를 이용하여 REVERSE 변환 가능.
  - Syntax

    CREATE INDEX *index_name* ON *table_name(column_name)* **REVERSE**;

    ALTER INDEX *index_name* REBUILD **NOREVERSE / REVERSE**;



인덱스

- **Function-based indexes**
  - Column과 Column의 연산에 대한 Index생성

```
CREATE INDEX sales_margin_idx ON sales(revenue -
```

```
SELECT ordid FROM sales WHERE (revenue - cost ) > 1000;
```

- **Descending indexes**
  - 내림차순으로 인덱스가 생성
  - 인덱스 된 각 컬럼들의 다른 Ordering을 지정함으로 인하여 Sorting요구 감소.

```
CREATE INDEX sales_city_margin_idx
    ON emp(city_name ASC, (revenue - cost)  DESC));
```

```
SELECT city_name, ordid, (revenue - cost) AS MARGIN
  FROM sales
 ORDER BY city_name ASC, margin DESC;
```

인덱스

- **Rebuilding Indexes Online**
  - Three Phase로 실행
    - Prepare
      . Minimal Table Lock를 요구
    - Bulid
      . 기존 Index를 재구축 하고, Rebuild동안 발생한 Data는 Journal Table에 임시 보관
    - Merge
      . Rebuild Index와 Journal Table의 Data를 Merge
  - CREATE INDEX ord_idx ON orders(ord_id) ONLINE;
  - ALTER INDEX ord_idx COALESCE;

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

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

인덱스

▪ 인덱스 적용 기준 및 컬럼 선정

➢ 인덱스 적용 기준

- 테이블에서 사용하는 블록수가 적은 경우(DB_FILE_MULTIBLOCK_READ_COUNT 값 이하)
- 컬럼의 분포도가 10 ~ 15 % 이내인 경우 적용
- 분포도가 범위 이내더라도 절대량이 많은 경우에는 다른 방법 적용
- 분포도가 범위 이상이더라도 부분범위처리를 목적으로 하는 경우에는 적용
- 인덱스만을 사용하여 요구를 해결하는 경우는 분포도가 나쁘더라도 적용할 수 있음(손익분기점)

➢ 인덱스 컬럼 선정

- 분포도가 좋은 컬럼은 단독적으로 생성하여 활용도 향상
- 자주 조합되어 사용되는 경우는 결합인덱스 생성
- 액세스 경우의 수를 만족할 수 있도록 각 인덱스간의 역할 분담 (Application 별이 아닌 사용형태별)
- 가능한 수정이 빈번하지 않는 컬럼
- 가능한 한 컬럼이 여러 인덱스에 포함되지 않게 할 것
- 결합 인덱스의 컬럼순서 선정에 주의 (사용빈도, 유일성, SORT유형, 부분범위처리 고려)
- 반복 수행되는 조건은 가장 빠른 수행속도를 낼 수 있도록 하는데 초점을 맞출 것
- 실제 조사된 액세스 종류를 토대로 선정 및 검증

인덱스

▪ 인덱스 고려사항

- 새로 추가된 인덱스는 기존 Access 경로에 영향을 미칠 수가 있음
- 지나치게 많은 인덱스는 많은 오버헤드를 발생.
- 넓은 범위를 인덱스로 처리 시 많은 오버헤드 발생.
- 옵티마이저를 위한 통계데이터를 주기적으로 갱신.
- 인덱스를 위한 추가적인 저장공간이 필요해 짐
- 인덱스의 개수는 테이블의 사용형태에 따라 다름

  (검색 위주형, 동시 다량 처리형, 단일 처리형, 배치 처리형)
- 분포도가 양호한 컬럼도 처리 조건(범위)에 따라 분포도가 나빠 질 수 있음
- NULL 을 가지는 컬럼의 인덱스 ENTRY는 생성되지 않음
- 인덱스 사용원칙을 준수해야 인덱스가 사용되어짐

인덱스

▪ 인덱스 정보 조회

뷰	설명
DBA_INDEXES	• 티베로내의 모든 인덱스에 대한 정보
USER_INDEXES	• 현재 유저에 속한 인덱스에 대한 정보
ALL_INDEXES	• 유저가 접근 가능한 인덱스에 대한 정보
DBA_IDX_COLUMNS	• 티베로내의 모든 인덱스에 적용된 컬럼에 대한 정보
USER_IDX_COLUMNS	• 현재 유저에 속한 인덱스에 적용된 컬럼에 대한 정보
ALL_IDX_COLUMNS	• 유저가 접근 가능한 인덱스에 적용된 컬럼에 대한 정보

4. 파티션(PARTITION) 테이블

파티션(Partition) 테이블

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

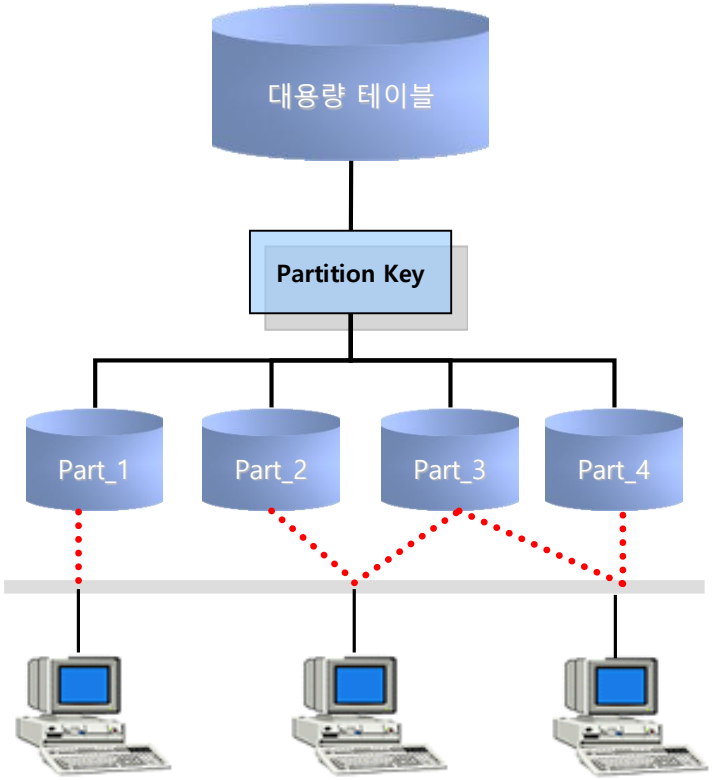
- 파티션 테이블의 장점
  - 개선된 가용성
    - 파티션은 독립적으로 관리가능.
    - Backup and Restore를 파티션 별로 작업 가능.
    - 같은 테이블에서 Unavailable한 파티션은 다른 파티션에 영향을 주지 않음.

  - 관리의 편리성
    - 사용자가 지정한 값으로 파티션이 가능.
    - 테이블스페이스 간에 파티션 이동이 가능.
    - 파티션 레벨에서 Select, Delete, Update가 가능.

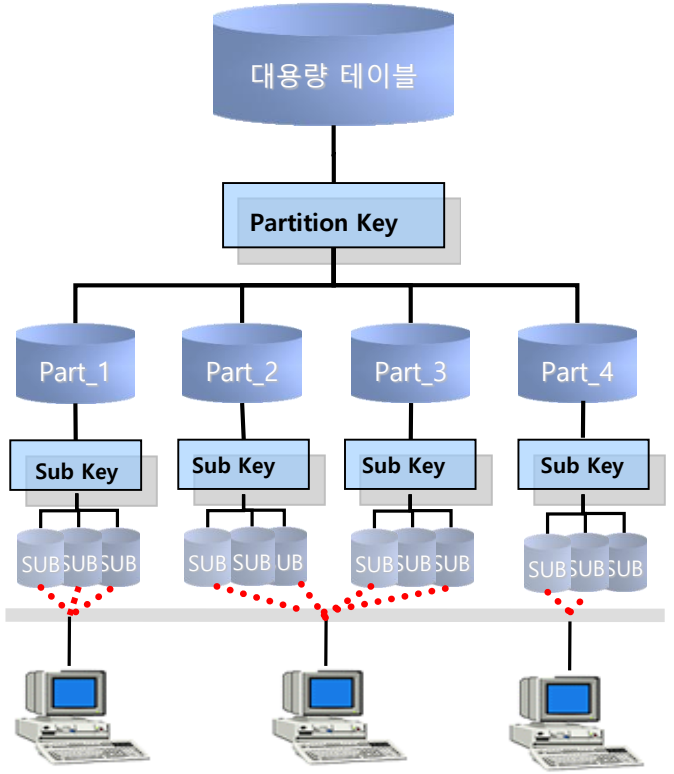
  - 개선된 성능
    - 데이터를 액세스할 때 액세스하는 범위를 줄여 퍼포먼스 향상.

파티션(Partition) 테이블

- **Single 파티션과 Composite 파티션 비교**



➢ **Partition의 특징**
- 대용량 테이블에 적합 (파티션 기준 10G 이하)
- 종류는 Range, Hash, List 파티션 존재
- 월별 관리는 대부분 Range 파티션 사용
- 보관주기 후 데이터는 Truncate 삭제 가능



➢ **Composite Partition의 특징**
- 초대용량 테이블에 적합 (파티션 기준 10G이상.)
- Sub Partition 구성으로 성능의 극대화
- 종류는 Range + (Hash, List 구성)
- Range의 편의성과 Hash의 성능의 장점을 취합

TmaxTibero

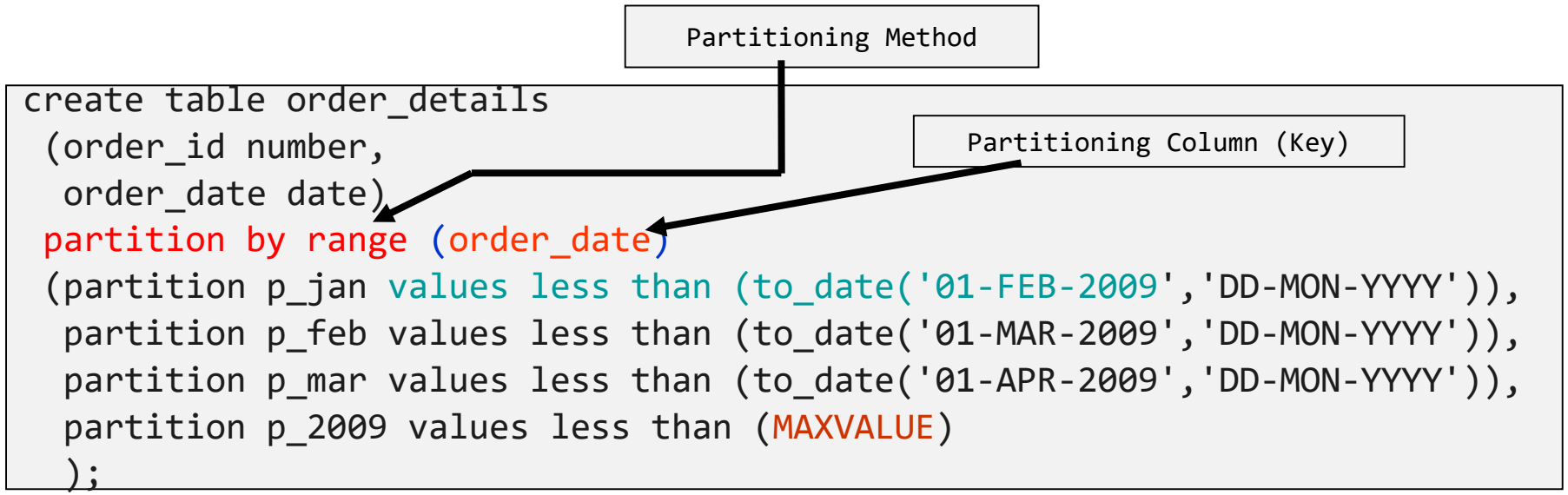
파티션(Partition) 테이블

- **Single 파티션과 Composite 파티션 비교 (계속)**

구 분	Partition 테이블 특성 및 기준	Composite Partition 테이블 특성 및 기준
특성	➢ **Partition의 특징** ▪ 대용량 테이블에 적합 (파티션 기준 10G 이하) ▪ 종류는 Range, Hash, List 파티션 존재 ▪ 월별 관리는 대부분 Range 파티션 사용 ▪ 보관주기 후 데이터는 Truncate 삭제 가능 ▪ Composite Partition은 Sub-Partition의 관리 부분이 Partition 보다 존재하므로 적절한 기준 이하는 Partition 으로 관리	➢ **Composite Partition의 특징** ▪ 초대용량 테이블에 적합 (파티션 기준 10G 이상) ▪ Partition 특성과 대부분 동일 ▪ 종류는 Range + (Hash, List 구성) ▪ Range의 관리와 Hash의 성능의 장점을 취합 (Range Partition의 관리의 용이성 및 Hash의 특징인 배치 및 병렬화를 제공) ▪ 서브 Partition은 객체의 관리만 논리적으로 할 뿐 하나의 테이블과 동일하게 물리적으로 취급

파티션(Partition) 테이블

- **Range 파티션**
  - Column Value의 범위를 기준으로 하여 행을 분할하는 형태
  - Range Partition에서 Table은 단지 논리적인 구조이며 실제 데이터가 물리적으로 저장되는 곳은 Partition으로 나누어진 Tablespace 에 저장
  - PARTITION BY RANGE (column_list) : 기본 Table에서 어느 Column을 기준으로 분할할지를 정함
  - VALUES LESS THAN (value_list) : 각 Partition이 어떤 값의 범위를 포함 할지 Upper Bound를 정함



파티션(Partition) 테이블

- HASH 파티션
  - Partitioning column의 Partitioning Key 값에 Hash 함수를 적용하여 Data를 분할하는 방식
  - 데이터 이력관리의 목적 보다 성능 향상의 목적으로 나온 개념
  - Hash Partition은 Range Partition에서 범위를 기반으로 나누었을 경우 특정 범위에 분포도가 몰려서 각기 Size가 다르게 되는 것을 보완하여, 일정한 분포를 가진 파티션으로 나누고, 균등한 데이터 분포도를 이용한 병렬처리로 퍼포먼스를 보다 향상 가능.
  - Hash Partition에서 Table은 단지 논리적인 구조이며 실제 데이터가 물리적으로 저장되는 곳은 Partition으로 나누어진 Tablespace에 저장

```
CREATE TABLE employees (
   empno NUMBER(4),
   ename VARCHAR(30),
   sal   NUMBER
)
 PARTITION BY HASH (empno) (
  PARTITION h1 TABLESPACE t1,
  PARTITION h2 TABLESPACE t2,
  PARTITION h3 TABLESPACE t3,
  PARTITION h4 TABLESPACE t4
);
```

파티션(Partition) 테이블

- LIST 파티션
  - Partitioning column의 특정 값으로 분할하는 방식
  - 데이터 분포도가 낮지 않고, 균등하게 분포되어 있을 때 유용
  - Composite Partition에서 'Range-List' 일 경우 그 효율이 더욱 높아짐
  - 다른 파티션 방식처럼 다중 컬럼을 지원하지 않고 단일 컬럼만 가능

```
CREATE TABLE sales_list
(  salesman_id  NUMBER(5),
   salesman_name VARCHAR(30),
   sales_state   VARCHAR(20),
   sales_amount  NUMBER(10),
   sales_date    DATE)
PARTITION BY LIST(sales_state)
(
   PARTITION sales_west VALUES ('California', 'Hawaii'),
   PARTITION sales_east VALUES ('New York', 'Virginia', 'Florida'),
   PARTITION sales_central VALUES ('Texas', 'Illinois'),
   PARTITION sales_other VALUES (DEFAULT)
);
```

파티션(Partition) 테이블

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

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

- **파티션 추가 삭제**

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

SQL> ALTER TABLE ORDERED
        DROP PARTITION PART1;
```

파티션(Partition) 테이블

- **파티션 키 선정 시 고려사항**

  - 성능향상을 위한 부분을 고려

    . 액세스 유형에 따라 Partioning이 이루어질 수 있도록 파티션 Key를 선정

  - 관리의 용이성을 대한 고려

    . 데이터의 생성 주기와 소멸주기가 파티션과 일치 필요

  - 각 파티션의 I/O 분산과 각 파티션의 크기를 의 설정

파티션(Partition) 테이블

- **파티션 테이블 정보 조회**

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

5. 파티션(Partition) 인덱스

파티션(Partition) 인덱스

- Local & Global 파티션(Partition) 인덱스

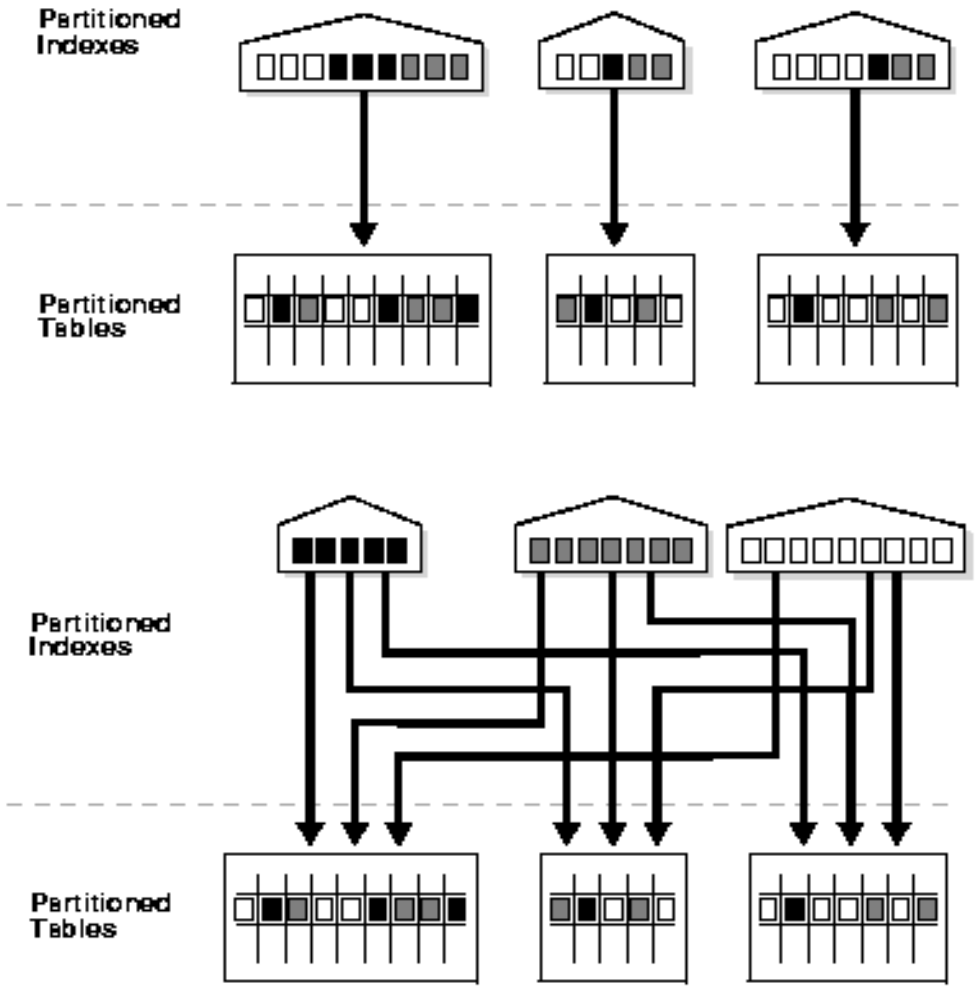
LOCAL INDEX

인덱스는 테이블 파티션과 동일하게 분할됨.

한개의 인덱스 파티션은 대응하는 테이블 파티션의 row만 포함함.

GLOBAL INDEX

한개의 파티션 파티션은 여러 테이블 파티션의 rows를 포함함.



TmaxTibero

파티션(Partition) 인덱스

- **Local 파티션(Partition) 인덱스**
  - 로컬 파티션 인덱스는 테이블의 한 파티션과 1:1로 대응
  - 대응되는 Index Partition과 Table Partition은 각각 같은 범위
  - OLTP 보다는 DSS 환경에 알맞음.

```
CREATE TABLE sales (
   acct_no          NUMBER(5)  NOT NULL,
   person           VARCHAR(30),
   sales_amount     NUMBER(8)  NOT NULL,
   week_no          NUMBER(2)  NOT NULL)
PARTITION BY RANGE (week_no)
             (PARTITION p1 VALUES LESS THAN   (4) TABLESPACE data1,
              PARTITION p2 VALUES LESS THAN   (8) TABLESPACE data2,
              PARTITION p3 VALUES LESS THAN   (MAXVALUE) TABLESPACE
   data3
);
```

```
CREATE INDEX sales_idx ON sales(week_no, acct_no )
         LOCAL
             (PARTITION I_p1 TABLESPACE IDX1,
              PARTITION I_p2 TABLESPACE IDX2,
              PARTITION I_p3 TABLESPACE IDX3
);
```

파티션(Partition) 인덱스

- **Local 파티션(Partition) 인덱스**

  . Non-Prefix Index : index에서 맨 앞에 위치한 컬럼을 기준으로 파티션

```
CREATE TABLE sales (
   acct_no          NUMBER(5)  NOT NULL,
   person           VARCHAR(30),
   sales_amount     NUMBER(8)  NOT NULL,
   week_no          NUMBER(2)  NOT NULL)
PARTITION BY RANGE (week_no)
             (PARTITION p1 VALUES LESS THAN   (4) TABLESPACE data1,
              PARTITION p2 VALUES LESS THAN   (8) TABLESPACE data2,
              PARTITION p3 VALUES LESS THAN   (MAXVALUE) TABLESPACE
   data3
);
```

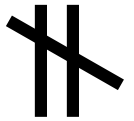
```
CREATE INDEX sales_nonp_idx ON sales(acct_no, week_no )
         LOCAL
             (PARTITION I_p1 TABLESPACE IDX1,
              PARTITION I_p2 TABLESPACE IDX2,
              PARTITION I_p3 TABLESPACE IDX3
);
```

파티션(Partition) 인덱스

- **Global 파티션(Partition) 인덱스**
  - 글로벌 파티션 인덱스는 테이블의 파티션과 무관. 어느 파티션에 있는 행도 가르킬 수 있음.
  - 테이블 파티션과는 독립적으로 구성
  - OLTP 환경에 알맞음.
  - prefixed global index만이 존재하며, non-prefixed global index는 생성이 불가능

```
CREATE TABLE sales (
   acct_no              NUMBER(5)  NOT NULL,
   person                VARCHAR(30),
   sales_amount     NUMBER(8)  NOT NULL,
   week_no            NUMBER(2)  NOT NULL)
PARTITION BY RANGE (week_no)
(PARTITION p1 VALUES LESS THAN (4) TABLESPACE data0,
 PARTITION p2 VALUES LESS THAN (8) TABLESPACE data1,
 PARTITION p3 VALUES LESS THAN (MAXVALUE) TABLESPACE
   data2
);
```

Table Partition 1~3



```
CREATE INDEX sales_idx (
   ON sales(week_no, acct_no )
GLOBAL
(PARTITION I_p1 VALUES LESS THAN (4)  TABLESPACE
   data0_Idx,
 PARTITION I_p2 VALUES LESS THAN (MAXVALUE)  TABLESPACE
   data2_Idx
);
```

Index Partition 1~2

파티션(Partition) 인덱스

- **파티션 인덱스 정보 조회**

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

6. 뷰(VIEW)

뷰(View)

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

- 뷰(View) 생성

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

- 뷰(View) 삭제

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

- 뷰(View) 정의 변경
  - 뷰의 정의 변경은 저장된 SQL구문을 새로 생성하는 것이므로 삭제 후 새로 생성

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

뷰(View)

▪ 뷰(View) 정보 조회

뷰	설명
DBA_VIEWS	• 티베로내의 모든 뷰에 대한 정보
USER_VIEWS	• 현재 유저에 속한 뷰에 대한 정보
ALL_VIEWS	• 유저가 접근 가능한 뷰에 대한 정보
DBA_UPDATABLE_COLUMNS	• 티베로내의 모든 뷰에 속한 컬럼에 대한 갱신 가능성 정보
USER_UPDATABLE_COLUMNS	• 현재 유저에 속한 뷰에 속한 컬럼에 대한 갱신 가능성 정보
ALL_UPDATABLE_COLUMNS	• 유저가 접근 가능한 뷰에 속한 컬럼에 대한 갱신 가능성정보

7. 동의어(SYNONYM)

시노님(Synonym)

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

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

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

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

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

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

시노님(Synonym)

시노님(Synonym) 정보 조회

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

8. 시퀀스(SEQUENCE)

시퀀스(Sequence)

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

- 자동으로 유일한 번호를 생성한다.
- 공유하는 객체이다.
- Primary Key 값을 생성하는 용도로 사용되기도 한다.
- Application 의 코드를 대체할 수 있다.
- 메모리에 캐시된 값을 이용하여 시퀀스 성능을 향상 시킬 수 있다.

Sequence 장점

** 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(Sequence)

▪시퀀스 생성 구문

- 연속된 번호를 생성하기 위해 다음과 같은 문법을 사용함

```
CREATE SEQUENCE sequence
                [INCREMENT BY n]
                [START WITH n]
                [{MAXVALUE n | NOMAXVALUE}]
                [{MINVALUE n | NOMINVALUE}]
                [{CYCLE n | NOCYCLE}]
                [{CACHE n | NOCACHE}];
```

구성요소	설명
INCREMET BY	시퀀스의 간격을 지정한다. 이 값을 양수로 입력하면 시퀀스의 값이 증가하게 되고, 음수로 입력하면 시퀀스의 값이 감소하게 된다. 지정하지 않으면 디폴트 1로 간격이 지정된다. 이 값은 MAXVALUE에서 MINVALUE를 뺀 값보다 클 수 없다.
START WITH	시퀀스의 시작 값을 지정한다. 시작 값은 MINVALUE와 MAXVALUE 사이의 값을 지정할 수 있다. 시작 값을 지정하지 않으면 INCREMENT BY값이 양수인 경우 MINVALUE가, INCREMENT BY값이 음수인 경우 MAXVALUE가 시작 값이 된다.

**다음 페이지에 계속

시퀀스(Sequence)

구성요소	설명
NOMAXVALUE	MAXVALUE를 지정하지 않는 것과 같다. MAXVALUE는 시퀀스의 증가 값이 양수인 경우 시퀀스가 가질 수 있는 가장 큰 값인 INT64_MAX가 되고 증가 값이 음수인 경우 -1이 된다.
MAXVALUE	MAXVALUE의 값을 지정한다. MAXVALUE는 MINVALUE보다 작아서는 안 된다. 지정하지 않으면 NOMAXVALUE와 같은 값으로 동작한다.
NOMINVALUE	MINVALUE를 지정하지 않는 것과 같다. MINVALUE는 시퀀스의 증가 값이 양수인 경우 1의 값을 가지고, 증가 값이 음수인 경우 시퀀스가 가질 수 있는 가장 작은 값인 INT64_MIN이 된다.
MINVALUE	MINVALUE의 값을 지정한다. MINVALUE는 MAXVALUE보다 클 수 없다. 지정하지 않으면 NOMINVALUE와 같은 값으로 동작한다.
CYCLE	CYCLE이 지정되면 MAXVALUE나 MINVALUE에 도달해도 계속 시퀀스 값을 생성한다. 증가 값이 양수일 때 시퀀스의 값이 MAXVALUE를 넘게 되면 MINVALUE 값을 주게 되고, 증가 값이 음수일 때 시퀀스 값이 MINVALUE보다 작아지면 MAXVALUE를 주게 된다.

시퀀스(Sequence)

구성요소	설명
NOCYCLE	CYCLE과 반대로 MAXVALUE 또는 MINVALUE에 도달하게 되면 더 이상 값을 만들지 않는다. CYCLE이나 NOCYCLE 둘 다 지정하지 않으면 NOCYCLE이 기본값이 된다.
CACHE	시퀀스의 속도를 높이기 위해 시퀀스 캐시에 지정된 개수만큼의 시퀀스 값을 저장해 두고 시퀀스 값을 요청하면 시퀀스 캐시에서 받아오게 된다. (기본값: 20) 저장해 둔 시퀀스 값을 다 사용하게 되면 새로 그만큼 더 받아오게 되며, 이때만 데이터 사전의 값을 수정하게 된다. 즉, CACHE로 임의의 수 N을 지정하였다면, NOCACHE로 사용할 때보다 1/N의 데이터 사전 수정이 이루어지게 되므로 속도를 높일 수 있다. 단 비정상 종료가 발생한 경우 캐시에 저장된 값은 없어진다.
NOCACHE	시퀀스 캐시를 사용하지 않고 매번 직접 값을 받아온다. 따라서 매번 데이터 사전을 수정하게 된다.
ORDER	클러스터 환경에서 노드 간에 발급되는 시퀀스 값의 순서를 유지한다. 노드에 관계 없이 요청된 순서대로 시퀀스 값이 발급된다. 클러스터 환경으로 설정한 경우에만 지정할 수 있다.
NOORDER	ORDER와 반대로 클러스터 환경에서 노드 간에 발급되는 시퀀스 값의 순서를 유지하지 않는다. 노드 별로 각각의 시퀀스 캐시를 유지하므로 한 노드 안에서는 요청된 순서대로 시퀀스 값이 발급되지만, 전체 노드에서는 요청된 순서와 발급되는 시퀀스 값의 순서가 다를 수 있다. (ORDER나 NOORDER 둘 다 지정하지 않으면 NOORDER가 기본값이 된다.)

시퀀스(Sequence)

▪(예제) TEST_SEQ 시퀀스 생성, 사용 예제

```
SQL> CREATE SEQUENCE test_seq
       START WITH 10
       MINVALUE 5
       MAXVALUE 45
       INCREMENT BY 2
       CYCLE;

SQL> SELECT test_seq.nextval FROM dual;

TEST_SEQ.NEXTVAL
----------------
              10

SQL> SELECT test_seq.nextval FROM dual;

TEST_SEQ.NEXTVAL
----------------
              12

SQL> SELECT test_seq.currval FROM dual;

TEST_SEQ.NEXTVAL
----------------
              12
```

구분	NEXTVAL, CURRVAL 사용 가능 여부
사용 가능	• SELECT문 • INSERT문의 VALUES 절, SELECT 절 • UPDATE문의 SET 절
사용 불가능	• VIEW의 SELECT 절 • SELECT문의 DISTINCT 키워드 • SELECT문의 GROUP BY, HAVING, ORDER BY 절 • SELECT, DELETE, UPDATE문의 서브 쿼리 • CREATE TABLE, ALTER TABLE의 DEFAULT

시퀀스(Sequence)

사용 초기화

- 시퀀스 현재 값 조회

```
SQL> SELECT TEST_SEQ02.CURRVAL FROM DUAL;
   CURRVAL
----------
        44
```

- 원래 값으로 설정하기 위해 음수(-) 값으로 INCREMENT 설정

```
SQL> ALTER SEQUENCE TEST_SEQ02 INCREMENT BY -34;
SQL> ALTER SEQUENCE TEST_SEQ02 INCREMENT BY 2;
```

- NEXTVAL, CURRVAL 조회

```
SQL> SELECT TEST_SEQ02.NEXTVAL FROM DUAL;
   NEXTVAL
----------
        10
SQL> SELECT TEST_SEQ02.CURRVAL FROM DUAL;
   CURRVAL
----------
        10
```

시퀀스(Sequence)

- **시퀀스(Sequence) 정보 조회**

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

9. 잡(JOB)

JOB(DBMS_JOB)

▪Tibero에서는 주기적으로 데이터베이스에 추가된 JOB을 검사하여, 사용자가 설정한 시각이 되면 해당하는 JOB을 실행한다.

- DBMS_JOB은 PSM에서 사용 가능한 문장을 JOB으로 등록하고, 이 JOB을 실행할 수 있는 연산을 제공하는 패키지이다.

- DBMS_JOB 패키지 내의 프러시저를 이용하여, JOB을 데이터베이스에 추가하고 바로 실행하거나 정해진 시각에 실행되도록 설정할 수 있다.

JOB 특징

** JOB을 추가 또는 변경하는 경우 커밋을 실행하지 않아도 자동으로 커밋되며, JOB 내에서 실행한 작업도 자동으로 커밋된다.

** JOB 실행이 실패한 경우에는 재실행되며, 16번 실패하게 되면 해당 JOB은 broken 상태가 된다.

JOB(DBMS_JOB)

▪데이터베이스에 새로운 JOB을 추가하기

- SUBMIT 프로시저를 사용하며, 문법은 다음과 같다.

```
DBMS_JOB.SUBMIT
(
    job           OUT  BINARY_INTEGER,
    what          IN   VARCHAR2,
    next_date     IN   DATE DEFAULT sysdate,
    interval      IN   VARCHAR2 DEFAULT 'null',
    no_parse      IN   BOOLEAN DEFAULT FALSE,
    instance      IN   BINARY_INTEGER DEFAULT 0,
    force         IN   BOOLEAN DEFAULT FALSE
);
```

JOB(DBMS_JOB)

파라미터	설명
job	실행할 JOB의 번호이다.
what	실행할 PL/SQL 프러시저 또는 PSM 문장의 시퀀스이다.
next_date	job을 언제 처음 시작할 것인지 지정한다. ** 1분후 : SYSDATE + 1 / 24 / 60
interval	다음 JOB이 실행될 시각을 계산하기 위한 연산식이다. ** 하루 1회 : 'sysdate + 1' 매일 03시 : 'TRUNC(SYSDATE+1)+3/24' 매주월요일 : 'next_day(sysdate,''MONDAY'')' 매달 1일 06시 : 'TRUNC(ADD_MONTHS(SYSDATE, 1))+6/24' 한번 실행 : 'null'
no_parse	TRUE: submit을 할 때 JOB을 파싱하지 않고, JOB이 실행될 때 파싱을 하게 된다. 따라서 파싱의 실패 여부가 최초 실행 시점에 보고된다. FALSE: JOB에 관련된 프러시저를 미리 파싱한다.
instance	JOB을 수행할 INSTANCE이다. (기본값: 0, ANY_INSTANCE의 의미)
force	지원되지 않는 기능이므로 값을 무시한다.

JOB(DBMS_JOB)

▪(예제) JOB 생성, 사용 예제

1. job 생성, job동작에 의한 데이터 입력작업 확인

```
SQL> CREATE TABLE T1(C1 NUMBER);

SQL> declare
  job_no number;
begin
  dbms_job.submit(job_no,
 'begin insert into tibero.t1 values(1); end;'
 ,sysdate, 'sysdate + 1/24/60/60');
end;
/

SQL> select sum(c1) from t1;

   SUM(C1)
----------
        12

SQL> select sum(c1) from t1;

   SUM(C1)
----------
        18
```

2. job 조회, 수행 중지하기

```
SQL> SELECT JOB, BROKEN FROM USER_JOBS;

       JOB BROKEN
---------- ------
        12 N

SQL> EXEC DBMS_JOB.BROKEN(12,TRUE);

SQL> SELECT JOB, BROKEN FROM USER_JOBS;

       JOB BROKEN
---------- ------
        12 Y

SQL> SELECT SUM(C1) FROM T1;

   SUM(C1)
----------
      1293

SQL> SELECT SUM(C1) FROM T1;

   SUM(C1)
----------
      1293
```

JOB(DBMS_JOB)

그 밖의 프로시저

파라미터	설명
REMOVE	DB에 추가된 job을 삭제하는 프로시저 DBMS_JOB.REMOVE(job); * job : 삭제할 job number
CHANGE	DB에 저장되어 있는 job의 field들을 변경하는 프로시저 DBMS_JOB.CHANGE(job, what, next_date, interval); * job : 실행할 job number * what : 실행할 PL/SQL procedure 혹은 psm 문장의 sequence * next_date : job을 다음 수행할 시간 * interval : job을 수행 후 nex_date를 update하기 위한 expression. date type으로 evaluate되는 문자열
WHAT	job이 수행하는 작업을 변경하는 프로시저 DBMS_JOB.WHAT(job, WHAT); * job : 실행할 job number * what : 실행할 PL/SQL procedure 혹은 psm 문장의 sequence

JOB(DBMS_JOB)

그 밖의 프로시저

파라미터	설명
NEXT_DATE	job이 schedule되어 Tibero에 의해 자동으로 실행될 때를 변경하는 프로시저 DBMS_JOB.NEXT_DATE(job, next_date); * job : 실행할 job number * next_date : job이 schedule되어 실행될 시간
INTERVAL	job 실행주기 파라미터를 변경하는 프로시저 DBMS_JOB.INTERVAL(job, interval); * job : 실행할 job number * interval : job을 수행 후 next_date를 update하기 위한 expression. date type으로 evaluate되는 문자열
BROKEN	DB에 저장되어 있는 job의 상태를 정상 or Broken 상태로 설정하는 프로시저 DBMS_JOB.BROKEN(job, broken, next_date); * job : 실행할 job number * broken : job이 broken 된 경우 true, 정상 상태인 경우 false
RUN	job을 현재 session에서 즉시 수행시키는 프로시저. job이 broken되어 있어도 실행하고, 실행에 성공한 경우 job을 정상 상태로 변경한다. DBMS_JOB.RUN(job);

TmaxTibero

감사합니다