Real Alternative DBMS ALTIBASE, Since 1999

ALTIBASE 모니터링 쿼리 가이드

ALTIBASE 6.3.1

2015. 01.08

Document Control

Change Record

Date	Author	Change Reference
2010-03-09	durusari	Created
2010-03-29	khhan	Template Modified
2010-10-20	durusari	Added, Modified
2014-12-05	Likepd80	Added, Modified
2015-01-08	Likepd80	Added, Modified

Reviews

Date	Name (Position)
2010-11-09	lim272(SP), dplee(TS), son2865(SC), fhan(TS), khhan(SC)

Distribution

Name	Location

목차

개요

본 문서는 ALTIBASE 모니터링을 위해 기본적으로 습득해야 할 사항과 그에 따른 일반적인 모니터링 쿼리 예시를 제시하는 문서로 아래와 같이 크게 3 개의 섹션으로 구성되어 있습니다.

- ALTIBASE 모니터링 개요
- ALTIBASE 메타테이블(meta table) 및 성능뷰(performance view) 개요
- 모니터링 요소 와 그에 대응하는 모니터링 쿼리

모든 섹션은 ALTIBASE 5.3.3 을 기준으로 작성되었습니다. 다만, 모니터링 쿼리는 5.1.5 에서도 사용할 수 있도록 변경하여야 할 부분에 대하여 별도로 명시를 하였습니다.

본 문서는 예시로 제시되는 모니터링 쿼리와 관련된 메타테이블 및 성능뷰 일부에 대하여서만 간략히 설명하고 있으며 컬럼설명 또한 컬럼이름이 직관적이어서 별도의 설명이 필요 없다고 판단되는 경우, 이전에 이미 설명한 컬럼이 중복되어 나오는 경우는 생략하였습니다. 따라서, 보다 상세한 설명과 이해를 위해서 기본적으로 ATC (http://atc.altibase.com)에 게시된 운영자 매뉴얼을 함께 참조하시기 바랍니다.

모니터링 쿼리 예시만 참조하려면 "모니터링 요소" 섹션으로 바로 이동하시기 바랍니다.

본 문서에 포함된 ERD는 CA ERwin ® Data Modeler을 사용하여 IDEF1X 표기법으로 작성되었습니다.

본 문서와 관련된 오류사항은 ALTIBASE 기술본부 대표 메일인 support@altibase.com으로 보내주시기 바랍니다.

ALTIBASE 모니터링 개요

ALTIBASE 모니터링을 위한 기본적인 사항에 대해 설명한다.

모니터링 분류

ALTIBASE는 DBMS이다. DBMS 모니터링은 아래와 같이 크게 3 가지로 분류할 수 있다.

내부 모니터링

데이터 딕셔너리에 대한 쿼리를 통한 DBMS 내부에서의 모니터링을 의미한다.

외부 모니터링

OS 명령어를 통한 DBMS 외부에서의 모니터링을 의미한다.

trace로그 모니터링

DBMS에 의해 기록되는 각종 trace로그에 대한 모니터링을 의미한다.

데이터 딕셔너리

데이터베이스에 대한 모든 정보를 요약 및 저장하여 DBMS를 효율적으로 사용할 수 있도록 하는 것으로 ALTIBASE 데이터 딕셔너리는 두 가지로 구성되어 있다.

메타테이블 (meta table)

데이터베이스 객체를 관리하기 위해 데이터베이스 생성시점에 자동으로 생성되는 테이블이다. 해당 테이블은 사용자 "SYSTEM_"의 소유로 일반 사용자는 SELECT만 가능하다.

성능뷰 (performance view)

SELECT 시점에 ALTIBASE 내부의 상태정보를 뷰 형태로 제공되는 것으로 ALTIBASE 내부의 최신 정보를 얻을 수 있다. 물리적으로 저장되지 않으며 이 역시 SELECT만 가능하며 성능뷰의 접두어는 "v$"이다.

모니터링 방법

현업에서 일반적으로 사용하는 모니터링 방법은 아래와 같다.

- 쉘 스크립트 작성
- 응용프로그램 작성
- 유틸리티 활용

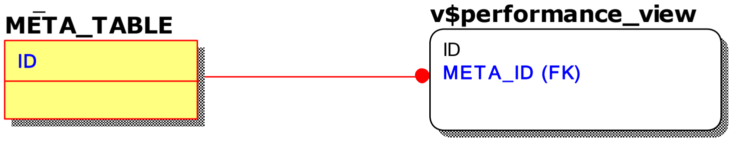
유틸리티의 경우, 비공식 유틸리티이나 ALTIBASE 기술본부에 의해 활용되는 ALTIMON과 ALTIBASE 5.1.5 부터 공식적으로 제공되는 AdminCenter2 for DBA와 3rd party 제품인 Orange for ALTIBASE DBA edition 등이 있다.

ALTIBASE 메타테이블 및 성능뷰 개요

ALTIBASE 내부 모니터링을 위해서는 ALTIBASE 데이터 딕셔너리인 메타테이블과 성능뷰에 대한 이해가 우선적으로 필요하다. 본 문서에서는 사용자의 이해와 쿼리 편의를 고려하여 메타테이블과 성능뷰의 관계를 ERD(Entity-Relationship Diagram)를 사용하여 표현하였다. 이에 앞서 숙지해야 할 사항에 대해 설명한다.

ERD 표기에 관하여

사용자의 이해와 쿼리 편의를 고려하여 기반 테이블이 존재하지 않는 성능뷰 특성을 무시, 아래와 같이 성능뷰를 테이블로 표현하여 조인 시 참고할 주요 키를 FK 형태로 표현하였다. 이러한 표기로 인한 혼동을 피하기 위해 메타테이블은 배경을 노란색으로 성능뷰는 흰색으로 표시하였다.



또한, 위 ERD에서 메타테이블의 ID 컬럼과 성능뷰의 META_ID 컬럼처럼 동일한 속성의 컬럼이 메타테이블 및 성능뷰에 따라 서로 다른 이름일 수 있는데, 이런 경우는 컬럼에 동일한 색상을 주어 같은 속성임을 나타냈다.

유의사항

ALTIBASE 메타테이블과 성능뷰는 아래와 같은 특성이 있음을 유의해야 한다.

1. 동일한 속성의 컬럼이 메타테이블 및 성능뷰에 따라 서로 다른 이름일 수 있다.
2. ALTIBASE 버전에 따라 메타테이블 및 성능뷰의 컬럼이름이 변경되거나 삭제될 수 있다.
3. ALTIBASE 버전에 따라 메타테이블 및 성능뷰가 추가되거나 삭제될 수 있다.

용어

설명에 앞서 혼동이 발생할 수 있는 일부 용어에 대한 정의이다.

세션 (session)

ALTIBASE에 접속한 사용자의 접속 단위를 의미한다. 하나의 사용자는 동시에 여러 개의 세션을 가질 수 있다.

구문 (statement)

트랜잭션에서 수행되는 SQL 하나하나를 의미하는 용어로 문맥에 따라 "SQL (구)문", "Query", "질의(문)", "쿼리(문)"는 모두 동일한 의미를 가지는 경우가 많다. 본 문서에서는 "쿼리"로 통일한다.

메모리DB GC (garbage collector) or GC

사용자가 commit을 수행하면 MVCC(Multi Versioning Concurrency Control / 다중레코드동시제어) 기법에 의해 유지되던 변경전의 레코드는 삭제가 되어야 한다.

ALTIBASE는 변경전의 레코드 삭제를 위해 별도 쓰레드를 운영하는데, 이를 각각 메모리DB GC, 디스크DB GC라 칭한다. 하지만, ALTIBASE 5.3.3 부터 디스크DB의 MVCC 방식이 변경되어 디스크DB GC가 없어지면서 메모리DB GC만 존재하기에 줄여서 GC 또는 Ager라 한다.

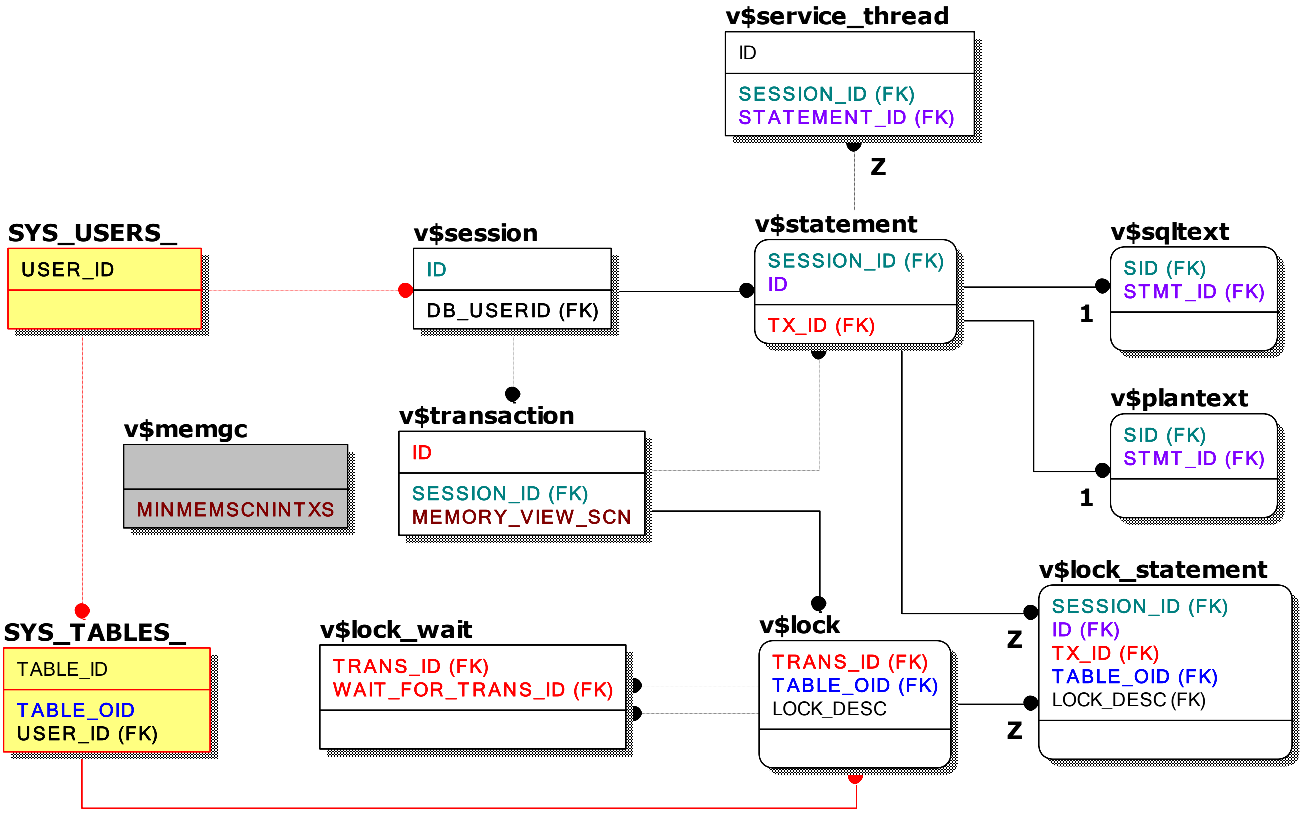
세션, 쿼리, 트랜잭션, lock, 서비스쓰레드, 메모리DB GC 관련 주요 메타테이블 및 성능뷰

메타테이블 및 성능뷰를 통하여 실시간으로 세션, 쿼리, 트랜잭션, lock, 서비스쓰레드, 메모리DB GC 상태를 확인 가능하다. 관련 조인쿼리 시 참고할 주요 키 컬럼만 표기한 메타테이블 및 성능뷰의 관계는 아래와 같다.

별개로 표시되어 있는 v$memgc는 메모리DB GC에 대한 성능뷰로 다른 메타테이블 및 성능뷰와 직접적인 관계는 없으나 v$transaction의 특정 컬럼과 함께 활용된다.

유의사항으로는 v$statement의 컬럼 중 쿼리수행과 관련된 통계정보는 TIMED_STATISTICS 프로퍼티가 활성화(1)되어야만 갱신되므로 해당 프로퍼티의 활성화 여부를 반드시 확인하여야 한다는 것이다. TIMED_STATISTICS 기본값은 비활성화(0)이다.

참고로 이중화를 수행으로 인한 트랜잭션도 v$transaction, v$lock, v$lock_wait를 활용하나 세션번호, 쿼리번호가 없는 이중화 트랜잭션 특성상 이중화 섹션에서 별도로 언급하는 것으로 한다.



주요 메타테이블

SYS_USERS_

모든 사용자의 정보가 저장된 메타테이블로 사용자이름(USER_NAME)과 같은 정보를 확인할 수 있어 가장 기본적으로 활용된다.

SYS_TABLES_

모든 테이블(큐테이블 포함)뿐만 아니라 시퀀스, 뷰의 정보도 함께 저장한 메타테이블로 해당 객체이름(TABLE_NAME)과 같은 정보를 확인할 수 있어 SYS_USERS_와 함께 기본적으로 활용된다.

주요 성능뷰

v$session

현재 접속되어 있는 사용자의 세션 정보를 나타내는 성능뷰이다.

v$statement, v$sqltext, v$plantext

v$statement는 세션과 관련된 쿼리의 정보 및 쿼리레벨의 통계정보를 나타내는 성능뷰로 쿼리의 수행시간 및 수행빈도 측정이 가능하다. 세션 별로 가장 마지막에 direct 수행(execution)한 하나의 쿼리와 prepare된 다수의 쿼리에 한하여서만 유지되는 뷰로 관련 세션을 종료되면 사라지게 된다. 이 뷰를 통하여 쿼리의 텍스트도 최대 16K까지 확인이 가능하다.

쿼리의 텍스트가 16K를 초과하는 경우는 쿼리 텍스트 전체를 나타내는 성능뷰인 v$sqltext를 활용하며 쿼리에 대응하는 실행계획을 확인하려 할 때는 v$plantext를 활용한다.

앞서 언급했듯 v$statement의 컬럼 중 쿼리수행과 관련된 통계정보는 TIMED_STATISTICS 프로퍼티가 활성화(1)되어야만 제공되므로 해당 프로퍼티의 활성화 여부를 반드시 확인하여야 한다. TIMED_STATISTICS 기본값은 비활성화(0)이다.

v$service_thread

서비스쓰레드의 상태를 나타내는 성능뷰로 v$session, v$statement와 조인하여 관련 세션과 쿼리를 확인할 수 있다. 뷰 자체만으로도 의미가 있어 단독으로 모니터링 하기도 한다.

v$transaction, v$memgc

v$transaction은 현재 수행되는 모든 트랜잭션의 정보를 나타내는 성능뷰로 lock 관련 모니터링 시 기본적으로 사용된다. 또한, 메모리DB GC의 정보를 나타내는 성능뷰인 v$memgc를 통하여 세션, 구문과 연계된 MVCC 상태를 확인할 수도 있다.

v$lock, v$lock_wait

v$lock은 트랜잭션 수행 중 발생한 lock에 대한 모든 정보를 나타내는 성능뷰로 lock의 선후관계를 나타내는 v$lock_wait과 함께 활용한다.

v$lock_statement

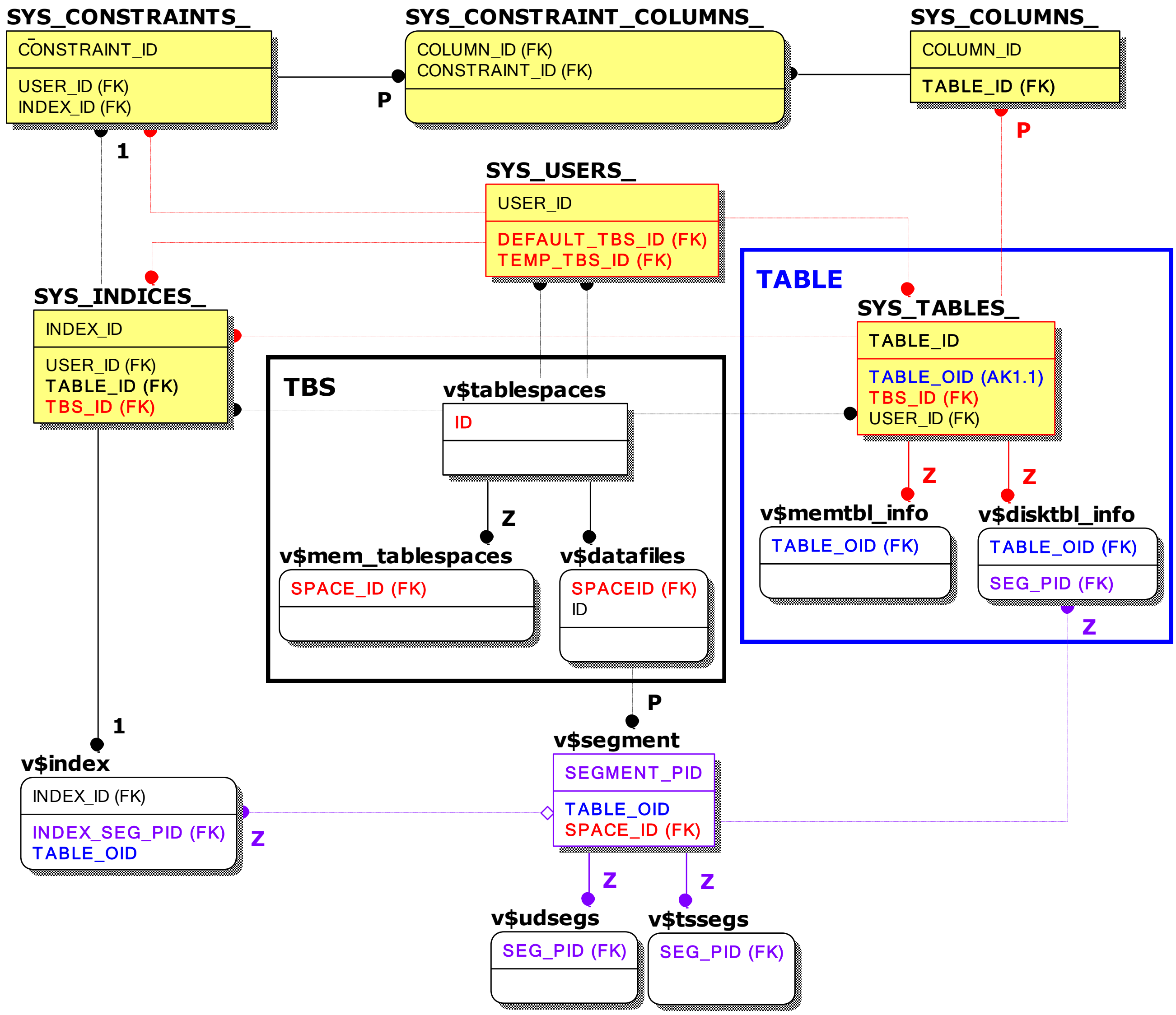
현재 lock을 획득한 트랜잭션에서 가장 마지막에 수행된 구문을 나타내는 성능뷰로 lock 발생시 관련 세션 및 구문에 대한 신속한 확인을 위해 주로 활용된다.

테이블스페이스, 테이블, 컬럼, 인덱스, 제약조건 관련 주요 메타테이블 및 성능뷰

ALTIBASE는 hybrid DBMS 특성상 테이블스페이스, 테이블에 대한 정보를 메모리DB, 디스크DB에 따라 별도(TBS, TABLE 박스 참조)로 확인이 가능하게끔 되어있으며 세그먼트의 경우는 v$segment와 관계(보라색)된 각각의 성능뷰를 통해 상세한 자원 사용현황을 확인할 수 있다.

하지만, 인덱스의 경우 현재로서 메모리 인덱스 크기 측정을 위한 메타테이블이나 성능뷰가 제공되지 않는 관계로 테이블 전체 레코드 수에 하나의 레코드에 대한 메모리 인덱스 크기인 16byte를 곱하는 것으로 대체하여야 한다. 디스크 인덱스는 v$index를 참조, 해당 세그먼트를 통하여 실시간 크기 측정이 가능하다.

테이블스페이스, 테이블, 컬럼, 인덱스, 제약조건 관련 조인쿼리 시 참고할 주요 키 컬럼만 표기한 메타테이블 및 성능뷰의 관계는 아래와 같다.



특이사항으로는 주요 메타테이블인 SYS_TABLE_과 다른 메타테이블간의 조인 시는 조인컬럼으로 TABLE_ID를 사용하고 성능뷰와의 조인 시는 대체 키인 TABLE_OID 컬럼(파란색)을 사용한다는 것이다.

참고로 표기는 되지 않았지만 SYS_USERS_, SYS_TABLES_을 제외한 모든 메타테이블은 쿼리 편의를 고려하여 USER_ID와 TABLE_ID 컬럼이 모두 존재한다. 성능뷰 v$segment와 v$index 역시 테이블 별 합산을 고려하여 TABLE_OID 컬럼이 존재한다.

주요 메타테이블

SYS_COLUMNS_

모든 테이블의 컬럼 정보가 저장된 메타테이블로 컬럼이름(COLUMN_NAME)은 물론 데이터타입(DATA_TYPE), 컬럼순서(COLUMN_ORDER)와 같은 상세한 정보를 확인할 수 있다.

SYS_CONSTRAINTS_

테이블의 제약조건 정보를 저장한 메타테이블로 제약조건유형(CONSTRAINT_TYPE)을 확인할 수 있으며 PK, FK, UK와 같은 인덱스 생성을 필요로 하는 제약조건의 경우 관련 인덱스번호(INDEX_ID)를 확인할 수 있다.
또한, FK의 경우 참조테이블(REFERENCED_TABLE_ID)도 확인이 가능하다.

SYS_CONSTRAINT_COLUMNS

제약조건의 대상이 되는 컬럼의 컬럼번호(COLUMN_ID)를 확인할 수 있는 메타테이블로 관련 제약조건번호(CONSTRAINT_ID)와 함께 대응되어 있다.

SYS_INDICES_

모든 인덱스 정보가 저장된 메타테이블로 인덱스이름(INDEX_NAME)은 물론 테이블스페이스번호(TBS_ID), 인덱스유형(INDEX_TYPE), 구성컬럼개수(COLUMN_CNT)와 같은 상세한 정보를 확인 가능하다.

주요 성능뷰

v$tablespaces

모든 테이블스페이스 정보를 나타내는 성능뷰로 테이블스페이스 관련 모니터링 시 기본적으로 활용된다.

v$mem_tablespaces

메모리 테이블스페이스에 대해서만 상세한 정보를 나타내는 성능뷰로 메모리DB 운영을 위해 실제로 사용하는 물리적 메모리 크기를 구할 수 있다.

v$datafiles

디스크 테이블스페이스를 구성하는 데이터 파일의 상세정보를 나타내는 성능뷰로 물리적인 데이터 파일의 경로, 상태 및 크기를 확인 수 있다.

v$segment

디스크 DB의 세그먼트 정보를 나타내는 성능뷰로 디스크 테이블, 디스크 인덱스, 언두 테이블스페이스, 임시 테이블스페이스와 같은 디스크DB 관련 객체의 정확한 크기를 구할 수 있다.

v$memtbl_info

메모리 테이블에 대해서만 상세한 정보를 나타내는 성능뷰로 메모리 테이블 별 순수 데이터의 크기 및 합계를 구할 수 있다.

v$disktbl_info

디스크 테이블에 대해서만 상세한 정보를 나타내는 성능뷰로 해당 세그먼트(SEG_PID)를 알 수 있어 디스크 테이블 별 순수 데이터의 크기 및 합계를 구할 수 있다.

v$index

모든 인덱스에 대한 간략한 정보를 나타내는 성능뷰로 디스크 인덱스의 경우 해당 세그먼트(INDEX_SEG_PID)를 알 수 있어 디스크 인덱스의 크기를 구할 수 있다.

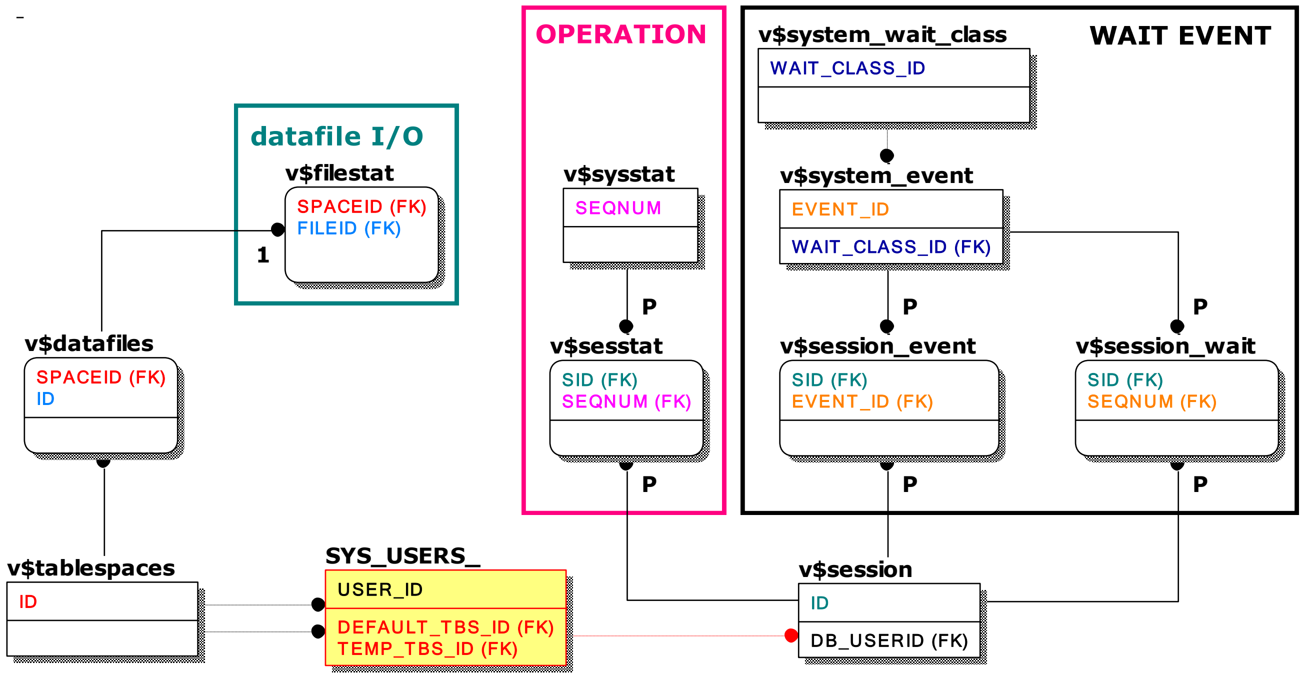
통계정보 관련 주요 성능뷰

성능뷰를 통하여 DBMS 레벨, 세션 레벨, 쿼리 레벨에서의 대기이벤트(wait event)와 연산(operation)에 대한 통계정보, DBMS 레벨의 디스크 테이블스페이스 데이터 파일 I/O, 메모리 사용량 버퍼풀 통계정보와 같은 다양한 통계정보를 확인할 수 있다.

레벨에 따라 통계정보를 생성하는 기준은 차이가 있다. DBMS 레벨의 통계정보는 ALTIBASE 구동시점부터 정보를 누적하며 ALTIBASE가 종료되면 초기화된다. 따라서, 특정기간 동안의 값을 알기 위해서는 (현재의 값 - 측정 시작 시점의 값)을 모든 칼럼 값에 대해 계산해야 한다. 세션레벨 통계정보는 관련 세션이 접속되어 있는 동안만 유지된다. 쿼리 레벨의 통계정보는 세션 별로 가장 마지막에 direct 수행(execution)한 하나의 쿼리와 prepare된 다수의 쿼리에 한하여서만 유지되며 관련 세션이 종료되면 이 역시 사라지게 된다. 쿼리 레벨의 통계정보는 쿼리섹션에서 이미 언급한 v$statement을 통하여 제공되는 것으로 본 섹션에서는 생략한다.

유의할 사항으로는 통계정보는 TIMED_STATISTICS 프로퍼티가 활성화(1)되어야만 가능하므로 해당 프로퍼티의 활성화 여부를 반드시 확인하여야 한다는 것이다. TIMED_STATISTICS 기본값은 비활성화(0)이다.

통계정보 관련 성능뷰는 대부분 뷰 자체만으로도 모니터링이 가능하므로 조인이 불필요하나 대기이벤트, 연산, 디스크 테이블스페이스의 데이터 파일 I/O에 대한 통계정보의 경우 특정 세션 또는 특정 데이터 파일에 연관을 지어 확인할 필요가 있다. 관련 정보를 얻기 위해 조인쿼리 시 참고할 주요 키 컬럼만 표기한 메타테이블 및 성능뷰의 관계는 아래와 같다.



대기이벤트 관련 성능뷰

대기이벤트란 “세션”또는 “ALTIBASE 쓰레드”의 일련의 대기 작업을 의미한다. 예를 들면, 서비스쓰레드가 disk buffer에 적재된 page를 접근하기 위해 page의 latch 획득을 대기하는 작업, 로그기록을 위해 로그버퍼의 latch 획득을 대기하는 작업등이 있다.

참고로 ALTIBASE는 대기이벤트를 그룹화하기 위해 상위 개념인 대기이벤트 클래스(wait event class)를 사용하여 8 가지로 분류하고 있다.

v$system_wait_class

"세션" 또는 "ALTIBASE 쓰레드"의 대기이벤트에 대한 통계정보를 "대기이벤트 클래스" 별로 나타낸다. ALTIBASE 구동 이후부터 누적되는 통계정보로 종료 시 초기화 된다.

v$system_event

"세션"또는 "ALTIBASE 쓰레드"의 대기이벤트에 대한 통계정보를 "대기이벤트" 별로 나타낸다. ALTIBASE 구동 이후부터 누적되는 통계정보로 종료 시 초기화 된다.

v$session_event

"세션"의 대기이벤트에 대한서만 통계정보를 나타낸다. 세션이 종료되면 관련 통계정보는 사라진다.

v$session_wait

"세션"중 조회시점에 "활성화된 세션"의 대기이벤트에 대한서만 통계정보를 나타낸다. 세션이 유휴상태(idle)가 되거나 종료되면 관련 통계정보는 사라진다.

연산 관련 성능뷰

연산이란 "세션"또는 "ALTIBASE 쓰레드"가 수행하는 각종 연산 작업을 의미한다. 예를 들면, 특정세션의 쿼리 수행, 서비스쓰레드의 리두로그 쓰기 등이 있다.

v$sysstat

"세션"또는 "ALTIBASE 쓰레드"의 연산에 대한 통계정보를 "연산" 별로 나타낸다. ALTIBASE 구동 이후부터 누적되는 통계정보로 종료 시 초기화 된다.

v$sesstat

"세션"의 연산에 대한서만 통계정보를 나타낸다. 세션이 종료되면 관련 통계정보는 사라진다.

데이터 파일 I/O

v$filestat

디스크 테이블스페이스의 데이터 파일 별 I/O 통계정보를 나타낸다.

기타 주요 성능뷰

그 외 주요 통계정보를 제공하는 성능뷰는 아래와 같다. 성능뷰 자체가 의미 있는 통계정보를 제공하므로 다른 성능뷰와 조인할 필요가 없다.

v$memstat

ALTIBASE가 현재 사용하는 메모리의 사용량을 모듈 별로 나타내는 성능뷰로 ALTIBASE 메모리 사용량이 비정상적일 때 주요 참고 지표가 된다.

모듈의 최고 메모리 사용량을 의미하는 컬럼 MAX_TOTAL_SIZE는 ALTIBASE 구동 시점부터 유지하는 것으로 ALTIBASE 종료 시 초기화 된다.

v$buffpool_stat

쿼리 수행시 Disk에서 페이지를 읽지 않고 버퍼풀의 기존 페이지를 재사용한 비율을 의미하는 “hit ratio”와 같은 버퍼풀 관련 통계정보를 실시간으로 나타낸다.

v$lfg

리두로그 파일 관련 성능뷰로 특정 컬럼 하나가 주요 모니터링 대상이다.

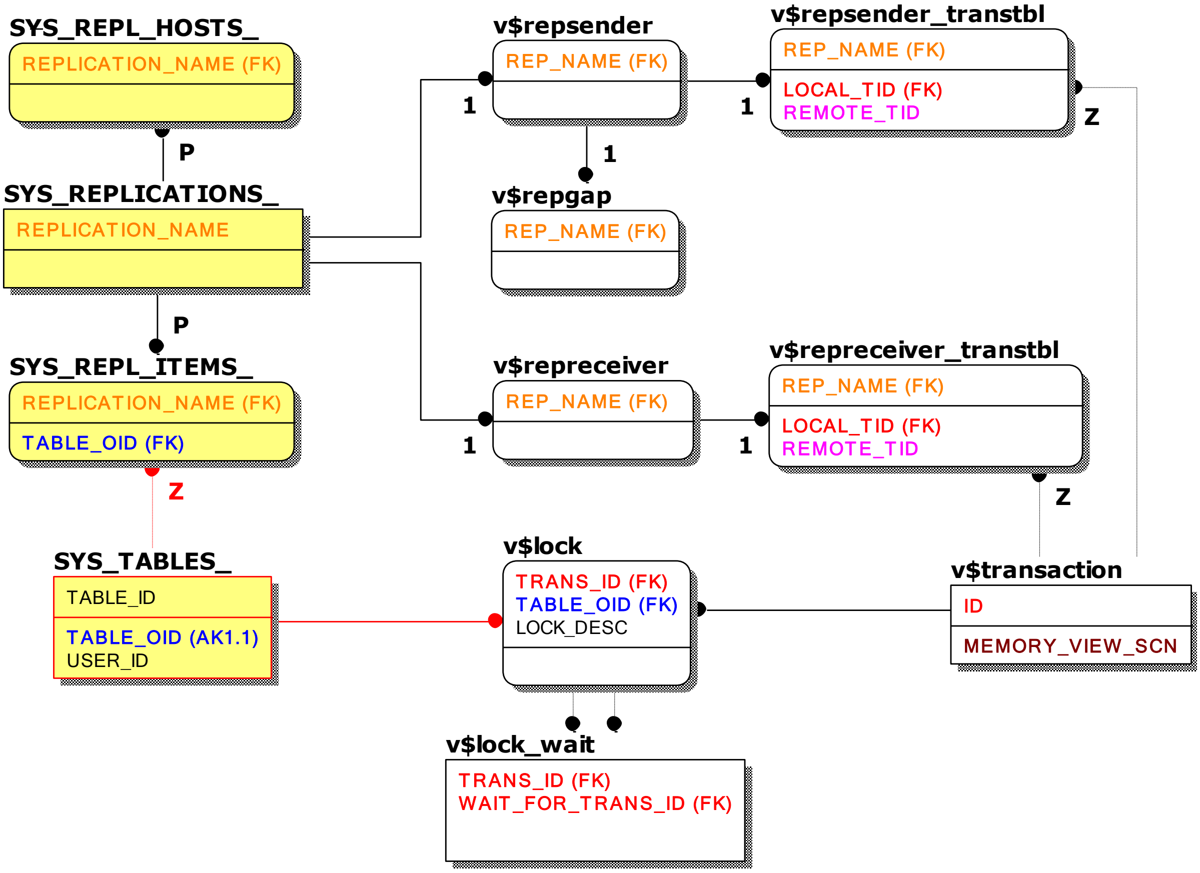
컬럼 LF_PREPARE_WAIT_COUNT는 현재 리두로그 파일에서 새로운 리두로그 파일로 switching 하려 할 때, 다음 리두로그 파일이 미처 생성되지 않아 service thread가 기다린 횟수를 나타낸다. 이 값이 크다면 PREPARE_LOG_FILE_COUNT 프로퍼티의 값을 더 큰 값으로 변경 후 적용(재구동)하여 충분한 개수의 리두로그 파일을 미리 만들어지도록 한다.

ALTIBASE 구동 시점부터 누적 및 합산되는 형태로 ALTIBASE 종료 시 초기화 된다.

이중화 관련 주요 메타테이블 및 성능뷰

이중화 관련 모니터링을 위한 조인쿼리 시 참고할 주요 키 컬럼만 표기한 메타테이블 및 성능뷰의 관계는 아래와 같다. 참고로 SYS_REPL_ITEMS_는 사용자명과 테이블이름을 알기 위해 SYS_USERS_와 SYS_TABLES_을 굳이 조인하지 않아도 되게끔 컬럼이름과 사용자이름이 함께 저장되어 있다.

또한, 앞서 언급했듯이 세션번호와 쿼리번호가 없는 이중화 트랜잭션 특성상 v$repsender_transtbl, v$repreceiver_transtbl을 통하여 이중화 상대 서버에서 대응되는 트랜잭션을 식별한다.



주요 메타테이블

SYS_REPLICATIONS_

모든 이중화 객체에 대한 정보가 저장된 메타테이블로 이중화 대상서버가 이중화를 반영한 시점(XSN)과 같은 상세한 정보를 확인할 수 있다.

SYS_REPL_HOSTS_

이중화 대상 서버의 정보가 저장된 메타테이블로 이중화 대상서버의 주소(HOST_IP)와 포트번호(PORT_NO)를 확인할 수 있다.

SYS_REPL_ITEMS_

이중화 대상 테이블에 대한 정보가 저장된 메타테이블이다.

주요 성능뷰

v$repsender

이중화 송신 쓰레드인 sender의 상태를 나타내는 성능뷰로 sender가 구동되어 있지 않으면 조회되지 않는다.

v$repgap

최신 리두로그일련번호와 sender가 전송한 리두로그일련번호의 간격(gap) 나타내는 성능뷰로 이중화 대상 서버간 동기화 정도를 의미하므로 이중화 모니터링 시 필수적으로 활용된다. sender에 의해 측정되므로 sender가 구동되어 있지 않으면 이 역시 조회되지 않는다.

v$repreceiver

이중화 수신 쓰레드인 receiver의 상태를 나타내는 성능뷰로 receiver가 구동되어 있지 않으면 조회되지 않는다.

v$repsender_transtbl

sender가 수행중인 이중화 트랜잭션에 대한 정보를 나타내는 성능뷰로 해당 트랜잭션과 대응되는 상대편 서버의 트랜잭션을 알 수 있다. 이 역시 sender가 구동되어 있지 않으면 조회되지 않는다.

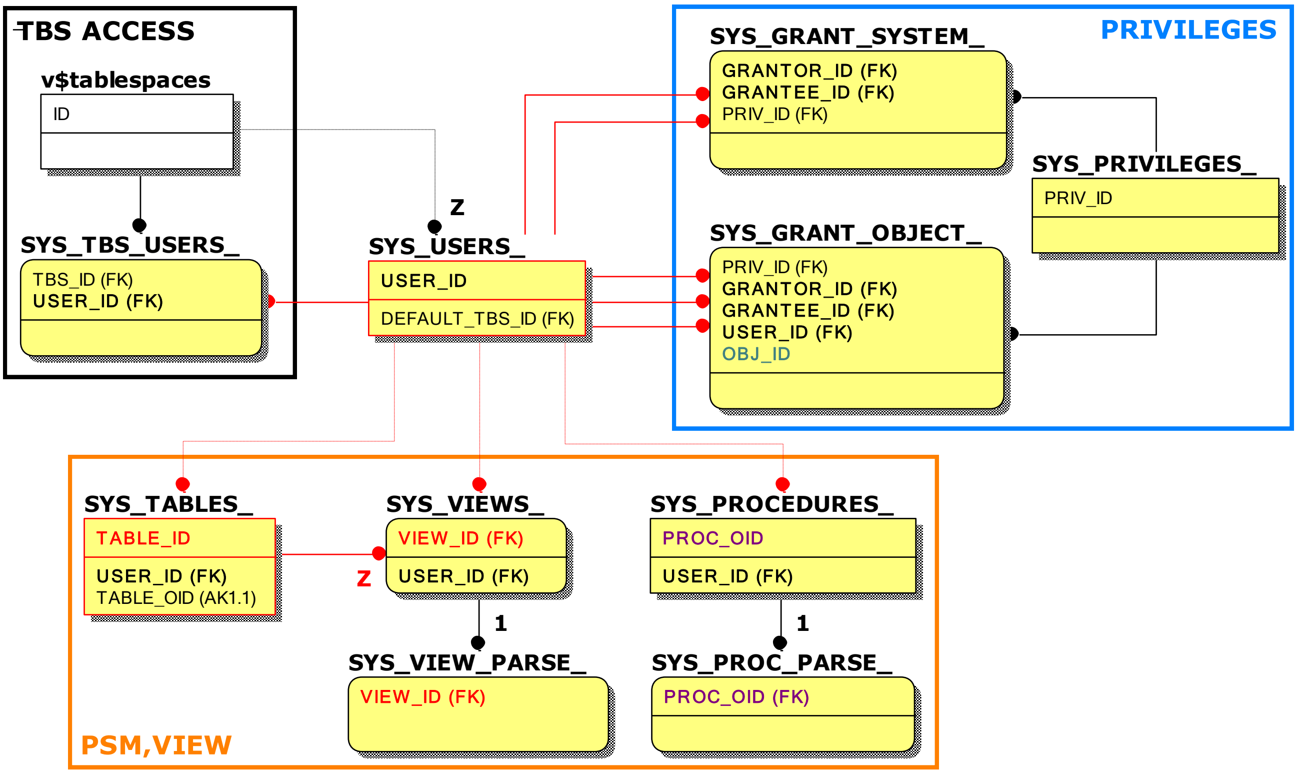
v$repreceiver_transtbl

receiver가 수행중인 이중화 트랜잭션에 대한 정보를 나타내는 성능뷰로 해당 트랜잭션과 대응되는 상대편 서버의 트랜잭션을 알 수 있다. 이중화 트랜잭션의 lock으로 인해 로컬 트랜잭션이 대기하는 현상이 발생할 때 주요 참고지표가 된다. 이 역시 receiver가 구동되어 있지 않으면 조회되지 않는다.

사용자의 테이블스페이스 접근 가능여부, 시스템/객체 권한, PSM, 뷰 관련 주요 메타테이블 및 성능뷰

실시간 모니터링이 필요하진 않으나 운영업무 중 자주 확인되는 요소로 "사용자의 테이블스페이스 접근 가능여부", "시스템/객체 권한", "스키마객체"를 들 수 있다. 스키마객체의 경우는 PSM(프로시저/펑션/타입세트), 뷰가 대표적이다.

관련 정보를 얻기 위해 조인쿼리 시 참고할 주요 키 컬럼만 표기한 메타테이블 및 성능뷰의 관계는 아래와 같다.



참고로 SYS_GRANT_OBJECT_에서 객체번호인 OBJ_ID 컬럼(녹색)에 대응하는 다른 객체의 컬럼이름은 객체에 따라 서로 달라지므로 주의가 필요하다. 여기서는 TABLE_ID(붉은색), VIEW_ID(붉은색), PROC_OID(보라색) 컬럼을 예로 들 수 있다.

또한, 대부분의 데이터베이스 객체는 그에 대응하는 별도의 메타테이블을 가지나 뷰의 경우는 테이블과 함께 관리되므로 뷰의 이름과 같은 구체적인 정보는 SYS_TABLES_에서 확인하여야만 한다. SYS_VIEWS_는 뷰의 컴파일 여부만 저장되어 있다. (즉, SYS_VIEWS_의 VIEW_ID는 SYS_TABLES_의 TABLE_ID와 같다.)

주요 메타테이블

SYS_TBS_USERS_

사용자가 접근 가능한 테이블스페이스 목록이 저장된 메타테이블이다.

SYS_PRIVILEGES_

시스템권한, 객체권한에 대한 권한번호(PRIV_ID)와 권한이름(PRIV_NAME)이 저장된 메타테이블이다.

SYS_GRANT_SYSTEM_

사용자가 부여 받은 시스템권한에 대한 권한번호(PRIV_ID)가 저장된 메타테이블이다.

SYS_GRANT_OBJECT_

사용자가 부여 받은 객체권한에 대한 권한번호(PRIV_ID)와 객체정보가 저장된 메타테이블로 객체번호(OBJ_ID)를 통해 해당 객체의 상세정보를 확인할 수 있다.

SYS_VIEWS_, SYS_VIEW_PARSE_

SYS_VIEWS_는 뷰의 컴파일상태만 저장된 메타테이블이다. 뷰의 생성구문은 SYS_VIEW_PARSE_를 통하여 확인할 수 있다.

SYS_PROCEDURES_, SYS_PROC_PARSE_

PSM(프로시져, 펑션, 타입세트)에 대한 상세한 정보가 저장된 메타테이블이다. 각 PSM에 대응하는 생성구문은 SYS_PROC_PARSE_를 통하여 확인할 수 있다.

모니터링 요소

ALTIBASE 모니터링 시 일반적으로 요구되는 사항을 최소단위로 분리, 이를 모니터링 요소라 칭하고 각 모니터링 요소를 표로 제시한다. 사용자는 아래 표를 참조하여 모니터링 하고자 하는 요소에 대응되는 쿼리 및 OS 명령어를 확인하여 활용하도록 한다. 쿼리 이해를 위한 메타테이블 및 성능뷰의 관계에 대한 간략한 설명은 "ALTIBASE 메타테이블 및 성능뷰 개요" 섹션을 참조하도록 한다.

표에서 각 항목의 의미는 아래와 같다.

- **구분** MEM(메모리DB만 해당), DISK(디스크DB만 해당), ALL(메모리/디스크DB 공통), REP(이중화 사용시)
- **그룹** "모니터링 요소"의 그룹
- **ID** "모니터링 요소"에 대한 인공식별자
- **모니터링 요소** 모니터링이 가능한 최소단위
- **비고** 세부사항 및 기타 참고사항

구분	그룹	ID	모니터링 요소	비고
ALL	Session	SS01	전체 세션 개수	
ALL		SS02	세션 정보	
ALL		SS03	SYSDBA 자격으로 접속중인 세션 정보	
ALL	Statement	ST01	전체 쿼리 개수	
ALL		ST02	쿼리 정보	TIMED_STATISTICS 활성필요
ALL		ST03	현재 수행중인 쿼리 개수	
ALL		ST04	현재 수행중인 쿼리 정보	TIMED_STATISTICS 활성필요
ALL		ST05	장시간 수행 쿼리 정보	10 분 이상 (600 초) / TIMED_STATISTICS 활성필수
ALL		ST06	장시간 수행되는 DML 트랜잭션의 마지막 쿼리 정보	한 시간이상 (3600 초) / TIMED_STATISTICS 활성필요
ALL		ST07	풀 스캔 쿼리 정보	인덱스 미사용, 모든 페이지를 접근하는 쿼리
ALL		ST08	풀 스캔 쿼리 수행횟수 통계	
ALL		ST09	세션 별 쿼리 목록	
ALL	Service Thread	SV01	서비스쓰레드 상태	
ALL	Transaction & lock	TL01	트랜잭션 및 Lock 정보	이중화 트랜잭션 포함
ALL	redo logfile	LO01	리두로그 파일 정보	
ALL		LO02	리두로그 파일 prepare 대기 누적횟수	
MEM	G C	GC01	메모리 DB GC gap	
MEM		GC02	GC 가 대기하고 있는 트랜잭션에서 수행중인 쿼리	순간적인 CPU 소모가 큰 쿼리 검출용으로 활용 가능
ALL	Memory	MS01	ALTIBASE 의 메모리 사용현황 (모듈단위)	
ALL		MS02	ALTIBASE 의 메모리 사용량 합계	
MEM	TBS(tablespace)	TS01	메모리 테이블스페이스 사용량	
MEM		TS02	전체 메모리 테이블스페이스 사용량	
DISK		TS03	디스크 테이블스페이스 사용량	
DISK		TS04	디스크 언두 테이블스페이스 사용량	[TS03]에서 언두 테이블스페이스 부분만 추림.
DISK		TS05	트랜잭션 별 언두 테이블스페이스 사용량	[TS04]에서 트랜잭션 별 사용량 결과
ALL		TS06	전체 테이블스페이스 사용량	[TS01], [TS02], [TS03]을 UNION ALL 한 결과
MEM		TS07	메모리 테이블스페이스 데이터 파일 체크포인트 경로	
MEM		TS08	메모리 테이블스페이스 데이터 파일	
DISK		TS09	디스크 테이블스페이스 데이터 파일	
DISK		TS10	디스크 테이블스페이스 데이터 파일 별 I/O	
DISK		TS11	디스크 테이블스페이스 데이터 파일 별 단일 페이지 Read I/O	현재 멀티 페이지 Read 를 지원하지 않음.
DISK		TS12	임시 테이블스페이스 현재 사용량	
DISK		TS13	트랜잭션 별 임시 테이블스페이스 사용량	
ALL		TS14	전체 테이블스페이스 상태	
DISK	Disk Buffer	DB01	디스크 버퍼 Hit Ratio	
MEM	Object	OB01	메모리 테이블	
MEM		OB02	큐	
MEM		OB03	Efficiency 가 낮은 메모리 테이블	크기가 1024M, efficiency 가 50% 이하
MEM		OB04	메모리 인덱스, 큐 인덱스	
DISK		OB05	디스크 인덱스	
DISK		OB06	디스크 테이블	

DISK		OB07	디스크 파티션 정보	
ALL		OB08	시퀀스	
ALL		OB09	시노님	
ALL		OB10	PSM(프로시저/펑션/타입세트)	
ALL		OB11	PSM 생성구문	PSM 이름을 입력
ALL		OB12	VIEW	
ALL		OB13	VIEW 생성구문	
ALL		OB14	PACKAGE	
ALL		OB15	PACKAGE 서브프로그램 정보	
ALL		OB16	PACKAGE 생성구문	
ALL	Privileges	PV01	사용자 시스템 권한	
ALL		PV02	사용자 오브젝트 권한	
ALL	Constraints	CT01	전체 제약조건 목록	
ALL		CT02	PK, FK, UNIQUE 관련 제약조건 및 테이블, 인덱스 목록	
ALL		CT03	복합 인덱스 컬럼 구성 목록	
ALL		CT04	인덱스 정보 요약	
REP	Replication	RP01	이중화 sender 정보	
REP		RP02	이중화 receiver 정보	
REP		RP03	이중화갭	
REP		RP04	이중화 전체 현황	
REP		RP05	이중화를 수행하지 못해 누적된 리두로그 파일 측정	응용에 따라 간접적으로 MB 환산이 가능.
REP		RP06	이중화 대상 테이블 목록	

모니터링 쿼리

각 모니터링 요소에 대응하는 쿼리로 모니터링 목적에 따라 SELECT, WHERE, LIMIT절을 변경하도록 한다. 컬럼에 대한 간략한 설명은 있으나 컬럼이름이 직관적이어서 별도의 설명이 필요 없다고 판단되는 경우, 이전에 이미 설명한 컬럼이 중복되어 나오는 경우에는 생략하였다. 컬럼에 대한 보다 상세한 설명과 이해를 위해서 기본적으로 ATC (http://atc.altibase.com)에 게시된 운영자 매뉴얼을 함께 참조하는 것을 권장 한다.

Session

세션의 상태를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다. 특정 세션에 관한 정보만 모니터링 하려면 WHERE절에 v$session의 "id" 컬럼을 기술하면 된다.

[SS01] 전체 세션 개수 [back]

```
SELECT count(*) tot_stmt_cnt FROM v$session
;
```

[SS02] 세션 정보 [back]

```
SELECT a.id session_id,
      a.db_username user_name,
      replace2(replace2(a.comm_name, 'socket-', null), '-server', null) client_ip,
      a.client_app_info,
      a.client_pid,
      a.session_state,
      decode(a.autocommit_flag, 1, 'ON', 'OFF') autocommit,
      decode(a.login_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') + a.login_time /
(24*60*60), 'mm/dd hh:mi:ss')) login_time,
      decode(a.idle_start_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') +
a.idle_start_time / (24*60*60), 'mm/dd hh:mi:ss')) idle_time,
      nvl(ltrim(b.query), 'NONE') current_query
  FROM v$session a left outer join v$statement b on a.current_stmt_id = b.id
;
```

■ **주요 컬럼 설명**

client_ip	세션과 관련된 클라이언트 응용프로그램의 ip 주소.
client_app_info	세션과 관련된 클라이언트 응용프로그램의 이름.
client_pid	세션과 관련된 클라이언트 응용프로그램의 프로세스아이디로 클라이언트 응용프로그램이 실행되는 OS에서 관련 프로세스를 식별 가능하다.
session_state	세션의 상태를 나타내는 문자열로 INIT, AUTH, SERVICE READY, SERVICE, END, ROLLBACK, UNKNOWN 7가지 상태가 있다.
idle_time	세션이 아무것도 하지 않기 시작한 시간으로 idle_timeout 의 기준이 된다. 단위는 유닉스 시간(unix time)이다.
current_query	세션에서 가장 마지막으로 수행하였거나 현재 수행중인 쿼리.

[SS03] SYSDBA 자격으로 접속중인 세션 정보 [back]

```
SELECT a.id session_id,
      a.db_username user_name,
      replace2(replace2(a.comm_name, 'socket-', null), '-server', null) client_ip,
      a.client_app_info,
      a.client_pid,
      a.session_state,
      decode(a.autocommit_flag, 1, 'ON', 'OFF') autocommit,
      decode(a.login_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') + a.login_time /
(24*60*60), 'mm/dd hh:mi:ss')) login_time,
      decode(a.idle_start_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') +
```

```
a.idle_start_time / (24*60*60), 'mm/dd hh:mi:ss')) idle_time,
      nvl(ltrim(b.query), 'NONE') current_query
  FROM v$session a left outer join v$statement b on a.current_stmt_id = b.id
 WHERE a.sysdba_flag = 1
;
```

Statement

쿼리의 상태를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다. 쿼리수행과 관련된 상세한 시간정보를 확인하기 위해서는 기본적으로 TIMED_STATISTICS가 활성화(1)되어 있어야 한다.

특정 세션에 관한 정보만 모니터링 하려면 WHERE절에 v$statement의 "session_id" 컬럼을 기술하면 된다.

[ST01] 전체 쿼리 개수 [back]

```
SELECT count(*) as stmt_cnt FROM v$statement
;
```

[ST02] 쿼리 정보 [back]

```
SELECT session_id,
      id stmt_id,
      tx_id,
      (parse_time+validate_time+optimize_time) prepare_time,
      fetch_time,
      execute_time,
      total_time,
      execute_flag,
      decode(last_query_start_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') +
last_query_start_time / (24*60*60), 'mm/dd hh:mi:ss')) last_start_time,
      nvl(ltrim(query), 'NONE') query
  FROM v$statement
 ORDER BY execute_time desc
;
```

■ **주요 컬럼 설명**

stmt_id	쿼리의 아이디.
tx_id	쿼리와 관련된 트랜잭션의 아이디.
prepare_time	prepare(parse, validation, optimization)를 수행하는데 소요된 시간으로 단위는 마이크로 초다. 쿼리가 수행될 때마다 갱신된다.
execute_time	prepare 완료 후 execution을 수행하는데 소요된 시간으로 쿼리가 수행될 때마다 갱신된다. query timeout의 기준이 되며 단위는 마이크로 초다.
fetch_time	쿼리에 대한 결과를 클라이언트가 가져갈(fetch) 때 소요되는 시간이다. 결과의 크기에 따라 하나의 쿼리는 여러 번의 fetch 를 수행할 수 있으며 이 때마다 갱신된다. fetch_timeout의 기준이 되며 단위는 마이크로 초이다.
total_time	하나의 쿼리가 수행되기 위해 소요된 시간의 합계를 의미하는 것으로 prepare, fetch, execution을 모두 포함한다. 쿼리가 수행될 때마다 갱신되며 단위는 마이크로 초다.
execute_flag	이 값이 0이면 prepare만 된 상태이며 1이면 execution 중인 상태이다.
last_start_time	가장 마지막으로 수행된 쿼리가 수행을 시작한 시간으로 단위는 유닉스 시간(unix time)이다. 이 값은 TIMED_STATISTICS 프로퍼티를 활성화(0)하지 않아도 갱신된다.

[ST03] 현재 수행중인 쿼리 개수 [back]

```
SELECT count(*) as active_stmt_cnt FROM v$statement WHERE execute_flag = 1
;
```

[ST04] 현재 수행중인 쿼리 정보 [back]

```
SELECT session_id,
      id stmt_id,
      tx_id,
      (parse_time+validate_time+optimize_time) prepare_time,
      fetch_time,
      execute_time,
      total_time,
      decode(last_query_start_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') +
last_query_start_time / (24*60*60), 'mm/dd hh:mi:ss')) last_start_time,
      nvl(ltrim(query), 'NONE') query
  FROM v$statement
 WHERE execute_flag = 1
 ORDER BY execute_time desc
;
```

[ST05] 장시간으로 수행 쿼리 정보 (600 초 기준) [back]

```
SELECT session_id,
      id stmt_id,
      tx_id,
      (parse_time+validate_time+optimize_time) prepare_time,
      fetch_time,
      execute_time,
      total_time,
      decode(last_query_start_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') +
last_query_start_time / (24*60*60), 'mm/dd hh:mi:ss')) last_start_time,
      nvl(ltrim(query), 'NONE') query
  FROM v$statement
 WHERE execute_flag = 1
      and execute_time/1000000 > 600
 ORDER BY execute_time desc
;
```

[ST06] 장시간 수행되는 DML트랜잭션의 마지막 쿼리 정보 (3600 초 기준) [back]

```
SELECT st.session_id,
      ss.comm_name client_ip,
      ss.client_pid,
      ss.client_app_info,
      (base_time - tr.first_update_time) as utrans_time,
      execute_time,
      total_time,
      decode(last_query_start_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') +
last_query_start_time / (24*60*60), 'mm/dd hh:mi:ss')) last_start_time,
      nvl(ltrim(st.query), 'NONE') query
  FROM v$transaction tr,
      v$statement st,
      v$sessionmgr,
      v$session ss
 WHERE tr.id = st.tx_id
      and st.session_id = ss.id
      and tr.first_update_time != 0 -- 0:read only transaction
      and (base_time - tr.first_update_time) > 3600
 ORDER BY utrans_time desc
;
```

■ 주요 컬럼 설명

utrans_time	트랜잭션이 최초에 변경연산을 시작한 시점을 기준으로 현재까지의 경과시간을 의미한다. utrans_timeout의 기준이 되며 단위는 초(sec)다. 참고로 이 값을 계산하기 위한 v$sessionmgr의 base_time과 v$transaction의 first_update_time은 TIMED_STATISTICS 프로퍼티를 활성화(0)하지 않아도 갱신된다.

[ST07] 풀 스캔 쿼리 정보 [back]

```
SELECT session_id,
       s.comm_name client_ip,
       s.client_pid,
       s.client_app_info,
       decode(last_query_start_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') +
last_query_start_time / (24*60*60), 'mm/dd hh:mi:ss')) last_start_time,
       (parse_time+validate_time+optimize_time) prepare_time,
       fetch_time,
       execute_time,
       total_time,
       nvl(ltrim(query), 'NONE') query
  FROM v$statement t,
       v$session s
 WHERE s.id = t.session_id
       and (mem_cursor_full_scan > 0 or disk_cursor_full_scan > 0)
       and upper(query) not like '%INSERT%'
 ORDER BY execute_time desc
;
```

[ST08] 풀 스캔 쿼리 수행횟수 통계 (현재 접속중인 세션에 한함) [back]

```
SELECT count(execute_success) execute_cnt,
       substr(ltrim(query),1,40) query
  FROM v$statement
 WHERE (mem_cursor_full_scan > 0 or disk_cursor_full_scan > 0)
        and upper(query) not like '%INSERT%'
 GROUP BY query
 ORDER BY execute_cnt desc
;
```

■ **주요 컬럼 설명**

execute_cnt	성공적으로 수행 완료된 쿼리 실행 회수의 합계

[ST09] 세션 별 쿼리 목록 [back]

```
SELECT session_id, id stmt_id, tx_id, substr(query,1,30) query FROM v$statement ORDER BY 1,2
;
```

Service Thread

서비스쓰레드와 관련된 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[SV01] 서비스쓰레드 상태 [back]

```
SELECT run_mode, state, count(*) cnt FROM v$service_thread GROUP BY run_mode, state
;
```

■ **주요 컬럼 설명**

run_mode	서비스쓰레드의 작동 모드. / SHARED, DEDICATED
state	서비스쓰레드의 상태. 일반적으로 POLL 또는 EXECUTE 상태이다. NONE : 서비스쓰레드가 초기화된 상태 POLL : 서비스쓰레드가 이벤트를 기다리고 있는 상태 QUEUE-WAIT : 서비스쓰레드가 작업 큐(Task Queue)를 대기하는 상태 EXECUTE : 서비스쓰레드가 쿼리를 수행중인 상태

Transaction & Lock

트랜잭션 및 Lock 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[TL01] Transaction 및 lock 정보 [back]

```
SELECT tx.id tx_id,
      wait_for_trans_id blocked_tx_id,
      decode(tx.status,
                  0, 'BEGIN',
                  1, 'PRECOMMIT',
                  2, 'COMMIT_IN_MEMORY',
                  3, 'COMMIT',
                  4, 'ABORT',
                  5, 'BLOCKED',
                  6, 'END') status,
      decode(tx.log_type, 0, u1.user_name, 'REPLICATION') user_name,
      decode(tx.log_type, 0, tx.SESSION_ID, rt.rep_name) session_id,
      decode(tx.log_type, 0, st.comm_name, rr.peer_ip) client_ip,
      decode(st.autocommit_flag, 1, 'ON', 'OFF') autocommit,
      l.lock_desc,
      decode(tx.first_update_time, 0, '0', to_char(to_date('1970010109','YYYYMMDDHH') +
tx.first_update_time / (60*60*24), 'MM/DD HH:MI:SS')) first_update_time,
      u2.user_name||'.'||t.table_name table_name,
      decode(tx.LOG_TYPE, 0, substr(st.query, 1, 10), 'REMOTE TX_ID '||remote_tid) current_query,
      decode(tx.DDL_FLAG,0, 'non-DDL', 'DDL') ddl,
      decode(tx.first_undo_next_lsn_fileno, -1, '-', tx.first_undo_next_lsn_fileno) 'logfile#'
  FROM v$transaction tx,
      v$lock l
      left outer join (SELECT st.*, ss.autocommit_flag, ss.db_userid, ss.comm_name
                        FROM v$statement st, v$session ss
                       WHERE ss.id = st.session_id
                            and ss.CURRENT_STMT_ID = st.id) st on l.trans_id = st.tx_id
      left outer join v$repreceiver_transtbl rt on l.trans_id = rt.local_tid
      left outer join v$repreceiver rr on rt.rep_name = rr.rep_name
      left outer join v$lock_wait lw on l.trans_id = lw.trans_id
      left outer join system_.sys_users_ u1 on st.db_userid = u1.user_id,
      system_.sys_tables_ t
      left outer join system_.sys_users_ u2 on t.user_id = u2.user_id
 WHERE tx.id = l.trans_id
      and t.table_oid = l.table_oid
      and tx.status != 6 -- 6:END
 ORDER BY tx.id, st.id, tx.first_update_time desc
;
```

■ 주요 컬럼 설명

tx_id	트랜잭션의 아이디.
blocked_tx_id	lock 획득 대기를 유발한 트랜잭션의 아이디로 없는 경우는 공백이다.
status	트랜잭션의 상태를 의미하는 0~6 까지의 숫자를 문자열로 나타낸다. BEGIN(0), PRECOMMIT(1), COMMIT_IN_MEMORY(2), COMMIT(3), ABORT(4), BLOCKED(5), END(6)
user_name	트랜잭션을 수행중인 사용자의 이름으로 이중화 트랜잭션은 "REPLICATION "이다.
session_id	트랜잭션을 수행중인 세션의 아이디로 이중화 트랜잭션은 해당 "이중화 객체 이름"이다.
client_ip	세션과 관련된 클라이언트 응용프로그램의 ip 주소로 이중화 트랜잭션은 "이중화 대상 서버"의 ip이다.
first_update_time	트랜잭션이 최초로 변경연산을 시작한 시간. SELECT만 수행하는 트랜잭션은 0 이다.
lock_desc	lock에 대한 문자열로 IX_LOCK, IS_LOCK, X_LOCK 등이 있다.
table_name	lock 획득 대상 테이블로 "사용자이름.테이블이름" 형식으로 나타낸다.
current_query	트랜잭션에서 가장 마지막으로 수행하였거나 현재 수행중인 쿼리. 이중화 트렌잭션의 경우는 상대편 서버의 ip 주소와 트랜잭션 아이디를 나타낸다.
ddl	트랜잭션의 DDL 여부. / DDL(0) non-DDL(1)
logfile#	트랜잭션과 관련된 리두로그 파일번호. SELECT만 수행하는 트랜잭션은 "-" 이다.

redo logfile

redo logfile 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[LO01] 리두로그 파일 정보 [back]

```
SELECT oldest_active_logfile oldest_logfile,
       current_logfile current_logfile,
       current_logfile-oldest_active_logfile logfile_gap
  FROM v$archive
;
```

■ 주요 컬럼 설명

oldest_logfile	가장 오래된 리두로그 파일의 번호.
current_logfile	현재 사용중인 온라인 리두로그 파일의 번호.
logfile_gap	현재 사용중인 온라인 리두로그 파일과 가장 오래된 리두로그 파일 사이의 리두로그 파일 개수.

[LO02] 리두로그 파일 prepare 대기 누적횟수 [back]

```
SELECT lf_prepare_wait_count FROM v$lfg
;
```

■ 주요 컬럼 설명

lf_prepare_wait_count	현재 리두로그 파일에서 새로운 리두로그 파일로 switching 하려 할 때, 다음 리두로그 파일이 미처 생성되지 않아 service thread가 기다린 횟수를 나타낸다. 이 값이 크다면 PREPARE_LOG_FILE_COUNT 프로퍼티의 값을 더 큰 값으로 변경 후 적용(재구동)하여 충분한 개수의 리두로그 파일을 미리 만들어지도록 한다. ALTIBASE 구동 시점부터 누적 및 합산되는 형태로 ALTIBASE 종료 시 초기화 된다.

GC

GC 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[GC01] 메모리DB GC gap [back]

```
SELECT gc_name, scnoftail, minmemscnintxs, add_oid_cnt-gc_oid_cnt gc_gap FROM v$memgc
;
```

■ 주요 컬럼 설명

gc_name	GC의 이름으로 2 개가 존재한다. MEM_LOGICAL_AGER는 인덱스의 old version을 삭제하는 GC이며 MEM_DELTHR는 테이블 레코드의 old version을 삭제하는 GC이다.
scnoftail	GC에 의해 확인된 삭제해야 할 old version중 가장 최신 version의 번호
minmemscnintxs	ALTIBASE 가장 오래된 old version의 번호
gc_gap	각 GC가 삭제해야 할 old version의 양의 의미하는 것으로 이 값이 클수록 삭제해야 할 old version의 양이 많다는 의미이다. 이 때 scnoftail > minmemscnintxs 이면 active 트랜잭션 때문에 삭제를 진행하지 못하는 것이고, 반대로 scnoftail < minmemscnintxs 이면 삭제는 계속 진행 중이지만 계속되는 갱신 트랜잭션의 완료로 인해 삭제해야 할 old version이 증가하고 있는 상황으로 판단할 수 있다.

[GC02] GC가 대기하고 있는 트랜잭션에서 수행중인 쿼리 [back]

GC가 대기하고 있는 트랜잭션이란 "ALTIBASE에서 가장 오래된 old version"을 참조하는 트랜잭션을 의미한다. 이와 같은 트랜잭션으로 인해 GC가 동작하지 않고 무한히 대기하게 된다면 메모리 사용량 또한 무한히 증가될 수 있으므로 GC gap 증가 시 확인요소 중 한가지다.

아래 쿼리를 수행함으로 GC가 대기하고 있는 트랜잭션을 수행하는 세션에서 수행한 쿼리를 확인할 수 있다. 또한, 인덱스 스캔은 하나 분포도가 좋지 않아 풀 스캔이나 다름없어 순간적인 CPU 소모를 하는 쿼리를 검출할 때도 유용하게 활용될 수 있다.

주의할 것은 상황에 따라 한번의 수행으로 검출되지 않을 수 있기에 주기적, 반복적으로 수행하여야 한다는 것이다.

```
SELECT session_id,
       total_time,
       execute_time,
       tx_id,
       query
  FROM v$statement
 WHERE tx_id in (SELECT id
                   FROM v$transaction
                  WHERE memory_view_scn = (SELECT minmemscnintxs FROM v$memgc limit 1))
       and execute_flag = 1
 ORDER BY 2 desc
;
```

Memory

OS에서 ALTIBASE가 점유하는 메모리 사용률 관련 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[MS01] ALTIBASE의 메모리 사용현황 (모듈단위) [back]

```
SELECT name,
      round(max_total_size/1024/1024) 'ALLOC_MAX(M)',
      round(alloc_size/1024/1024) 'ALLOC(M)'
  FROM v$memstat ORDER BY 3 desc
;
```

■ 주요 컬럼 설명

NAME	ALTIBASE를 구성하는 모듈의 이름으로 모듈의 목록은 ALTIBASE 버전에 따라 차이가 있을 수 있다. 주요 모듈에 대한 간략한 설명은 아래와 같다. • Storage_Disk_Ager : Disk Garbage Collection 에 사용되는 메모리 • Storage_Disk_Buffer : Disk Buffer 관리에 사용되는 메모리 • Storage_Disk_Collection : Direct Path Insert와 LOB 연산에 사용되는 메모리 • Storage_Disk_Datafile : Disk File 관리에 사용되는 메모리 • Storage_Disk_Index : Disk Index 관리에 사용되는 메모리 • Storage_Disk_Page : Disk Page 관리에 사용되는 메모리 • Storage_Disk_Recovery : Disk 복구에 사용되는 메모리 • Storage_Memory_Ager : Memory Garbage Collection 에 사용되는 메모리 • Storage_Memory_Collection : Memory Record 관리에 사용되는 메모리 • Storage_Memory_Interface : Cursor등의 Interface 관리에 사용되는 메모리 • Storage_Memory_Locking : Locking 관리에 사용되는 메모리 • Storage_Memory_Manager : Memory Data 저장에 사용되는 메모리 • Storage_Memory_Index : Index 관리에 사용되는 메모리 • Storage_Memory_Page : Memory Page 관리에 사용되는 메모리 • Storage_Memory_Recovery : Memory 복구에 사용되는 메모리 • Storage_Memory_Utility : Storage Manager Tool 에 사용되는 메모리 • Storage_Memory_Transaction : Transaction 관리에 사용되는 메모리 • Temp_Memory : Memory Index 저장에 사용되는 메모리 • Transaction_Table : Transaction Table 관리에 사용되는 메모리 • Transaction_OID_List : Garbage Collection의 OID 저장에 사용되는 메모리 • Transaction_Table_Info : Transaction Table 정보 관리에 사용되는 메모리

	• Index_Memory : Index 저장에 사용되는 메모리 • LOG_Memory : 로깅에 사용되는 메모리 • Query_Meta : meta 정보 관리에 사용되는 메모리 • Query_DML : dml 수행에 사용되는 메모리 • Query_Sequence : 시퀀스 관리에 사용되는 메모리 • Query_Replication : 이중화에 사용되는 메모리 • Query_PSM_Node : psm array 변수 관리에 사용되는 메모리 • Query_PSM_Execute : psm 수행에 사용되는 메모리 • Query_Prepare : 쿼리 prepare에 사용되는 메모리 • Query_Execute : 쿼리 execution에 사용되는 메모리 • Query_Transaction : trigger 수행에 사용되는 메모리 • Query_Binding : 쿼리 호스트변수 bind에 사용되는 메모리 • Query_Conversion : 쿼리 호스트변수 bind시 conversion에 사용되는 메모리 • Query_Common : 쿼리 관리에 사용되는 기타 메모리 • Mathematics : 각종 연산에 사용되는 메모리
ALLOC_MAX(M)	해당 모듈이 사용했던 최대 메모리 크기.
ALLOC(M)	해당 모듈이 현재 사용하고 있는 메모리 크기.

[MS02] ALTIBASE의 메모리 사용량 합계 [back]

```
SELECT round(sum(max_total_size)/1024/1024) 'ALLOC_MAX(M)',
      round(sum(alloc_size)/1024/1024) 'ALLOC(M)'
  FROM v$memstat
;
```

TBS(tablespace)

TBS 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[TS01] 메모리 테이블스페이스 사용량 [back]

```
SELECT decode(type, 0, 'MEMORY_DICTIONARY',
                  1, 'MEMORY_SYS_DATA',
                  2, 'MEMORY_USER_DATA',
                  8, 'VOLATILE_USER_DATA') TBS_TYPE,
       name TBS_NAME,
       decode(maxsize, 140737488322560, '', round(maxsize/1024/1024,2)) 'MAX(M)',
       round(allocated_page_count * page_size / 1024 / 1024, 2) 'TOTAL(M)',
       round(nvl(m.alloc_page_count-m.free_page_count,total_page_count)*page_size/1024/1024,2)
'ALLOC(M)',
       mt.used 'USED(M)',
       decode(maxsize, 140737488322560, round((m.alloc_page_count-m.free_page_count)*page_size/
mem_max_db_size*100, 2),
       decode(maxsize, 0, '-', round((m.alloc_page_count-m.free_page_count)*page_size/maxsize*100,
2))) 'USAGE(%)',
       round(mt.used / round(allocated_page_count * page_size / 1024 / 1024, 2)*100,2)
'TOTAL_USAGE(%)',
       decode(state,1,'OFFLINE',2,'ONLINE',5,'OFFLINE BACKUP',6,'ONLINE BACKUP',128,'DROPPED',
'DISCARDED') STATE,
       decode(autoextend_mode,1,'ON','OFF') 'AUTOEXTEND'
FROM v$database d, v$tablespaces t, v$mem_tablespaces m,
       (      SELECT tablespace_id, round(sum((fixed_used_mem + var_used_mem))/(1024*1024),3) used
              FROM v$memtbl_info
              GROUP by tablespace_id) mt
WHERE  t.id = m.space_id
       and id = mt.tablespace_id
UNION ALL
SELECT 'VOLATILE_USER_DATA' TBS_TYPE,
       space_name TBS_NAME,
       decode(v.max_size, 140737488322560, '', round(v.max_size/1024/1024,2)) 'MAX(M)',
```

```
        round(allocated_page_count * page_size / 1024 / 1024, 2) 'TOTAL(M)',
        round(nvl(v.alloc_page_count-v.free_page_count,total_page_count)*page_size/1024/1024,2)
'ALLOC(M)',
        '' 'USED(M)',
        decode(v.max_size, 140737488322560, round((v.alloc_page_count-
v.free_page_count)*page_size/d.mem_max_db_size*100, 2),
        decode(v.max_size, 0, '', round((v.alloc_page_count-
v.free_page_count)*page_size/v.max_size*100, 2))) 'USAGE(%)',
        '' 'TOTAL_USAGE(%)',
        decode(v.space_status,1,'OFFLINE',2,'ONLINE',5,'OFFLINE BACKUP',6,'ONLINE
BACKUP',128,'DROPPED', 'DISCARDED') STATE,
        decode(v.autoextend_mode,1,'ON','OFF') 'AUTOEXTEND'
FROM v$database d, v$tablespaces t, v$vol_tablespaces v
where  t.type = '8'
        and t.id = v.space_id
;
```

■ **주요 컬럼 설명**

MAX(M)	메모리 테이블스페이스가 최대로 할당 가능한 페이지의 합계로 페이지는 "사용중인 페이지"와 "빈 페이지"로 구분된다. 최대크기가 지정되지 않은 경우는 "UNDEFINED"나 0 으로 나타낸다.
TOTAL(M)	메모리 테이블스페이스가 현재까지 할당 받은 페이지의 합계. 즉, 현재까지 할당된 "사용중인 페이지"와 "빈 페이지"의 합계로 데이터 파일 크기와 대응한다. ALTIBASE 구동 시점엔 "사용중인 페이지"만 메모리에 적재된다.
ALLOC(M)	메모리 테이블스페이스가 현재까지 할당 받은 페이지중 "빈 페이지"를 제외한 "사용중인 페이지"만의 합계이다. 예를 들어, 100M 크기의 메모리 테이블에 DROP 또는 TRUNCATE를 수행하면 전체 페이지 합계는 변함없으나 페이지 반납을 통해 "사용중인 페이지"가 "빈 페이지"가 되므로 이 값이 0 에 가깝게 된다.
USED(M)	메모리 테이블스페이스의 "사용중인 페이지"중에서 "실제로 데이터가 적재된 페이지"의 합계이다. 예를 들어, ALLOC이 100M 크기인 메모리 테이블에 전체 DELELE를 수행하면 ALLOC은 100M로 변함없으나 USED는 0 에 가깝게 된다.
MAX_USAGE(%)	메모리 테이블스페이스가 "최대로 할당 가능한 페이지" 대비 "사용중인 페이지"에 대한 백분율.(즉,MAX/ USED) 딕셔너리 테이블스페이스와 같이 최대 크기가 지정되지 않은 메모리 테이블스페이스의 경우는 프로퍼티 MEM_MAX_DB_SIZE에 의해 정의되는 "전체 메모리 테이블스페이스가 최대로 할당 가능한 페이지" 대비로 계산되며 최대 크기를 지정하지 않은 경우는 "UNDEFINED"나 0 으로 나타낸다.
TOTAL_USAGE(%)	메모리 테이블스페이스가 "현재까지 할당받은 페이지" 대비 "사용중인 페이지"에 대한 백분율.(즉,TOTAL/ USED) 최대 값이 지정되지 않은 테이블스페이스는 maxsize가 0 이므로 MAX 대비 USED를 구할 수 없기 때문에 해당 값이 MAX_USAGE(%)의 사용율을 대체 한다.
AUTOEXTEND	메모리 테이블스페이스의 자동 확장여부. / ON(1), OFF(2)

[TS02] 전체 메모리 테이블스페이스 사용량 [back]

```
SELECT mem_max_db_size/1024/1024 'MAX(M)',
       round(mem_alloc_page_count*32/1024, 2) 'TOTAL(M)',
       trunc((mem_alloc_page_count-mem_free_page_count)*32/1024, 2) 'ALLOC(M)',
       (SELECT round(sum((fixed_used_mem + var_used_mem))/(1024*1024),3) FROM v$memtbl_info)
'USED(M)',
       trunc(((mem_alloc_page_count-mem_free_page_count)*32*1024)/mem_max_db_size, 4)*100 'USAGE(%)'
  FROM v$database
;
```

■ **주요 컬럼 설명**

MAX(M)	전체 메모리 테이블스페이스가 최대로 할당 가능한 페이지의 합계로 프로퍼티 MEM_MAX_DB_SIZE에 의해 정의된다.
TOTAL(M)	전체 메모리 테이블스페이스가 현재까지 할당 받은 페이지의 합계이다.
ALLOC(M)	전체 메모리 테이블스페이스가 현재까지 할당 받은 페이지중 "빈 페이지"를 제외한 "사용중인 페이지"만의 합계이다.
USED(M)	전체 메모리 테이블스페이스의 "사용중인 페이지"중에서 "실제로 데이터가 적재된 페이지"의 합계이다.
USAGE(%)	전체 메모리 테이블스페이스가 "최대로 할당 가능한 페이지" 대비 "사용중인 페이지"에 대한 백분율. (즉,ALLOC/MAX)

[TS03] 디스크 테이블스페이스 사용량 [back]

```
SELECT decode(type, 3, 'DISK_SYS_DATA',
                    4, 'DISK_USER_DATA',
                    5, 'DISK_SYS_TEMP',
                    6, 'DISK_USER_TEMP',
                    7, 'DISK_UNDO') TBS_TYPE,
       name TBS_NAME,
       to_char(round(d.max * page_size / 1024 /1024, 2)) 'MAX(M)',
       round(total_page_count * page_size / 1024 / 1024, 2) 'TOTAL(M)',
       decode(type, 7, round(
       (SELECT(sum(total_extent_count*page_count_in_extent)*page_size)/1024/1024 FROM v$udsegs)+
          (SELECT(sum(total_extent_count*page_count_in_extent)*page_size)/1024/1024 FROM
v$tssegs),2)
              ,round(allocated_page_count * page_size / 1024 / 1024, 2)) 'ALLOC(M)',
              decode(type, 3,  to_char(round(nvl(ds.used, 0)/1024/1024, 2)),
                           4,  to_char(round(nvl(ds.used, 0)/1024/1024, 2)), '-') 'USED(M)',
              decode(type, 7, round(((SELECT sum(total_extent_count*page_count_in_extent)
                                      FROM v$udsegs)+
                                     (SELECT sum(total_extent_count*page_count_in_extent)
                                      FROM v$tssegs)) / d.max* 100, 2),
                                          3, round((nvl(ds.used, 0))/(d.max*page_size)* 100, 2),
                                          4, round((nvl(ds.used, 0))/(d.max*page_size)* 100, 2),
                                         round(allocated_page_count / d.max * 100, 2)) 'USAGE(%)',
       '' 'TOTAL_USAGE(%)',
       decode(state, 1, 'OFFLINE', 2, 'ONLINE', 5, 'OFFLINE BACKUP', 6, 'ONLINE BACKUP', 128,
'DROPPED', 'DISCARDED') STATE,
       d.AUTOEXTEND
FROM v$tablespaces t left outer join(SELECT space_id, sum(total_used_size) USED
                                     FROM x$segment
                                     GROUP by space_id) ds
                                on ds.space_id = t.id,
       (SELECT  spaceid,
          sum(decode(maxsize, 0, currsize, maxsize)) as MAX,
          decode(max(autoextend), 1, 'ON', 'OFF') 'AUTOEXTEND'
        FROM v$datafiles
        GROUP by spaceid
       ) d
WHERE t.id = d.spaceid
;
```

■ 주요 컬럼 설명

MAX(M)	디스크 테이블스페이스가 최대로 할당 가능한 페이지의 합계로 페이지는 "사용중인 페이지"와 "빈 페이지"로 구분된다.
TOTAL(M)	디스크 테이블스페이스가 현재까지 할당 받은 페이지의 합계. 즉, 현재까지 할당된 "사용중인 페이지"와 "빈 페이지"의 합계로 데이터 파일 크기와 대응한다.
ALLOC(M)	디스크 테이블스페이스가 현재까지 할당 받은 페이지중 "빈 페이지"를 제외한 "사용중인 페이지"만의 합계이다. 예를 들어, 100M 크기의 디스크 테이블에 DROP 또는 TRUNCATE를 수행하면 전체 페이지 합계는 변함없으나 페이지 반납을 통해 "사용중인 페이지"가 "빈 페이지"가 되므로 이 값이 0 에 가깝게 된다.
USED(M)	디스크 테이블스페이스의 "사용중인 페이지"중에서 "실제로 데이터가 적재된 페이지"의 합계이다. 하지만, 디스크 테이블스페이스 관리 기법의 변화로 인해 ALTIBASE 5.1.5 부터 5.3 까지는 확인할 수 없다. ALTIBASE 4, 5.5 이상부터는 메모리 테이블스페이스처럼 확인이 가능하다.
USAGE(%)	디스크 테이블스페이스가 "최대로 할당 가능한 페이지" 대비 "사용중인 페이지"에 대한 백분율이다.(즉,ALLOC/TOTAL)

■ 5.1.5 변경사항

5.1.5 에는 v$udssegs와 v$tssegs가 없습니다. 아래와 같이 수정되어야 한다.

ALLOC(M)	decode(type, 7, round((SELECT sum(total_page_count-free_page_count)*(select page_size from v$tablespaces where id = 3)/1024/1024 FROM v$undo_seg), 2), round(allocated_page_count*page_size/1024/1024, 2)) **'ALLOC(M)'**
USAGE(%)	decode(type, 7, round((SELECT sum(total_page_count-free_page_count) FROM v$undo_seg) / d.max* 100, 2), round(allocated_page_count / d.max * 100, 2)) **'USAGE(%)'**

[TS04] 디스크 언두 테이블스페이스 사용량 [back]

```
select t.name TBS_NAME,
      round(d.max * page_size / 1024 /1024, 2) 'MAX(M)',
      round((total_page_count * page_size) / 1024 / 1024, 2) 'TOTAL(M)',
      round((u.total_ext_cnt*prop.extent_size)/1024/1024, 2) 'ALLOC(M)',
      round(((u.tx_ext_cnt+u.used_ext_cnt+u.unstealable_ext_cnt) * prop.extent_size)/1024/1024, 2)
'USED(M)',
      round((((u.tx_ext_cnt+u.used_ext_cnt+u.unstealable_ext_cnt) *
prop.extent_size)/(d.max*page_size))*100, 2) 'USAGE(%)',
      decode(state,1,'OFFLINE',2,'ONLINE',5,'OFFLINE BACKUP',6,'ONLINE BACKUP',128,'DROPPED',
'DISCARDED') STATE,
      d.autoextend
from v$tablespaces t,
      (select spaceid,
          sum(decode(maxsize, 0, currsize, maxsize)) as MAX,
          decode(max(autoextend),1,'on','off') 'AUTOEXTEND'
       from v$datafiles
       group by spaceid) d,
    v$disk_undo_usage u,
    (select value1 extent_size from v$property where name = 'SYS_UNDO_TBS_EXTENT_SIZE') PROP
where t.id = d.spaceid
    and t.id = 3 ;
```

주요 컬럼 설명

MAX(M)	언두 테이블스페이스의 최대 크기
TOTAL(M)	언두 테이블스페이스로 할당 받은 전체 크기
ALLOC(M)	현재까지 할당 받은 페이지 중 '빈 페이지'를 제외한 '사용 중인 페이지'만의 합계
USED(M)	사용 중인 페이지 중에서 트랜잭션에 의해 언두로 사용 중인 크기
USAGE(%)	사용량(MAX 대비 USED)
STATE	테이블스페이스 상태

[TS05] 트랜잭션 별 언두 테이블스페이스 사용량 [back]

```
select tx.id tx_id,
      decode(tx.session_id, -1, rep_tx.rep_name, tx.session_id) SESSION_ID,
      decode(tx.session_id, -1, 'REP'||' '||rep_r.peer_ip||':'||peer_port, s.comm_name) CLIENT_IP,
      decode(tx.session_id, -1, 'REPLICATION', u.user_name) USER_NAME,
      decode(s.autocommit_flag, 1, 'ON', 0, 'OFF', '', 'REPLICATION') AUTOCOMMIT,
      decode(tx.first_update_time, 0, '0', to_char(to_date('1970010109','yyyymmddhh') +
tx.first_update_time / (60*60*24), 'mm/dd hh:mi:ss')) TX_START_TIME,
      decode(stmt.last_query_start_time, '', '-', stmt.last_query_start_time)
LAST_QUERY_START_TIME,
      to_char(((ud_s.total_extent_count*ud_s.page_count_in_extent*tbs.page_size)/1024),
'999,999,999') 'UNDO_USED(KB)',
      decode(tx.session_id, -1, 'REMOTE TX_ID '||rep_tx.remote_tid,
replace(replace(replace(substring(stmt.query, 1,60) , chr(10), ' '),chr(13), ' '),chr(9),' '))
query
  from v$transaction tx left outer join (select stmt.tx_id , stmt.id stmt_id ,
decode(stmt.last_query_start_time, 0, '-', to_char(to_date('1970010109', 'yyyymmddhh') +
stmt.last_query_start_time / (24*60*60), 'yy/mm/dd hh:mi:ss')) LAST_QUERY_START_TIME , stmt.query
        from v$statement stmt ,
        (select tx_id, max(last_query_start_time) last_query_start_time
                              from v$statement
                           group by tx_id) last_query
       where (stmt.tx_id, stmt.last_query_start_time) in ((last_query.tx_id,
last_query.last_query_start_time)) ) stmt on tx.id = stmt.tx_id left outer join
v$repreceiver_transtbl rep_tx on tx.id = rep_tx.local_tid left outer join v$repreceiver rep_r on
rep_tx.rep_name = rep_r.rep_name left outer join v$session s on tx.session_id = s.id left outer
join system_.sys_users_ u on s.db_userid = u.user_id
    , v$tablespaces tbs
    , v$txsegs tx_s
    , v$udsegs ud_s
```

```
where 1=1
  and tx.id = tx_s.trans_id
  and tx_s.id = ud_s.txseg_entry_id
  and ud_s.space_id = tbs.id ;
```

주요 컬럼 설명

TX_ID	트랜잭션 ID
SESSION_ID	세션ID. 이중화 트랜잭션의 경우 이중화 객체 이름
CLIENT_IP	클라이언트 IP. 이중화의 경우 PEER IP
USER_NAME	데이터베이스 유저. 이중화 트랜잭션의 경우 'REPLICATION'
AUTOCOMMIT	AUTOCOMMIT MODE. 이중화 트랜잭션의 경우 'REPLICATION'
TX_START_TIME	트랜잭션의 시작 시각
LAST_QUERY_START_TIME	트랜잭션 내 SQL문 시작 시각
UNDO_USED(KB)	트랜잭션의 언두 테이블스페이스 사용량
QUERY	SQL문장

[TS06] 전체 테이블스페이스 사용량 [back]

```
SELECT 'MEMORY_DB_TOTAL' TBS_TYPE,
       '-' TBS_NAME,
       to_char(mem_max_db_size/1024/1024) 'MAX(M)',
       round(mem_alloc_page_count*32/1024, 2) 'TOTAL(M)',
       trunc((mem_alloc_page_count-mem_free_page_count)*32/1024, 2) 'ALLOC(M)',
       to_char((select round(sum((fixed_used_mem + var_used_mem))/(1024*1024),3) from
v$memtbl_info)) 'USED(M)',
       trunc(((mem_alloc_page_count-mem_free_page_count)*32*1024)/mem_max_db_size, 4)*100
'USAGE(%)',
       '' 'TOTAL_USAGE(%)',
       '' STATE,
       '' 'AUTOEXTEND'
FROM v$database
UNION ALL
SELECT decode(type, 0, 'MEMORY_DICTIONARY',
                    1, 'MEMORY_SYS_DATA',
                    2, 'MEMORY_USER_DATA',
                    8, 'VOLATILE_USER_DATA') TBS_TYPE,
       name TBS_NAME,
       decode(maxsize, 140737488322560, '', round(maxsize/1024/1024,2)) 'MAX(M)',
       round(allocated_page_count * page_size / 1024 / 1024, 2) 'TOTAL(M)',
       round(nvl(m.alloc_page_count-m.free_page_count,total_page_count)*page_size/1024/1024,2)
'ALLOC(M)',
       mt.used 'USED(M)',
       decode(maxsize, 140737488322560, round((m.alloc_page_count-m.free_page_count)*page_size/
mem_max_db_size*100, 2),
       decode(maxsize, 0, '', round((m.alloc_page_count-m.free_page_count)*page_size/maxsize*100,
2))) 'USAGE(%)',
       round(mt.used / round(allocated_page_count * page_size / 1024 / 1024, 2)*100,2)
'TOTAL_USAGE(%)',
       decode(state,1,'OFFLINE',2,'ONLINE',5,'OFFLINE BACKUP',6,'ONLINE BACKUP',128,'DROPPED',
'DISCARDED') STATE,
       decode(autoextend_mode,1,'ON','OFF') 'AUTOEXTEND'
FROM v$database d, v$tablespaces t, v$mem_tablespaces m,
      (       SELECT tablespace_id, round(sum((fixed_used_mem + var_used_mem))/(1024*1024),3) used
              FROM v$memtbl_info
              GROUP by tablespace_id) mt
WHERE t.id = m.space_id
      and id = mt.tablespace_id
UNION ALL
SELECT  'VOLATILE_USER_DATA' TBS_TYPE,
       space_name TBS_NAME,
       decode(v.max_size, 140737488322560, '', round(v.max_size/1024/1024,2)) 'MAX(M)',
       round(allocated_page_count * page_size / 1024 / 1024, 2) 'TOTAL(M)',
```

```
      round(nvl(v.alloc_page_count-v.free_page_count,total_page_count)*page_size/1024/1024,2)
'ALLOC(M)',
      '' 'USED(M)',
      decode(v.max_size, 140737488322560, round((v.alloc_page_count-
v.free_page_count)*page_size/d.mem_max_db_size*100, 2),
      decode(v.max_size, 0, '', round((v.alloc_page_count-
v.free_page_count)*page_size/v.max_size*100, 2))) 'USAGE(%)',
      '' 'TOTAL_USAGE(%)',
      decode(v.space_status,1,'OFFLINE',2,'ONLINE',5,'OFFLINE BACKUP',6,'ONLINE
BACKUP',128,'DROPPED', 'DISCARDED') STATE,
      decode(v.autoextend_mode,1,'ON','OFF') 'AUTOEXTEND'
FROM v$database d, v$tablespaces t, v$vol_tablespaces v
WHERE t.type = '8'
      and t.id = v.space_id
UNION ALL
SELECT  decode(type, 3, 'DISK_SYS_DATA',
                     4, 'DISK_USER_DATA',
                     5, 'DISK_SYS_TEMP',
                     6, 'DISK_USER_TEMP',
                     7, 'DISK_UNDO') TBS_TYPE,
      name TBS_NAME,
      to_char(round(d.max * page_size / 1024 /1024, 2)) 'MAX(M)',
      round(total_page_count * page_size / 1024 / 1024, 2) 'TOTAL(M)',
      decode(type, 7, round(
         (SELECT(sum(total_extent_count*page_count_in_extent)*page_size)/1024/1024 FROM v$udsegs)+
         (SELECT(sum(total_extent_count*page_count_in_extent)*page_size)/1024/1024 FROM
v$tssegs),2)
             ,round(allocated_page_count * page_size / 1024 / 1024, 2)) 'ALLOC(M)',
             decode(type, 3, to_char(round(nvl(ds.used, 0)/1024/1024, 2)),
                          4, to_char(round(nvl(ds.used, 0)/1024/1024, 2)), '') 'USED(M)',
             decode(type, 7, round(((SELECT sum(total_extent_count*page_count_in_extent)
                                  FROM v$udsegs)+
                                  (SELECT sum(total_extent_count*page_count_in_extent)
                                  FROM v$tssegs)) / d.max* 100, 2),
                                      3, round((nvl(ds.used, 0))/(d.max*page_size)* 100, 2),
                                      4, round((nvl(ds.used, 0))/(d.max*page_size)* 100, 2),
                                     round(allocated_page_count / d.max * 100, 2)) 'USAGE(%)',
      '' 'TOTAL_USAGE(%)',
      decode(state, 1, 'OFFLINE', 2, 'ONLINE', 5, 'OFFLINE BACKUP', 6, 'ONLINE BACKUP', 128,
'DROPPED', 'DISCARDED') STATE,
      d.AUTOEXTEND
FROM v$tablespaces t left outer join(SELECT space_id, sum(total_used_size) USED
                                     FROM x$segment
                                     GROUP by space_id) ds
                                on ds.space_id = t.id,
      (SELECT  spaceid,
         sum(decode(maxsize, 0, currsize, maxsize)) as MAX,
         decode(max(autoextend), 1, 'ON', 'OFF') 'AUTOEXTEND'
       FROM v$datafiles
       GROUP by spaceid
      ) d
WHERE t.id = d.spaceid
;
```

■ 5.1.5 변경사항

5.1.5 에는 v$udssegs와 v$tssegs가 없습니다. 디스크 테이블스페이스 쿼리의 컬럼은 아래와 같이 수정되어야 합니다.

ALLOC(M)	decode(type, 7, round((SELECT sum(total_page_count-free_page_count)*(select page_size from v$tablespaces where id = 3)/1024/1024 FROM v$undo_seg), 2), round(allocated_page_count*page_size/1024/1024, 2)) **'ALLOC(M)'**
USAGE(%)	decode(type, 7, round((SELECT sum(total_page_count-free_page_count) FROM v$undo_seg) / d.max* 100, 2), round(allocated_page_count / d.max * 100, 2)) **'USAGE(%)'**

[TS07] 메모리 테이블스페이스 데이터 파일 체크포인트 경로 [back]

```
SELECT m.space_id tbs_id, space_name tbs_name, checkpoint_path
  FROM v$mem_tablespaces m, v$mem_tablespace_checkpoint_paths c
 WHERE m.space_id = c.space_id
;
```

[TS08] 메모리 테이블스페이스 데이터 파일 [back]

```
SELECT mem_data_file datafile_name FROM v$stable_mem_datafiles
;
```

■ 주요 컬럼 설명

datafile_name	데이터 파일의 이름. 물리적인 경로와 파일명까지 나타낸다.

[TS09] 디스크 테이블스페이스 데이터 파일 [back]

```
SELECT b.name tbs_name,
       a.id 'FILE#',
       a.name datafile_name,
       currsize*8/1024 'ALLOC(M)',
       round(case2(a.maxsize=0, currsize, a.maxsize)*8/1024) 'MAX(M)',
       decode(autoextend, 0, 'OFF', 'ON') 'AUTOEXTEND'
  FROM v$datafiles a,
       v$tablespaces b
 WHERE b.id = a.spaceid
 ORDER BY b.name, a.id
;
```

■ 주요 컬럼 설명

FILE#	데이터 파일의 번호. 하나의 디스크 테이블스페이스는 여러 개의 데이터 파일을 가질 수 있다.

[TS10] 디스크 테이블스페이스 데이터 파일 별 I/O [back]

```
SELECT name tbs_name,
       a.phyrds phy_read,
       a.phywrts phy_write,
       a.phyrds+a.phywrts phy_total,
       trunc(a.phyrds/read_sum*100,2) 'READ(%)',
       trunc(a.phywrts/write_sum*100,2) 'WRITE(%)',
       trunc( (a.phyrds+a.phywrts) / (read_sum+write_sum) * 100 , 2) 'TOTAL(%)',
       a.avgiotim avg_io_time
  FROM v$filestat a,
       v$datafiles b,
       (SELECT sum(phyrds) read_sum,
               sum(phywrts) write_sum
          FROM v$filestat ) c
 WHERE a.spaceid = b.spaceid
       and a.fileid = b.id
       and read_sum > 0
       and write_sum > 0
 ORDER BY a.phyrds+a.phywrts desc, rownum desc
;
```

■ 주요 컬럼 설명

phy_read	물리적 Read I/O 발생 회수.
phy_write	물리적 Write I/O 발생 회수.
phy_total	물리적 I/O 발생 회수.
read_per	전체 물리적 Read I/O 대비 발생 비율.
write_per	전체 물리적 Write I/O 대비 발생 비율.
total_per	전체 물리적 I/O 대비 발생 비율.

avg_io_time	평균 물리적 I/O 시간으로 단위는 밀리 초다.

[TS11] 디스크 테이블스페이스 데이터 파일 별 단일 페이지 Read I/O [back]

```
SELECT b.name tbs_name,
      a.singleblkrds read_cnt_per_page,
      a.singleblkrdtim read_time_per_page,
      trunc(a.singleblkrdtim/a.singleblkrds,2) average_time
  FROM v$filestat a,
      v$datafiles b
 WHERE a.spaceid = b.spaceid
      and a.fileid = b.id
      and a.singleblkrds > 0
 ORDER BY average_time desc
;
```

■ 주요 컬럼 설명

singleblkrds	단일 페이지에 대한 Read 개수.
singleblkrdtim	단일 페이지에 대한 Read 시간으로 단위는 밀리 초이다.
average_wait	단일 페이지에 대한 평균 Read 시간으로 단위는 밀리 초이다.

[TS12] 임시 테이블스페이스 사용량 [back]

```
SELECT sum(normal_area_size) SUM_NORMAL_AREA_SIZE
FROM x$temptable_stats
WHERE drop_time = '19700101_090000'
;
```

■ 주요 컬럼 설명

NORMAL_AREA_SIZE	사용한 Temp Tablespace 공간의 크기

[TS13] 트랜잭션 별 임시 테이블스페이스 사용량 [back]

```
select NORMAL_AREA_SIZE
from x$temptable_stats t, v$statement stmt
where t.TRANSACTION_ID = stmt.TX_ID
;
```

■ 주요 컬럼 설명

NORMAL_AREA_SIZE	사용한 Temp Tablespace 공간의 크기

[TS14] 전체 테이블스페이스 상태 [back]

```
SELECT name tbs_name,
      decode(state,
             1, 'OFFLINE',
             2, 'ONLINE',
             5, 'OFFLINE BACKUP',
             6, 'ONLINE BACKUP',
             128, 'DROPPED',
             'DISCARD') state
  FROM v$tablespaces
;
```

Disk Buffer

Disk Buffer 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[DB01] 디스크 버퍼 Hit Ratio [back]

```
SELECT hit_ratio 'HIT_RATIO(%)' FROM v$buffpool_stat
;
```

■ 주요 컬럼 설명

HIT_RATIO(M)	ALTIBASE 구동 이후부터 Disk Buffer에 이미 적재된 page가 재사용되는 비율을 의미하는 것으로 이 값이 작을수록 물리적인 읽기(read page) 횟수가 많다는 것이다. 물리적인 읽기 횟수가 많으면, 시스템이 빠른 질의 처리를 못한다. 이 컬럼은 (get_pages + fix_pages - read_pages) / (get_pages + fix_pages) 계산을 수행한 것이다.

Object

object 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[OB01] 메모리 테이블 [back]

```
SELECT a.user_name,
       b.table_name,
       d.name tablespace_name,
       c.table_oid,
       round((c.fixed_alloc_mem + c.var_alloc_mem)/(1024*1024),2) 'ALLOC(M)',
       round((c.fixed_used_mem + c.var_used_mem)/(1024*1024),2) 'USED(M)',
       round((c.fixed_used_mem + c.var_used_mem)/(c.fixed_alloc_mem + c.var_alloc_mem)*100,2)
'EFFICIENCY(%)'
  FROM system_.sys_users_ a,
       system_.sys_tables_ b,
       v$memtbl_info c,
       v$tablespaces d
 WHERE a.user_name <> 'SYSTEM_'
       and b.table_type = 'T'
       and a.user_id = b.user_id
       and b.table_oid = c.table_oid
       and b.tbs_id = d.id
;
```

■ 주요 컬럼 설명

ALLOC(M)	테이블이 할당 받은 페이지의 합계이다.
USED(M)	테이블이 할당 받은 페이지중에서 "실제로 데이터가 적재된 페이지"의 합계이다. 예를 들어, ALLOC이 100M 크기인 메모리 테이블에 전체 DELELE를 수행하면 ALLOC은 100M로 변함없으나 USED는 0 에 가깝게 된다.
EFFICIENCY(%)	테이블이 소유한 페이지중에서 "실제로 데이터가 적재된 페이지"에 대한 백분율로 공간 효율성을 나타낸다. (즉, USED/ALLOC)

[OB02] 큐 [back]

```
SELECT a.user_name,
       b.table_name,
       d.name tablespace_name,
       c.table_oid,
       round((c.fixed_alloc_mem + c.var_alloc_mem)/(1024*1024),2) 'ALLOC(M)',
       round((c.fixed_used_mem + c.var_used_mem)/(1024*1024),2) 'USED(M)',
       round((c.fixed_used_mem + c.var_used_mem)/(c.fixed_alloc_mem + c.var_alloc_mem)*100,2)
'EFFICIENCY(%)'
  FROM system_.sys_users_ a,
       system_.sys_tables_ b,
       v$memtbl_info c,
       v$tablespaces d
 WHERE a.user_name <> 'SYSTEM_'
```

```
      and b.table_type = 'Q'
      and a.user_id = b.user_id
      and b.table_oid = c.table_oid
      and b.tbs_id = d.id
;
```

[OB03] Efficiency 가 낮은 메모리 테이블 (크기가 1024M, Efficiency가 50% 이하) [back]

```
SELECT c.user_name,
      b.table_name,
      round((fixed_alloc_mem+var_alloc_mem)/1024/1024, 2) as 'ALLOC(M)',
      round((((fixed_alloc_mem+var_alloc_mem)-(fixed_used_mem+var_used_mem))/1024/1024),2)
'FREE(M)',
      round((fixed_used_mem+var_used_mem)/(fixed_alloc_mem+var_alloc_mem+0.01)*100, 2)
'EFFICIENCY(%)'
  FROM v$memtbl_info a ,
      system_.sys_tables_ b,
      system_.sys_users_ c
 WHERE a.table_oid = b.table_oid
      and b.user_id = c.user_id
      and round((fixed_alloc_mem+var_alloc_mem)/1024/1024, 2) >= 1024
      and round((fixed_used_mem+var_used_mem)/(fixed_alloc_mem+var_alloc_mem+0.01)*100, 2) <= 50
      and b.user_id <> 1
 ORDER BY 'FREE(M)' desc
;
```

■ 주요 컬럼 설명

FREE(M)	테이블이 할당 받은 페이지에서 "실제로 데이터가 적재된 페이지"를 제외한 페이지의 합계 (즉, ALLOC-USED)

[OB04] 메모리 인덱스, 큐 인덱스 [back]

```
SELECT c.user_name,
      decode(f.table_type, 'Q', 'QUEUE', 'T', 'TABLE') object_type,
      table_name object_name,
      d.space_name tablespace_name,
      e.index_name,
      rpad(case2(e.index_type=1, 'b-tree', 'r-tree'),10,' ') index_type,
      '16 bytes * rowcount' 'ALLOC'
  FROM v$index b,
      system_.sys_users_ c,
      v$mem_tablespaces d,
      system_.sys_indices_ e,
      system_.sys_tables_ f
 WHERE b.index_id = e.index_id
      and e.user_id = c.user_id
      and f.user_id = e.user_id
      and f.tbs_id = d.space_id
      and f.table_oid = b.table_oid
      and c.user_name <> 'SYSTEM_'
;
```

[OB05] 디스크 인덱스 및 사용량 [back]

```
-- NON-PARTITIONED INDEX
SELECT u.user_name as 'USER_NAME',
       tbl.table_name as 'TABLE_NAME',
       decode(tbl.is_partitioned, 'F', 'NON-PARTITIONED', 'T', 'PARTITIONED') as 'PARTITIONED',
       idx.index_name as 'INDEX_NAME',
       'NON-PARTITIONED' as 'PARTITION_INDEX',
       tbs.name as 'TBS_NAME',
       round((tbs.total_page_count*tbs.page_size)/1024,1) as 'TBS_MAX(KB)',-- 테이블스페이스의 최대 크기
       round((tbs.extent_page_count * tbs.page_size * seg.total_extent_count)/1024, 1) as
'ALLOC(KB)', -- 테이블에서 할당받은 전체 크기
```

```
       round(seg.total_used_size/1024, 1) as 'USED(KB)', -- 테이블의 실 사용량(데이터 사용량)
       round((((tbs.extent_page_count * seg.total_extent_count)/tbs.total_page_count)*100), 1) as
'USAGE(%)' -- 테이블스페이스의 최대 크기를 기준으로 한 사용량
       --, round(((seg.total_used_size/(tbs.extent_page_count * tbs.page_size *
seg.total_extent_count))*100), 1) as 'USAGE(%)'     -- 테이블의 할당 크기를 기준으로 한 사용량
FROM  x$segment seg,
      v$index i,
      system_.sys_indices_ idx,
      system_.sys_tables_ tbl,
      v$tablespaces tbs,
      system_.sys_users_ u
WHERE seg.table_oid = i.table_oid
      and seg.segment_pid = i.index_seg_pid
      and i.index_id = idx.index_id
      and idx.table_id = tbl.table_id
       and seg.space_id = idx.tbs_id
       and seg.space_id = tbs.id
       and idx.user_id = u.user_id
       and tbl.table_type = 'T'
       and idx.is_partitioned = 'F'
UNION ALL
-- PARTITIONED INDEX
SELECT u.user_name as 'USER_NAME',
       tbl.table_name as 'TABLE_NAME',
       pt.partition_name as 'PARTITIONED',
       idx.index_name as 'INDEX_NAME',
       pi.index_partition_name as 'PARTITION_INDEX',
       tbs.name as 'TBS_NAME' ,
       round((tbs.total_page_count*tbs.page_size)/1024,1) as 'TBS_MAX(KB)',-- 테이블스페이스의 최대 크기
       round((tbs.extent_page_count * tbs.page_size * seg.total_extent_count)/1024, 1) as
'ALLOC(KB)', -- 테이블에서 할당받은 전체 크기
       round(seg.total_used_size/1024, 1) as 'USED(KB)', -- 테이블의 실 사용량(데이터 사용량)
       round((((tbs.extent_page_count * seg.total_extent_count)/tbs.total_page_count)*100), 1) as
'USAGE(%)' -- 테이블스페이스의 최대 크기를 기준으로 한 사용량
       --, round(((seg.total_used_size/(tbs.extent_page_count * tbs.page_size *
seg.total_extent_count))*100), 1) as 'usage(%)' -- 테이블의 할당 크기를 기준으로 한 사용량
FROM   x$segment seg,
       v$index i ,
       v$tablespaces tbs ,
       system_.sys_indices_ idx,
       system_.sys_table_partitions_ pt ,
       system_.sys_index_partitions_ pi ,
       system_.sys_tables_ tbl ,
       system_.sys_users_ u
WHERE  seg.table_oid = i.table_oid
       and seg.segment_pid = i.index_seg_pid
       and i.table_oid = pt.partition_oid
       and i.index_id = pi.index_id
       and seg.space_id = pi.tbs_id
       and pt.partition_id = pi.table_partition_id
       and pi.tbs_id = tbs.id
       and pi.table_id = tbl.table_id
       and pi.user_id = u.user_id
       and idx.index_id = pi.index_id
       and idx.is_partitioned = 'T'
       and tbl.table_type = 'T'
ORDER BY user_name, table_name, index_name, partitioned, partition_index
;
```

[OB06] 디스크 테이블 [back]

```
-- DISK NON-PARTITIONED TABLE DATA
SELECT u.user_name 'USER_NAME',
       tbl.table_name 'TABLE_NAME',
       'non-partitioned' 'PARTITIONED',
```

```
       tbs.name 'TBS_NAME',
       to_char((tbs.total_page_count * tbs.page_size)/1024 , '999,999,999,999') 'TBS_MAX(KB)',
       to_char((tbs.extent_page_count * tbs.page_size * seg.total_extent_count)/1024,
'999,999,999,999') 'ALLOC(KB)',
       to_char(seg.total_used_size/1024, '999,999,999,999') 'USED(KB)',
       round((((tbs.extent_page_count * tbs.page_size *
seg.total_extent_count)/(tbs.total_page_count * tbs.page_size))*100), 2) 'USAGE(%)'
FROM   x$segment seg,
       system_.sys_tables_ tbl,
       v$tablespaces tbs,
       system_.sys_users_ u
WHERE  seg.table_oid = tbl.table_oid
       and seg.space_id = tbl.tbs_id
       and seg.space_id = tbs.id
       and tbl.user_id = u.user_id
       and tbl.table_type = 'T'
       and tbl.is_partitioned = 'F'
       and seg.segment_type = 6
UNION ALL
-- DISK PARTITIONED TABLE DATA
SELECT u.user_name 'USER_NAME',
       tbl.table_name 'TABLE_NAME',
       pt.partition_name 'PARTITIONED',
       tbs.name 'TBS_NAME' ,
       to_char((tbs.total_page_count * tbs.page_size)/1024 , '999,999,999,999') 'TBS_MAX(KB)',
       to_char((tbs.extent_page_count * tbs.page_size * seg.total_extent_count)/1024,
'999,999,999,999') 'ALLOC(KB)',
       to_char(seg.total_used_size/1024, '999,999,999,999') 'USED(KB)',
       round((((tbs.extent_page_count * tbs.page_size *
seg.total_extent_count)/(tbs.total_page_count * tbs.page_size))*100), 2) 'USAGE(%)'
FROM   x$segment seg,
       v$tablespaces tbs,
       system_.sys_table_partitions_ pt,
       system_.sys_tables_ tbl,
       system_.sys_users_ u
WHERE  seg.table_oid = pt.partition_oid
       and seg.space_id = pt.tbs_id
       and pt.table_id = tbl.table_id
       and pt.tbs_id = tbs.id
       and pt.user_id = u.user_id
       and seg.segment_type = 6
       and tbl.table_type = 'T'
order by user_name, table_name, partitioned, tbs_name
;
```

[OB07] 파티션 테이블 정보 [back]

```
SELECT t.table_name, p.partition_name, p.partition_min_value, p.partition_max_value
FROM   system_.sys_table_partitions_ p, system_.sys_tables_ t
WHERE t.table_id = p.table_id
;
```

[OB08] 시퀀스 [back]

```
SELECT user_name,
      table_name seq_name,
      min_seq min,
      current_seq,
      max_seq max,
      increment_seq increment,
      is_cycle,
      cache_size cache
  FROM v$seq a,
      system_.sys_users_ b,
      system_.sys_tables_ c
```

```
 WHERE a.SEQ_OID = c.table_oid
      and b.user_id = c.user_id
      and b.user_name <> 'SYSTEM_'
;
```

[OB09] 시노님 [back]

```
select nvl(user_name, 'PUBLIC') synonym_owner,
      synonym_name,
      object_owner_name object_owner,
      object_name,
      to_char(a.last_ddl_time, 'YYYY-MM-DD HH:MI:SS') last_ddl_time
  from system_.SYS_SYNONYMS_ a left outer join system_.SYS_users_ b
      on a.synonym_owner_id = b.user_id
   and object_owner_name <> 'SYSTEM_'
;
```

■ **주요 컬럼 설명**

last_ddl_time	마지막으로 DDL이 수행된 시간.

[OB10] PSM(프로시저/펑션/타입세트) [back]

```
SELECT a.user_name,
      decode(object_type, 0, 'PROCEDURE', 1, 'FUNCTION', 3, 'TYPESET') psm_type,
      proc_name psm_name,
      decode(status, 0, 'VALID', 'INVALID') status,
      to_char(b.last_ddl_time, 'YYYY-MM-DD HH:MI:SS') last_ddl_time
  FROM system_.sys_users_ a,
      system_.sys_procedures_ b
 WHERE a.user_id = b.user_id
      and a.user_name <> 'SYSTEM_'
;
```

■ **주요 컬럼 설명**

status	컴파일 상태를 나타낸다. 1 이면 컴파일이 필요한 상태이다.

[OB11] PSM 생성구문 (PSM 이름을 입력) [back]

```
SELECT parse
  FROM system_.sys_proc_parse_
 WHERE proc_oid = (SELECT proc_oid
                    FROM system_.sys_procedures_
                   WHERE proc_name = 'MY_PROC')
 ORDER BY seq_no
;
```

[OB12] VIEW [back]

```
SELECT a.user_name,
      b.table_name view_name,
      decode(status, 0, 'VALID', 'INVALID') status,
      to_char(b.last_ddl_time, 'YYYY-MM-DD HH:MI:SS') last_ddl_time
  FROM system_.sys_users_ a,
      system_.sys_tables_ b,
      system_.sys_views_ c
 WHERE a.user_id = b.user_id
      and b.table_id = c.view_id
      and b.table_type = 'V'
;
```

[OB13] VIEW 생성구문 [back]

```
SELECT parse
  FROM system_.sys_view_parse_
 WHERE view_id = (SELECT table_id
                    FROM system_.sys_tables_
                   WHERE table_name = 'MY_VIEW')
 ORDER BY seq_no
;
```

[OB14] PACKAGE [back]

```
SELECT a.user_name,
       package_name,
       decode(package_type, 6, 'PACKAGE_SPEC', 7, 'PACKAGE_BODY') PACKAGE_TYPE,
       decode(status, 0, 'VALID', 'INVALID') STATUS,
       to_char(b.last_ddl_time, 'YYYY-MM-DD HH:MI:SS') LAST_DDL_TIME
FROM   system_.sys_users_ a,
       system_.sys_packages_ b
WHERE  a.user_id = b.user_id
       and a.user_name <> 'SYSTEM_'
;
```

■ **주요 컬럼 설명**

pakage_type	패키지 유형. 스펙인지 바디인지를 나타낸다.
status	컴파일 상태를 나타낸다. 1 이면 컴파일이 필요한 상태이다.

[OB15] PACKAGE 서브프로그램 정보 [back]

```
SELECT user_name,
       object_name,
       package_name,
       decode(sub_type, 0, 'PROCEDURE', 1, 'FUNCTION') SUB_TYPE,
       para_name,
       para_order,
       decode(inout_type, 0, 'IN', 1, 'OUT', 2, 'IN OUT') INOUT_TYPE,
       data_type,
       size,
       default_val
from   system_.sys_users_ a,
       system_.sys_package_paras_ b
WHERE  a.user_id = b.user_id
       and a.user_name <> 'SYSTEM_'
;
```

■ **주요 컬럼 설명**

sub_type	컴파일 상태를 나타낸다. 1 이면 컴파일이 필요한 상태이다.
para_name	서브프로그램 파라미터 이름
para_order	서브프로그램 파라미터 순서. 첫번째일 경우 1 을 가짐.
inout_type	파라미터의 입출력 여부
data_type	파라미터의 데이터 타입
size	파라미터 타입의 크기
default_val	파라미터의 기본값

[OB16] PACKAGE 생성구문 [back]

```
SELECT parse, decode(package_type, 6, 'PACKAGE_SPEC', 7, 'PACKAGE_BODY') PACKAGE_TYPE
FROM   system_.sys_package_parse_
WHERE  package_oid in (SELECT package_oid
                       FROM system_.sys_packages_
```

```
                        WHERE package_name = 'MY_PACKAGE')
ORDER BY seq_no
;
```

Privileges

Privileges 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[PV01] 사용자 시스템 권한 [back]

```
SELECT a.user_name grantee,
      c.user_name grantor,
      replace(d.priv_name, '_', ' ') priv_name
  FROM system_.sys_users_ a,
      system_.sys_grant_system_ b,
      system_.sys_users_ c,
      system_.sys_privileges_ d
 WHERE c.user_name <> 'SYSTEM_'
     and b.grantee_id = a.user_id
     and b.grantor_id = c.user_id
     and b.priv_id = d.priv_id
;
```

■ 주요 컬럼 설명

grantee	권한을 부여 받은 사용자의 이름
grantor	권한을 부여 해준 사용자의 이름
priv_name	권한의 이름

[PV02] 사용자 오브젝트 권한 [back]

```
SELECT a.user_name grantee,
      c.user_name grantor,
      f.user_name object_owner,
      e.table_name object_name,
      e.table_type object_type,
      replace(d.priv_name, '_', ' ') priv_name,
      decode(b.with_grant_option, 0, 'NO', 'YES') grantable
  FROM system_.sys_users_ a,
      system_.sys_grant_object_ b,
      system_.sys_users_ c,
      system_.sys_privileges_ d,
      system_.sys_tables_ e,
      system_.sys_users_ f
 WHERE c.user_name <> 'SYSTEM_'
      and b.grantee_id = a.user_id
      and b.grantor_id = c.user_id
      and b.priv_id = d.priv_id
      and b.obj_id = e.table_id
      and e.user_id = f.user_id
 ORDER BY grantee, grantor, object_owner, object_type, object_name, priv_name
;
```

■ 주요 컬럼 설명

object_owner	오브젝트를 소유하는 사용자의 이름
grantable	오브젝트 권한을 부여 받은 사용자가 다른 사용자에게도 해당 권한을 부여할 수 있는지를 의미한다. 1 이면 가능하고 0 이면 불가능하다.

Constraints

constraints 정보를 확인하기 위한 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[CT01] 전체 제약조건 목록 [back]

```
SELECT user_name,
      table_name object_name,
      decode(b.table_type, 'T', 'TABLE', 'Q', 'QUEUE', 'V', 'VIEW', 'SEQUENCE') object_type,
      c.constraint_name const_name,
      decode(c.constraint_type,0, 'FK', 1, 'NOT NULL', 2, 'UNIQUE', 3, 'PK', 4, 'NULL', 5,
'TIMESTAMP', 6, 'LOCAL UNIQUE') const_type,
      column_name
  FROM system_.sys_users_ a,
      system_.sys_tables_ b,
      system_.sys_constraints_ c,
      system_.sys_columns_ d,
      system_.sys_constraint_columns_ e
 WHERE a.user_id=c.user_id
      and b.table_id = c.table_id
      and a.user_id = d.user_id
      and a.user_id = e.user_id
      and b.table_id = d.table_id
      and b.table_id = e.table_id
      and c.constraint_id = e.constraint_id
      and d.column_id = e.column_id
      and a.user_name <> 'SYSTEM_'
;
```

■ 주요 컬럼 설명

object_type	오브젝트의 타입. / TABLE(T), VIEW(V), SEQUENCE(S), QUEUE(Q)
const_type	제약조건 타입. / FK(0),NOT NULL(1),UNIQUE(2),PK(3),NULL(4),TIMESTAMP(5),LOCAL UNIQUE(6)

[CT02] PK, FK, UNIQUE 관련 제약조건 및 테이블, 인덱스 목록 [back]

```
SELECT a.user_name,
      b.table_name,
      decode(c.constraint_type,0, 'FK', 2, 'UNIQUE', 3, 'PK', 4, 'NULL') const_type,
      c.constraint_name const_name,
      decode(d.index_name,c.constraint_name,NULL,index_name) index_name,
      (select table_name from system_.sys_tables_ where table_id = c.referenced_table_id) r_table,
      (select index_name from system_.sys_indices_ where index_id = c.referenced_index_id) r_index
  FROM system_.sys_users_ a,
      system_.sys_tables_ b,
      system_.sys_constraints_ c left outer join system_.sys_indices_ d on c.index_id = d.index_id
 WHERE c.table_id = b.table_id
      and a.user_name <> 'SYSTEM_'
      and c.user_id = a.user_id
      and c.constraint_type in (3, 0, 2, 6) --PK, FK, UNIQUE, LOCAL UNIQUE
 ORDER BY table_name, const_type
;
```

■ 주요 컬럼 설명

r_table	제약조건이 FK의 경우 참조대상 테이블의 이름.
r_index	제약조건이 FK의 경우 참조대상 인덱스의 이름.

[CT03] 복합 인덱스 컬럼 구성 목록 [back]

```
SELECT d.user_name,
      c.table_name,
```

```
      b.index_name,
      e.column_name,
      a.index_col_order col_order,
      decode(a.sort_order, 'A', 'ASC', 'D', 'DESC') sort
 FROM system_.sys_index_columns_ a,
      system_.sys_indices_ b,
      system_.sys_tables_ c,
      system_.sys_users_ d,
      system_.sys_columns_ e
WHERE d.user_name <> 'SYSTEM_'
      and c.table_type = 'T'
      and a.index_id = b.index_id
      and a.table_id = c.table_id
      and a.user_id = d.user_id
      and a.column_id = e.column_id
ORDER BY user_name, table_name, index_name, col_order
;
```

■ 주요 컬럼 설명

col_order	인덱스 컬럼의 구성 순서로 0 부터 시작한다.
sort	인덱스 컬럼의 정렬 형태. / ASC(A), DESC(D)

[CT04] 인덱스 정보 요약 [back]

```
SELECT a.user_name,
       c.index_name,
       c.index_id,
       b.table_name,
       nvl(d.name, 'SYS_TBS_MEMORY') tbs_name,
       c.is_unique,
       c.column_cnt
 FROM system_.sys_users_ a,
       system_.sys_tables_ b,
       system_.sys_indices_ c left outer join v$tablespaces d on c.tbs_id = d.id
WHERE a.user_name <> 'SYSTEM_'
       and b.table_type = 'T'
       and c.table_id = b.table_id
       and c.user_id = a.user_id
ORDER BY b.table_name, c.index_name
;
```

■ 주요 컬럼 설명

tbs_name	인덱스가 저장된 테이블스페이스의 이름, (A4 에서도 호환이 가능하도록 테이블스페이스가 없는 인덱스는 메모리 테이블스페이스로 간주)
is_unique	인덱스의 유니크 여부. / TRUE(T), FALSE(F)
column_cnt	인덱스를 구성하는 컬럼의 개수.

Replication

replication 정보를 확인하기 위한 각 모니터링 요소에 대응하는 쿼리는 아래와 같다.

[RP01] 이중화 sender 정보 [back]

```
SELECT rep_name,
      peer_ip remote_ip,
      peer_port remote_rep_port,
      decode(status, 0, 'STOP', 1, 'RUN', 2, 'RETRY') as staus,
      decode(net_error_flag, 0, 'OK', 'Error') as network,
```

```
     xsn
  FROM v$repsender
;
```

■ 주요 컬럼 설명

rep_name	이중화 객체의 이름.
remote_ip	이중화 대상인 원격서버의 ip 주소
remote_rep_port	이중화 대상인 원격서버의 이중화 포트번호로 원격서버의 프로퍼티에 의해 정해진다.
status	sender의 현재 상태로 1 이어야 정상이다. / STOP(0), RUN(1), RETRY(2)
repl_mode	sender의 현재 이중화 모드 / lazy, eager
network	network 에러 여부로 0 이어야 정상이다. / OK(0), ERROR(1)
XSN	sender가 마지막으로 송신한 SN(Serial Number/리두로그일련번호)으로 v$repgap의 REP_SN과 동일하다.

[RP02] 이중화 receiver 정보 [back]

```
SELECT rep_name, peer_ip remote_ip, peer_port remote_rep_port, apply_xsn FROM v$repreceiver
;
```

■ 주요 컬럼 설명

remote_ip	이중화 주체인 원격서버의 ip 주소.
remote_rep_port	이중화 주체인 원격서버의 sender가 접속한 포트번호로 특별히 정해지지 않았기에 그때그때 다르다.
apply_xsn	receiver가 현재 반영중인 원격서버의 SN.

[RP03] 이중화갭 [back]

```
SELECT rep_name, rep_sn, rep_last_sn, rep_gap, read_file_no, start_flag FROM v$repgap
;
```

■ 주요 컬럼 설명

rep_sn	sender가 마지막으로 송신한 SN, v$repsender의 XSN과 동일
rep_last_sn	지역서버의 트랜잭션에 의해 가장 최근에 로깅된 SN
rep_gap	rep_last_sn과 rep_sn의 간격으로 비동기화 정도를 나타낸다. (즉, rep_last_sn - rep_sn)
read_file_no	sender 가 이중화를 위해 마지막으로 접근한 리두로그 파일번호로써 메모리상의 이중화 전용 리두로그 버퍼를 읽을 때는 갱신되지 않으므로 이중화갭이 없는데도 과거의 리두로그 파일번호가 나타날 수 있다. 따라서, 이 컬럼의 값은 이중화갭이 있을 때만 의미가 있는 값이다.
start_flag	이중화의 구동상태로 일반적으로 0 이다. / NORMAL(0), QUICK(1), SYNC(2), SYNC_ONLY(3), SYNC RUN(4), SYNC END(5), RECOVERY(6), OFFLINE(7)

[RP04] 이중화 전체 현황 [back]

```
SELECT a.replication_name rep_name,
       d.host_ip remote_ip,
       nvl(to_char(e.rep_gap), '-') as rep_gap,
       a.xsn restart_xsn,
       decode(b.peer_port, NULL, 'OFF', 'ON') as sender,
       decode(c.peer_port, NULL, 'OFF', 'ON') as receiver
  FROM system_.sys_repl_hosts_ d ,
       system_.sys_replications_ a
       left outer join v$repsender b on a.replication_name = b.rep_name
       left outer join v$repreceiver c on a.replication_name = c.rep_name
       left outer join
       (select rep_name, max(rep_gap) rep_gap from v$repgap group by rep_name) e
       on a.replication_name = e.rep_name
 WHERE a.replication_name = d.replication_name
 ORDER BY rep_name
;
```

■ 주요 컬럼 설명

restart_xsn	이중화 대상인 원격서버가 반영한 SN, 이중화 재 시작 시 재전송 기점을 의미

sender	sender의 작동 유무.
receiver	receiver의 작동 유무.

[RP05] 이중화를 수행하지 못해 누적된 리두로그 파일 측정 [back]

```
SELECT case2((buffer_min_sn < read_sn), 'REP BUFFER '||round((buffer_max_sn-read_sn)/(buffer_max_sn-buffer_min_sn)*100,2)||' % left ', (select to_char(cur_write_lf_no - read_file_no) FROM v$lfg, v$repgap)) logfile_for_rep
  FROM v$replogbuffer
;
```

■ 주요 컬럼 설명

logfile_for_rep	sender가 이중화 전용 리두로그 버퍼를 읽는 중에는 "이중화 전용 리두로그 버퍼에서 전송해야 할 로그"를 백분율로 나타내며 sender가 직접 리두로그 파일을 읽을 때는 "이중화를 해야 할 리두로그 파일의 개수"를 나타내므로 응용에 따라 간접적으로 MB 환산이 가능하다. 이 값이 백분율로만 나타난다면 이중화 전용 리두로그 버퍼에서만 이중화가 이루어 지는 것이므로 양호한 것이라 판단할 수 있다. (매우 빠르게 전송 시 순간적으로 음수가 나올 수도 있으나 이 경우는 무시하여도 된다.) 하지만, 리두로그 파일 개수로 빈번히 모니터링 될 때에는 프로퍼티 REPLICATION_LOG_BUFFER_SIZE 에 의해 정의되는 이중화 전용 리두로그 버퍼의 크기를 기본값인 30M 이상으로 늘리거나 sender를 위해 미리 메모리에 캐시(cache)할 리두로그 파일의 개수를 의미하는 프로퍼티 REPLICATION_PREFETCH_LOGFILE_COUNT을 늘려야 한다.

[RP06] 이중화 대상 테이블 목록 [back]

```
SELECT replication_name rep_name,
       local_user_name||'.'||local_table_name LOCAL_TBL,
       remote_user_name||'.'||remote_table_name REMOTE_TBL
  FROM system_.sys_repl_items_
 ORDER BY 1, 2
;
```

■ 주요 컬럼 설명

local_table	지역서버의 이중화 대상 테이블로 "사용자이름.테이블이름" 형식으로 출력된다.
remote_table	원격서버의 이중화 대상 테이블로 "사용자이름.테이블이름" 형식으로 출력된다.

알티베이스㈜
서울특별시 구로구 구로 3 동 182-13
대륭포스트 2 차 1008 호
02-2082-1000
http://www.altibase.com

대전사무소
대전광역시 서구 둔산동 921
주은리더스텔 901 호
042-489-0330

기술지원본부
서울특별시 구로구 구로 3 동 182-13
대륭포스트 2 차 908 호
02-2082-1000

기술지원센터
02-2082-1114
http://support.altibase.com