

- Administrator's Manual
 - 서문
 - 이 매뉴얼에 대하여
 - 1.Altibase 소개
 - Hybrid DBMS개념
 - Altibase 특징
 - Altibase 구조
 - 2.Altibase 구성요소
 - Altibase 디렉토리
 - 실행 바이너리
 - Altibase 라이브러리
 - 3.데이터베이스 생성
 - 데이터베이스 생성
 - 4.Altibase 구동 및 종료
 - Altibase 구동
 - Altibase 종료
 - 5.데이터베이스 객체 및 권한
 - 데이터베이스 객체 개요
 - 테이블
 - 임시 테이블
 - 압축 테이블
 - 큐
 - 제약조건
 - 인덱스
 - 뷰
 - Materialized View
 - 시퀀스
 - 시노님
 - 저장 프로시저 및 저장 함수
 - 트리거
 - 작업(Job)
 - 데이터베이스 사용자
 - 권한과 롤
 - 6.테이블스페이스
 - 테이블스페이스 정의 및 구조
 - 테이블스페이스 분류
 - 디스크 테이블스페이스
 - 언두 테이블스페이스

- 테이블스페이스 상태
- 테이블스페이스 관리
- 테이블스페이스 사용 예제
- 테이블스페이스 공간 관리

Altibase® Administration

Administrator's Manual



Altibase Administration Administrator's Manual

Release 7.1

Copyright © 2001~2018 Altibase Corp. All Rights Reserved.

본 문서의 저작권은 (주)알티베이스에 있습니다. 이 문서에 대하여 당사의 동의 없이 무단으로 복제 또는 전용할 수 없습니다.

(주)알티베이스

08378 서울시 구로구 디지털로 306 대륭포스트타워II 10층

전화: 02-2082-1114 팩스: 02-2082-1099

고객서비스포털: <http://support.altibase.com>

homepage: <http://www.altibase.com>

서문

이 매뉴얼에 대하여

이 매뉴얼은 Altibase를 구성, 관리, 사용하기 위해 필요한 개념에 대해 설명한다.

대상 사용자

이 매뉴얼은 다음과 같은 Altibase 사용자를 대상으로 작성되었다.

- 데이터베이스 관리자
- 시스템 관리자

- 성능 관리자

다음과 같은 배경 지식을 가지고 이 매뉴얼을 읽는 것이 좋다.

- 컴퓨터, 운영 체제 및 운영 체제 유틸리티 운용에 필요한 기본 지식
- 관계형 데이터베이스 사용 경험 또는 데이터베이스 개념에 대한 이해
- 데이터베이스 서버 관리, 운영 체제 관리 또는 네트워크 관리 경험

소프트웨어 환경

이 매뉴얼은 데이터베이스 서버로 Altibase 버전 7.1을 사용한다는 가정 하에 작성되었다.

이 매뉴얼의 구성

이 매뉴얼은 다음과 같이 구성되어 있다.

- 제 1장 Altibase 소개
이 장은 Altibase 서버를 이해하는데 필요한 개념, 특징 및 구조에 대한 개요를 제공한다.
- 제 2장 Altibase 구성 요소
이 장은 Altibase를 구성하고 있는 실행 바이너리 부문과 프로그래밍 라이브러리 부문 구성 요소들에 대해 설명한다.
- 제 3장 데이터베이스 생성
이 장은 데이터베이스를 구성하는 대표적 구성 요소인 테이블스페이스와 로깅 시스템의 종류 및 데이터베이스 생성 방법에 관하여 설명한다.
- 제 4장 Altibase 구동 및 종료
이 장은 Altibase를 구동 및 종료 시키는 방법과 Altibase 다단계 구동 시 내부적으로 수행하는 작업에 대해 설명한다.
- 제 5장 데이터베이스 객체 및 권한
이 장은 특정 사용자에 의해 생성된 제약조건, 인덱스, 시퀀스, 이중화, 테이블, 사용자 등 데이터베이스 객체들에 대해 설명한다. 또한, 시스템 및 스키마 객체 수준의 권한에 대해 설명한다.
- 제 6장 테이블스페이스 관리
이 장은 데이터베이스의 논리적 구조를 이해함으로써 데이터베이스의 공간 관리를 작은 단위로 제어하고, 물리적 데이터 영역을 효율적으로 관리하는 방법에 대해 설명한다.
- 제 7장 파티션드 객체
이 장은 대용량 데이터베이스 테이블을 여러 개의 작은 조각으로 분할하여 관리하는 파티션드 테이블에 대해 설명한다.

- 제 8장 트랜잭션 관리

이 장은 트랜잭션을 정의하고 트랜잭션을 사용하여 작업을 관리하는 방법에 대해 설명한다.

- 제 9장 버퍼 관리자

Altibase는 대용량의 데이터가 디스크에 저장될 수 있도록 지원하는데, 메모리의 공간은 한정되어 있으므로 이를 전부 메모리에 적재할 수 없으므로 테이블, 인덱스, 레코드 등 모든 데이터에 접근하기 위해서는 먼저 디스크의 데이터를 메모리에 적재해야 한다. 이 장은 이러한 메모리 공간을 할당하고, 메모리 공간이 부족한 경우 필요한 데이터를 제공할 수 있도록 메모리 공간을 유지 및 관리하는 버퍼 관리자에 대하여 설명한다.

- 제 10장 백업 및 복구

이 장은 시스템 정전 또는 디스크, 데이터 파일 손상 유실 등과 같은 예기치 않은 상황으로 인해 Altibase에 저장된 데이터가 손실될 경우를 대비하여 Altibase에서 지원하는 백업 및 복구에 대하여 설명한다.

- 제 11장 증분 백업과 복구

이 장은 Altibase가 제공하는 증분 백업과 증분 백업을 이용한 복구에 대하여 설명한다.

- 제 12장 서버/클라이언트 통신

이 장은 Altibase 서버와 클라이언트 응용프로그램간의 접속 방법과 프로토콜에 대해 설명한다.

- 제 13장 Altibase의 보안

이 장은 데이터베이스의 정보를 보호하기 위한 Altibase의 보안 기능에 대해 설명한다.

- 제 14장 Altibase 감사

이 장은 Altibase 서버 내에서 실행되고 있는 구문을 실시간으로 추적하고 로그를 기록하는 감사(Auditing) 기능에 대해 설명한다.

- 제 15장 Altibase 튜닝

이 장은 Altibase의 성능 향상을 위한 로그 파일 그룹과 그룹 커밋에 대해 설명한다.

- 제 16장 Altibase 모니터링 및 PBT

이 장은 Altibase의 운영상태를 확인하는 방법과 해당 내용을 분석하는 방법에 대해 설명한다. 또한, Altibase 운영 중 발생할 수 있는 여러 가지 문제 상황에 대하여 점검 사항 및 분석 방법에 대해 설명한다.

- A. 부록: Trace Log

이 부록은 Altibase 서버에서 실행되는 SQL문 관련 정보를 trace 로그로 남기는 방법을 설명한다.

- B. 부록: Altibase 최대치

이 부록은 Altibase 객체들의 최대값을 기술한다.





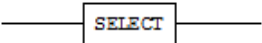
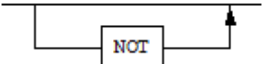
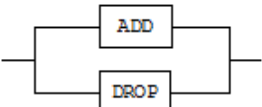
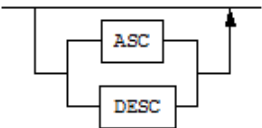
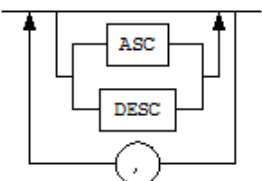
문서화 규칙

이 절에서는 이 매뉴얼에서 사용하는 규칙에 대해 설명한다. 이 규칙을 이해하면 이 매뉴얼과 설명서 세트의 다른 매뉴얼에서 정보를 쉽게 찾을 수 있다. 여기서 설명하는 규칙은 다음과 같다.

- 구문 다이어그램
- 샘플 코드 규칙

구문 다이어그램

이 매뉴얼에서는 다음 구성 요소로 구축된 다이어그램을 사용하여, 명령문의 구문을 설명한다.

구성 요소	의미
	명령문이 시작한다. 완전한 명령문이 아닌 구문 요소는 화살표로 시작한다.
	명령문이 다음 라인에 계속된다. 완전한 명령문이 아닌 구문 요소는 이 기호로 종료한다.
	명령문이 이전 라인으로부터 계속된다. 완전한 명령문이 아닌 구문 요소는 이 기호로 시작한다.
	명령문이 종료한다.
	필수 항목
	선택적 항목
	선택사항이 있는 필수 항목. 한 항목만 제공해야 한다.
	선택사항이 있는 선택적 항목
	선택적 항목. 여러 항목이 허용된다. 각 반복 앞부분에 콤마가 와야 한다.

샘플 코드 규칙

코드 예제는 SQL, Stored Procedure, iSQL, 또는 다른 명령 라인 구문들을 예를 들어 설명한다.

아래 테이블은 코드 예제에서 사용된 인쇄 규칙에 대해 설명한다.

규칙	의미	예제
[]	선택 항목을 표시	VARCHAR [(size)][[FIXED] VARIABLE]
{ }	필수 항목 표시. 반드시 하나 이상을 선택해야 되는 표시	{ ENABLE DISABLE COMPILE }
	선택 또는 필수 항목 표시의 인자 구분 표시	{ ENABLE DISABLE COMPILE } [ENABLE DISABLE COMPILE]
...	그 이전 인자의 반복 표시 예제 코드들의 생략되는 것을 표시	SQL> SELECT ename FROM employee; ENAME ----- SWNO HJNO HSCHOI . . . 20 rows selected.
그 밖에 기호	위에서 보여진 기호 이 외에 기호들	EXEC :p1 := 1; acc NUMBER(11,2);
기울임 꼴	구문 요소에서 사용자가 지정해야 하는 변수, 특수한 값을 제공해야만 하는 위치	SELECT * FROM <i>table_name</i> ; CONNECT <i>userID/password</i> ;
소문자	사용자가 제공하는 프로그램의 요소들, 예를 들어 테이블 이름, 칼럼 이름, 파일 이름 등	SELECT ename FROM employee;
대문자	시스템에서 제공하는 요소들 또는 구문에 나타나는 키워드	DESC SYSTEM_.SYS_INDICES_;

관련 자료

자세한 정보를 위하여 다음 문서 목록을 참조하기 바란다.

- Installation Guide
- Getting Started Guide
- SQL Reference
- Stored Procedures Manual
- iSQL User's Manual
- Utilities Manual
- Error Message Reference

Altibase는 여러분의 의견을 환영합니다.

이 매뉴얼에 대한 여러분의 의견을 보내주시기 바랍니다. 사용자의 의견은 다음 버전의 매뉴얼을 작성하는데 많은 도움이 됩니다. 보내실 때에는 아래 내용과 함께 고객센터포털(<http://support.altibase.com/kr/>)로 보내주시기 바랍니다.

- 사용 중인 매뉴얼의 이름과 버전
- 매뉴얼에 대한 의견
- 사용자의 성함, 주소, 전화번호

이 외에도 Altibase 기술지원 설명서의 오류와 누락된 부분 및 기타 기술적인 문제들에 대해서 이 주소로 보내주시면 정성껏 처리하겠습니다. 또한, 기술적인 부분과 관련하여 즉각적인 도움이 필요한 경우에도 고객센터포털을 통해 서비스를 요청하시기 바랍니다.

여러분의 의견에 항상 감사드립니다.

1.Altibase 소개

이 장에서는 Altibase를 처음 접하는 사용자들을 위해서 Hybrid DBMS의 등장 배경과 Altibase의 구조 및 특징에 대해서 설명한다.

Hybrid DBMS개념

이 절은 Altibase에서 선도하고 있는 새로운 개념인 Hybrid Database Management System(이하 Hybrid DBMS)에 대해서 설명한다.

Hybrid DBMS 등장 배경

Hybrid DBMS의 등장은 데이터를 저장하는 두 가지 대표적인 기억 장치인 메모리와 디스크의 특징과 밀접한 관련이 있다.

첫째, 메모리는 전자 게이트로 구성되어 있고 접근시간이 수 ns(십억 분의 일초) 단위로 매우 빠르며 접근시간이 균일하고 정전 시에는 데이터가 소실되는 특징을 가지고 있다. 즉, 메모리는 휘발성 저장 매체이다.

반면에, 디스크는 헤드암과 플래터로 구성되어 있다. 디스크 접근시간이 수 us(백만 분의 일초)로 메모리에 비해서 상대적으로 느리다. 게다가 접근시간이 균일하지 않다. 최근에 좀 더 대중화된 SSD(Solid State Drives)조차도 휘발성 메모리에 비해서는 접근 시간이 좀 느리다. 그러나 정전이 되더라도 디스크의 데이터는 영구히 보존되는 특징을 가지고 있다.

둘째, 메모리는 메인보드와 시스템 버스로 연결되어 있어 메인보드의 특성에 따라 최대 저장 크기가 결정 된다. 메인보드에 장착된 CPU가 32비트이면 메모리의 최대 크기는 4GB, CPU가 64비트라고 해도 최대 크기는 현재 수백 GB(십억 바이트)까지만 장착 가능하다. 반면에, 디스크는 메인보드와 I/O버스로 연결되어 있어 메인보드의 특징과 거의 무관하게 수 TB(일조 바이트)까지 구성이 가능하다.

요약하면, 일반적으로 메모리는 디스크에 비해 접근 시간이 수백배 빠르며 성능이 균일한 반면, 정전시 데이터가 소실되고 저장용량에 한계가 있다. 이에 반해, 디스크는 데이터가 영구히 저장되며 저장용량의 한계가 거의 없는 반면, 접근 시간이 느리고 일정하지 않다.

이런 기억장치의 특징에 따라서 DBMS는 디스크에 데이터를 저장하는 Disk-Resident DBMS(이하 DRDBMS)와 메모리에 데이터를 저장하는 Main-Memory DBMS(이하 MMDBMS)가 존재해 왔다. Hybrid DBMS는 이 두 가지 구조의 장점을 수용하고, 단점을 보완하여 개발 되었다.

DRDBMS 등장

DRDBMS구조는 데이터가 디스크에 저장되어 있고, DRDBMS가 디스크에 있는 데이터를 메모리 버퍼로 읽어서 응용프로그램에게 전달해 주는 형태이다.

이런 구조는 응용프로그램이 표준 SQL을 통해서 데이터에 접근하고, DBMS가 동시성제어 및 복구를 통해 데이터를 보호하기 때문에 응용프로그램의 개발이 훨씬 간편해지며 데이터 공유가 쉬운 장점이 있다. 또한 데이터가 디스크에 저장되어 있기 때문에 대용량 데이터베이스를 구성할 수 있다.

이런 장점 때문에, 지금까지 전 산업분야에서 DRDBMS가 광범위하게 사용되어 왔다.

하지만, 사회전반에 걸쳐 정보화가 급격히 진전되고 정보처리의 요구 성능이 높아지면서 데이터 처리에 대한 수요는 많았으나, DRDBMS의 낮은 평균 처리속도와 처리속도의 기복(jitter)의 문제 때문에 DRDBMS를 사용하지 못하는 분야가 많았다.

따라서 고성능 및 균일 성능의 데이터 처리를 필요로 하는 산업 분야에서는 지금까지 Custom Designed Memory DB를 사용하였다. 그러나 범용적이지 않고 데이터 관리에 필요한 모든 것을 직접 개발해야 했기 때문에 개발 및 유지보수가 어렵고, 성능, 가용성, 확장성 등에서 문제가 있었다.

MMDBMS 등장

MMDBMS의 구조는 데이터가 메모리에 저장되어 있고 MMDBMS가 메모리에 있는 데이터를 읽어서 응용프로그램에 전달하는 형태이다.

이런 구조는 DRDBMS의 장점인 표준 SQL을 통한 데이터 접근, 동시성 제어 및 복구를 통한 데이터 보호, 이를 통한 응용프로그램의 개발과 데이터 공유의 용이성 등의 장점을 그대로 가진다.

또한 MMDBMS는 데이터를 메모리에 저장하기 때문에 디스크에 데이터를 저장하는 DRDBMS에 비해 평균처리 속도가 매우 빠르며 메모리의 특징인 균일한 성능을 보장한다. 따라서 고성능 및 균일 성능의 필요성 때문에 DRDBMS를 사용할 수 없는 분야에서 각광을 받고 있다.

일반적으로 DRDBMS에 비해서 MMDBMS가 갱신 연산은 약 10배, 검색 연산은 약 3배 이상의 성능을 보인다.

갱신 연산이 DRDBMS에 비해서 수백배가 아닌 이유는 MMDBMS도 데이터 보호를 위해서 DRDBMS와 똑같이 로그파일을 디스크에 기록하기 때문이다. 그럼에도 불구하고 MMDBMS의 갱신 연산이 더 빠른 이유는 MMDBMS는 데이터 보호가 DRDBMS에 비해서 훨씬 단순하게 최적화되어 있기 때문이다.

또한 검색 연산이 DRDBMS에 비해서 수백배가 아닌 이유는 DRDBMS도 데이터접근 성능을 높이기 위해 메모리 버퍼를 사용하기 때문이다. 그럼에도 불구하고 MMDBMS의 검색 연산이 더 빠른 이유는 데이터접근이 단순하게 최적화되어 있고, 메모리 접근 시 성능의 기복(jitter)이 없기 때문이다.

이런 고성능 및 균일성능의 장점에도 불구하고 데이터를 메모리에 저장해야 하는 MMDBMS는 정보처리의 요구량이 방대하여 수백 GB이상의 데이터를 저장해야 하는 산업분야에서 다시 한계를 나타내게 되었다.

MMDBMS와 DRDBMS 혼용 구조 등장

이런 문제점을 극복하기 위해서 현재 가장 일반적으로 사용되는 구조는 데이터를 구분하여 저장하는 형태이다. 고성능이 필요한 데이터는 MMDBMS에, 대용량이 필요한 데이터는 DRDBMS에 저장함으로써 MMDBMS와 DRDBMS를 혼용한다.

이러한 구조는 MMDBMS와 DRDBMS의 공통정보에 대해서 서로 동기화해야 한다는 문제, MMDBMS와 DRDBMS를 같이 처리해야 하는 응용프로그램은 양쪽에 동시에 접속해서 처리해야 한다는 문제, 또한 장애 복구가 복잡하다는 문제들을 가지고 있다.

하지만 이제까지는 고성능처리와 대용량처리를 함께 처리할 방법이 없었기 때문에, 고성능처리와 대용량 정보처리가 필요한 분야에서는 일반적으로 활용되고 있다.

Hybrid DBMS 등장

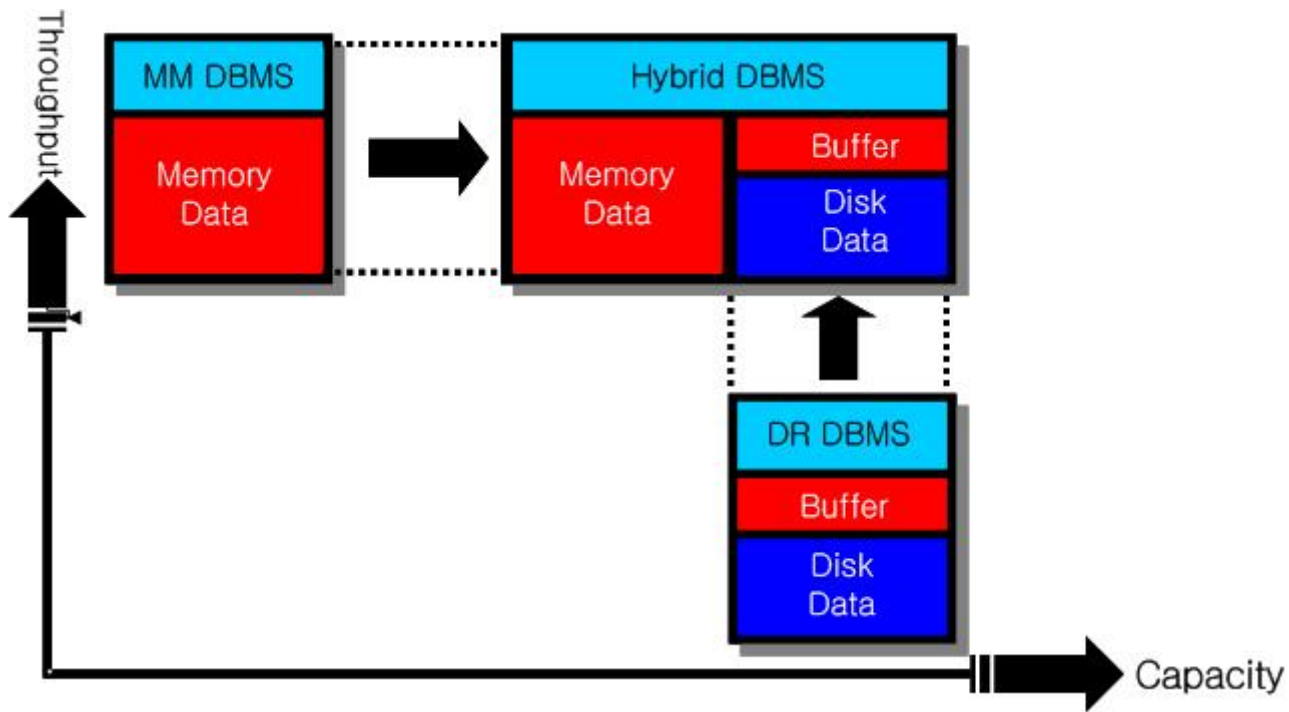
Hybrid DBMS는 DRDBMS의 구조, MMDBMS의 구조, 그리고 MMDBMS와 DRDBMS가 결합된 구조들의 장점을 수용하고 문제점을 해결하기 위해서 등장하게 되었다.

Hybrid DBMS는 데이터를 구분하여, 고성능이 필요한 데이터는 메모리에, 대용량이 필요한 데이터는 디스크에 저장하되, 이 두 가지 데이터를 처리하는 DBMS는 하나로 통합된 구조를 가지고 있다.

고성능 정보처리와 대용량 정보처리를 하나의 DBMS에서 통합, 처리하는 구조이기 때문에, 앞서 설명한 혼용구조의 동기화 문제, 복잡한 장애처리 문제, 응용프로그램이 복잡해지는 문제를 해결하게 되었다. 또한 MMDBMS 전용, DRDBMS 전용, Hybrid DBMS 등 다양한 구성이 가능하게 되었다.

요약하면, Hybrid DBMS는 고성능 정보처리를 가능하게 하는 MMDBMS와 대용량 정보처리를 가능하게 하는 DRDBMS의 장점을 결합하여 데이터는 구분하여 저장하고 데이터 관리는 통합한 구조이다.

즉, Hybrid DBMS는 효율적인 시간 활용으로 고성능 정보 처리가 가능하고, 효율적인 자원활용으로 대용량 정보처리를 할 수 있게 되었다. Hybrid DBMS는 고성능 및 대용량 정보처리가 모두 필요한 분야를 포함하여 포괄적으로 사용이 가능해졌다.



[그림 1-1] 고성능 대용량 DBMS구조

Altibase 특징

고성능 대용량 하이브리드 데이터베이스 시스템인 Altibase에 대한 일반적인 내용을 소개한다. 하이브리드 데이터베이스 시스템으로써 Altibase가 갖는 특징, 구조, 기능 등에 관하여 간략하게 설명한다. 이에 대한 자세한 내용은 Altibase 각각의 세부 매뉴얼을 참조한다.

데이터 모델

Altibase의 데이터 모델은 관계 모델 (relational model)을 채택하고 있다. 관계 모델은 세 가지 주요 요소를 포함한다.

구조(structure)는 데이터베이스를 저장하거나 접근하는 객체 단위로 테이블, 뷰, 인덱스 등을 일컫는다. 이들은 곧 연산자의 조작 단위가 된다.

연산(operation)은 데이터베이스의 데이터와 구조를 사용자들이 조작할 수 있도록 허용하는 행위(action)들을 정의한 것으로써 무결성 규칙을 수반한다.

무결성 규칙(integrity rule)은 데이터와 구조에 허용된 연산을 다루기 위한 법칙으로, 데이터와 구조를 보호하기 위한 것이다.

관계 데이터베이스 관리 시스템은 다음과 같은 장점을 제공한다.

- 물리적 데이터와 논리적 데이터 독립성을 유지한다.
- 모든 데이터에 대한 접근이 다양하고 쉽다.

- 데이터베이스 설계가 유연하다.
- 데이터베이스 저장 공간과 데이터의 중복을 줄일 수 있다.

엔진 구조

Altibase는 클라이언트-서버 구조를 제공한다. 클라이언트-서버 구조는 클라이언트가 통신망을 통해 서버에 접근하는 형태로 기존의 RDBMS가 제공하는 방식이다.

Altibase 서버는 내부적으로 다중 스레드 구조를 갖고 있다.

인터페이스

Altibase는 기존의 실시간 데이터베이스 시스템과는 달리 범용성을 추구하는 일환으로 산업 표준 인터페이스를 지원한다. Altibase에서 제공하는 데이터베이스 질의어는 SQL92와 SQL99 표준을 따른다.

프로그래밍 인터페이스로는 CLI, ODBC, JDBC, C/C++ Precompiler 등을 지원하고 있으며, 기존에 작성된 데이터베이스 응용 프로그램을 변환할 필요 없이 그대로 사용할 수 있다. Altibase가 지원하는 SQL에 대한 자세한 내용은 *SQL Reference* 를 참조한다.

다중버전 기법

Altibase는 다중버전 기법(이하 MVCC: Multi-Version Concurrency Control)을 이용한 동시성 제어를 수행한다. 다중 버전 기법은 하나의 데이터에 대해 여러 개의 버전을 유지하여 읽기와 쓰기 연산에 대한 충돌을 없앴으로서 최대의 성능을 발휘할 수 있도록 하는 것이다. 특히, 기존의 row locking 방식의 단점이었던 읽으려고 하는 데이터가 이미 다른 수정 연산에 의해 lock이 걸려 있거나, 수정하려는 데이터를 다른 읽기 연산이 읽는 중이어서 장기간 대기해야 하는 문제점을 제거하였으며, 필요 없는 오래된 데이터를 즉시 회수함으로써 메모리의 낭비를 방지하였다. 다중 버전 기법은 대규모 사용자가 접근하는 환경에서 최적의 성능을 발휘하고, 데이터베이스를 종료하지 않고도 즉시 백업할 수 있는 핫-백업(hot-backup) 시스템을 지원한다.

Altibase는 메모리 테이블과 디스크 테이블에 대해 외형상으로는 같은 기능을 하지만, 서로 다른 방법으로 MVCC를 구현하였다. 메모리 테이블은 레코드의 변경시마다 새로운 버전을 생성하는 out-place MVCC로 구현되어 있으며, 디스크 테이블의 경우는 변경된 데이터를 기존의 레코드에 덮어 쓰고, 변경 이전의 정보를 undo 테이블스페이스에 저장하여 참조하는 in-place MVCC 방식을 채택하고 있다.

트랜잭션

Altibase는 Hybrid DBMS의 구조에 맞추어 최고의 성능을 낼 수 있는 트랜잭션 구조와 이와 관련된 다양한 기능을 제공한다. 먼저 데이터베이스 내에서 동시에 수행될 수 있는 트랜잭션의 개수를 프로퍼티를 이용해서 조절할 수 있다. 또한 효율적인 서버 운영을 위해 AUTOCOMMIT 모드를 사용할 수 있다. 그리고 Altibase가 제공하는 트랜잭션의 고립화 수준(isolation level)은 read committed(=0), repeatable read(=1), no phantom read(=2)가 있으며, 사용자의 필요에 맞추어 적절하게 선택하여 사용할 수 있다.

로깅

데이터베이스 안정성과 영속성을 위하여 Altibase는 변경된 데이터베이스 내용에 대하여 로깅(logging)을 수행한다. 또한 시스템간의 이중화 (Replication) 작업 때의 성능을 극대화 시키기 위해 최적의 로그를 생성한다.

버퍼 풀 (Buffer Pool)

디스크 테이블스페이스에 접근하는 트랜잭션들의 성능을 향상시키기 위해 디스크에 대한 I/O 회수를 최소화해야 한다. 이를 위해 버퍼 풀을 사용함으로써 이전에 디스크로부터 읽은 페이지들 중 일정 부분을 메모리에 캐시해 두어 디스크에서 다시 읽어들이는 것을 방지한다. 버퍼 풀은 Hot-Cold LRU(Least Recently Used) 알고리즘에 의해 관리된다.

더블 라이트 (Double Write) 파일

Altibase 시스템의 페이지 크기와 파일 시스템의 물리적 페이지 크기가 다를 경우, 디스크 I/O 수행 중에 Altibase 서버가 비정상적으로 종료하면 페이지가 온전하지 못한 상태로 남아있을 수 있다.

이런 현상을 방지하기 위하여 Altibase는 페이지를 플러시할 때, 같은 이미지를 디스크의 더블 라이트 파일에 저장한 후, 페이지 원래 위치에 다시 저장한다. 그리고 Altibase를 재구동할 때 더블 라이트 파일의 내용과 실제 페이지의 내용을 비교하여 손상된 페이지를 복구한다.

더블 라이트 기능은 디스크의 결함을 보완해 주지만, 시스템의 성능을 떨어뜨릴 수 있다. 이 기능은 사용자가 성능을 위해 사용하지 않을 수 있다.

퍼지&핑퐁 체크포인트

Altibase는 최근의 데이터베이스 상태를 안전하게 백업(backup) 데이터베이스로 반영하기 위해 퍼지&핑퐁 체크포인트를 수행한다.

메인 메모리 데이터베이스에서의 퍼지 체크포인트(fuzzy checkpoint)는 모든 변경된 데이터 페이지가 백업 데이터베이스로 내려가 현재 수행중인 트랜잭션의 수행에 영향을 미칠 수 있기 때문에 퍼지 체크포인트와 더불어 핑퐁 체크포인트(ping pong checkpoint) 방식을 함께 수행한다. 즉, 백업 데이터베이스를 두 개로 관리함으로써, 체크포인트 과정에서 부하를 줄일 수 있어 트랜잭션 동작이 최대한의 성능을 발휘할 수 있다.

저장 프로시저 (Stored Procedure)

저장 프로시저는 입력 인자, 출력 인자, 입출력 인자를 가지고 바디(body) 내에 정의된 조건에 따라 여러 SQL 문을 한번에 수행하는 데이터베이스 프로시저다.

저장 프로시저의 종류는 리턴값 유무에 따라 프로시저와 함수로 나누어 진다. 자세한 내용은 *Stored Procedures Manual*을 참고한다.

데드락 감지 (Deadlock Detection)

데드락은 트랜잭션간의 리소스 할당이 자동으로 해제될 수 없는 비정상적인 트랜잭션 정지 상태이다. 이러한 경우 일반적으로 데드락을 감지하는 별도의 스레드 또는 프로세스를 두게 되는데, 이러한 감지 구조는 필연적으로 일시적인 서비스 중단 사태를 초래한다. Altibase는 별도의 데드락 감지 스레드를 두지 않고, 데드락이 발생하는 순간 데드락 상황을 감지하여, 신속히 조치를 취함으로써 어떠한 경우에도 서비스가 중지되지 않도록 하며, 지속적이고 안정적인 데이터베이스 운용을 보장한다.

테이블 컴팩션

데이터베이스 운용시, 실제로 특정 메모리 테이블이 필요한 메모리 공간 이상을 차지하는 경우가 발생한다. 주로 대량의 데이터가 삽입 된 후 변경 및 삭제가 이루어지는 경우가 그러하다. 이런 경우 해당 테이블에서 필요 없는 메모리를 시스템으로 반환할 수 있다면, 보다 효율적으로 메모리 사용이 가능하다. 이런 필요로 인해 Altibase는 메모리 테이블에 대해 테이블 단위의 컴팩션(compaction) 기능을 제공하며, 이 기능을 이용하여 메모리 및 테이블의 효율적 관리가 가능하다.

데이터베이스 이중화

Altibase는 시스템의 높은 가용성(high availability)과 무정지(fault tolerance) 시스템을 위하여 로그 기반의 데이터베이스 이중화(replication)를 제공한다. 로그 기반의 이중화 시스템 구조는 트랜잭션 로그를 기반으로 데이터베이스를 이중화시킴으로써, Altibase의 효율성을 높이고 시스템 부하를 줄일 수 있다. 서비스 중인 지역(local) 시스템의 이중화 관리 스레드는 로그 데이터를 원격(Remote) 시스템의 이중화 관리 스레드에 실시간으로 전달한다. 원격 시스템의 이중화 관리 스레드는 전달받은 로그 데이터를 분석하여 이것을 Altibase 서버에게 전달하고

Altibase 서버는 이 내용을 데이터베이스에 반영한다. 이렇게 함으로써 서비스 중인 컴퓨팅 시스템이 중단되었을 때, 시스템 복구 시간이 필요 없이 곧바로 다른 시스템을 사용하여 서비스할 수 있는 체제를 갖추고 있다.

Altibase는 부하 분산(load balancing) 기능도 제공한다. Altibase의 데이터베이스 복제 운영 환경에서, 서비스하는 트랜잭션들을 두 그룹 이상으로 나누어 각각의 트랜잭션이 해당 서버에서 수행되도록 구성하면, 각 서버에서 변경되는 데이터베이스 내용은 이중화를 통해서 상대방 서버에 반영됨으로써 복제된 데이터베이스의 일관성(consistency)을 보장할 수 있다.

Altibase Sharding

Altibase Sharding은 Altibase에 샤딩 기술을 도입함으로써 저장 용량과 시간당 처리량을 향상시켜 대용량의 데이터베이스를 분산 처리할 수 있도록 한 기능이다.

Altibase Sharding은 클라이언트측 샤딩과 서버측 샤딩을 모두 사용할 수 있다. 특히 클라이언트측 샤딩은 기존의 응용프로그램 또는 SQL을 수정하지 않고, 샤드 전용 라이브러리만 교체하는 것으로 적용할 수 있다.

또한 서버측 샤딩도 지원함으로써 사용자가 응용프로그램의 성능을 향상시키기 위해 클라이언트측 샤딩을 선택하거나, 호환성을 위해 서버측 샤딩으로 선택할 수 있다.

자세한 정보는 Altibase Sharding Guide를 참조하기 바란다.

클라이언트-서버 프로토콜

Altibase를 클라이언트-서버 구조로 운영할 때, 사용자는 응용 시스템의 구성에 적합한 클라이언트-서버 프로토콜을 선택하여 사용할 수 있다. Altibase가 제공하는 통신 프로토콜로는 TCP/IP, IPC, IPCDA와 Unix Domain socket이 있다.

TCP/IP(Transmission Control Protocol/Internet Protocol) 프로토콜은 네트워크 상에서 클라이언트-서버 간에 사용되는 사실상의 표준 통신 프로토콜이다. IPC(Inter Process Communication) 프로토콜은 Altibase가 제공하는 프로토콜로써, 공유메모리(shared memory)를 활용하여 클라이언트와 서버 간에 통신을 하도록 하였다. IPC 방법은 통신 패킷에 대하여 마샬링(marshaling)이 필요 없고 공유 메모리를 이용하기 때문에 다른 통신 프로토콜보다 빠른 통신 속도를 낼 수 있다.

IPCDA는 IPC 기반 통신 방법을 간소화하여 최고의 성능을 내도록 설계되었다. 공유 메모리에 직접 데이터를 읽고 쓰게 하여 메모리 액세스를 최소화하였다. 또한 자체 개발한 스핀락(SpinLock)을 이용하여 프로세스 간 유휴 시간(idle time)을 최소화하였다.

클라이언트 프로그램과 Altibase 서버가 상이한 컴퓨터 시스템에 존재하는 경우에는 인터넷 소켓을 이용한 TCP/IP 프로토콜을 사용하여야 하며, 이들이 동일 컴퓨터 시스템에 존재하는 경우에는 도메인 소켓을 이용한 프로토콜이나 IPC, IPCDA 프로토콜을 사용할 수 있다. 각각의 통신 프로토콜에 대한 성능은 IPCDA, IPC, 도메인 소켓, 인터넷 소켓 순으로 IPCDA가 가장 빠르다. 다만 IPCDA에서는 LOB 데이터를 지원하지 않는다.

서버와 클라이언트의 통신 방법에 대한 자세한 설명은 '서버/클라이언트 통신'을 참조하기 바란다.

데이터베이스 공간

Altibase의 데이터베이스는 데이터베이스의 모든 데이터를 모아 저장하는 하나 이상의 테이블스페이스로 구성되고, 테이블스페이스는 크게 메모리 공간과 디스크 공간으로 나누어진다. Altibase가 생성하는 시스템 테이블스페이스 외에 사용자가 메모리와 디스크 각각의 공간에 테이블스페이스를 추가할 수 있다.

Direct-Path INSERT

Direct-Path INSERT는 데이터를 입력할 때 페이지의 빈 공간을 찾아 들어가는 대신 새로운 페이지를 만들어 데이터를 입력한다. 즉 데이터를 입력할 때 테이블의 빈 공간(free space)을 사용하지 않고, 테이블스페이스로부터 익스텐트(extent)를 새로 할당받는다.

또한 버퍼 관리자를 사용하지 않고, 전용 버퍼(Private Buffer)를 사용하기 때문에 버퍼 공간에 대하여 여러 트랜잭션과의 경합을 줄인다. 그리고 INSERT를 APPEND 방식으로 수행하여 리두(Redo) 및 언두(Undo)를 하지 않거나 줄여 로깅에 들어가는 비용을 줄인다.

데이터베이스 링크

Altibase의 데이터베이스 링크는 지역적으로 분리되어 있으나, 네트워크로 연결된 이기종의 데이터 서버들을 연동하여 개별 데이터들을 통합해 하나의 결과를 생성할 수 있게 한다.

iSQL

Altibase는 iSQL을 이용해 데이터베이스를 관리할 수 있어 빠르고 간편하게 데이터베이스를 관리할 수 있다.

altiComp

Altibase는 altiComp 기능을 사용하여 두 데이터베이스를 테이블 단위로 비교, 검사하여 불일치하는 데이터의 정보를 출력하는 기능과 불일치가 발생한 경우 두 데이터베이스를 일치시키는 기능 등을 제공한다.

iLoader

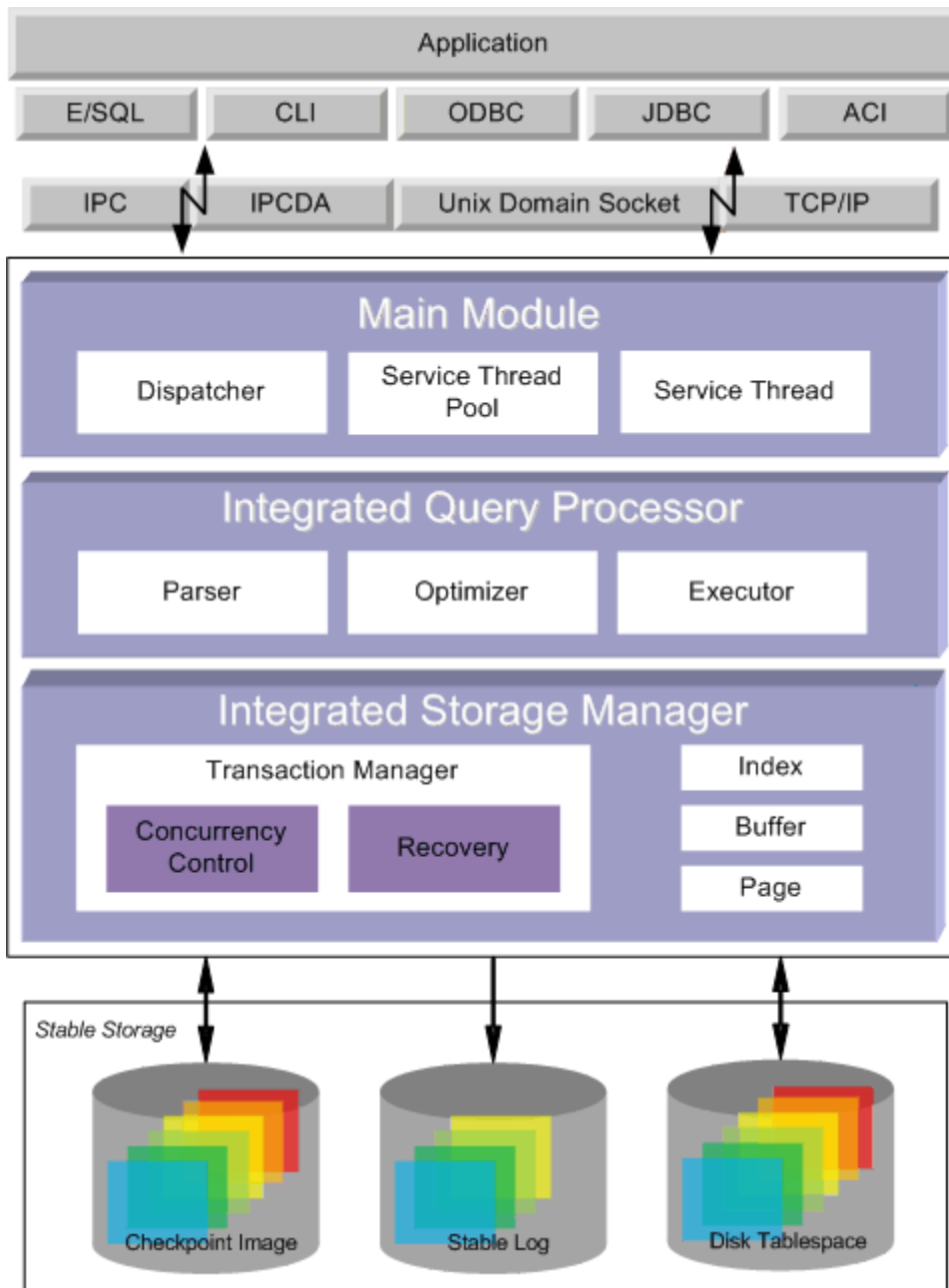
Altibase는 iLoader 유틸리티를 제공하여, 데이터베이스의 이전이나 테이블 단위의 백업 등을 할 때 테이블 단위로 데이터를 다운로드하거나 업로드할 수 있도록 지원한다.

Altibase 구조

본 절에서는 Altibase의 클라이언트-서버 구조를 중심으로 한 전체 구성도, 서버 프로세스 내부 구조, 데이터베이스 구조에 관하여 살펴본다.

전체 구성도

다음은 Altibase와 애플리케이션이 클라이언트-서버 구조로 구성된 시스템을 도식화한 그림이다. 특히 Altibase 서버 부분을 계층화된 구조(Layered Architecture)로 표현하여 클라이언트의 요청과 데이터가 어떤 흐름으로 처리되는지를 볼 수 있다. 서버 외의 부분은 애플리케이션과 데이터베이스 접근을 위한 드라이버(라이브러리) 및 통신 모듈로 구성된다.



[그림 1-2] Altibase 구성도

서버 프로세스 내부 구조

Altibase 서버 프로세스의 내부 구조를 보면 메인 쓰레드, 디스패처, 로드 밸런서, 서비스 쓰레드 풀, 서비스 쓰레드, 체크 포인트 쓰레드, 세션 관리 쓰레드, 가비지 콜렉션 쓰레드, 로그 플러시 쓰레드, 버퍼 플러시 쓰레드 그리고 아카이브로그 쓰레드가 있다. 각각의 쓰레드는 다음과 같은 일을 수행한다.

메인 쓰레드

모든 쓰레드를 생성/종료시키고 생성한 쓰레드들을 관리한다.

디스패처(Dispatcher)

클라이언트의 연결 요청이 있으면, 서비스 쓰레드 풀(pool)에서 대기 상태에 있는 서비스 쓰레드와 요청한 클라이언트를 연결시킨다.

로드 밸런서(Load Balancer)

각 서비스 쓰레드의 부하를 감지하여 서비스 쓰레드를 추가하거나 삭제하고 태스크를 서비스 쓰레드에 분배한다.

서비스 쓰레드

서비스 쓰레드는 질의를 처리한 후 결과를 클라이언트에 반환한다.

Altibase 서버를 구동하면 Altibase는 설정(alibase.properties) 정보에 명시된 개수만큼 서비스 쓰레드를 생성하여 서비스 쓰레드 풀에 저장한다.

체크포인트 쓰레드

고장 복구 시에 일의 양을 줄이기 위하여, 주기적 또는 임의로 현재의 데이터베이스 및 시스템에 대한 상황을 데이터파일에 기록하는 쓰레드이다.

세션 관리 쓰레드(Session Manager)

클라이언트와 서비스 쓰레드 간에 연결된 세션의 상태 즉, 이 세션이 단절되었지의 여부를 감시하는 쓰레드이다.

가비지 콜렉션 쓰레드(Ager)

다중 버전 기법에서는 한 데이터에 대해 필요없는 오래된 데이터가 생성될 수 있다. 가비지 콜렉션 쓰레드(garbage collection thread)는 이러한 데이터가 필요없게 된 순간 그 즉시 메모리 공간을 회수하여, 재사용할 수 있도록 조치를 취함으로써 메모리 사용의 효율성을 극대화한다. 가비지 콜렉션 쓰레드를 Ager라고도 한다.

로그 플러시 쓰레드

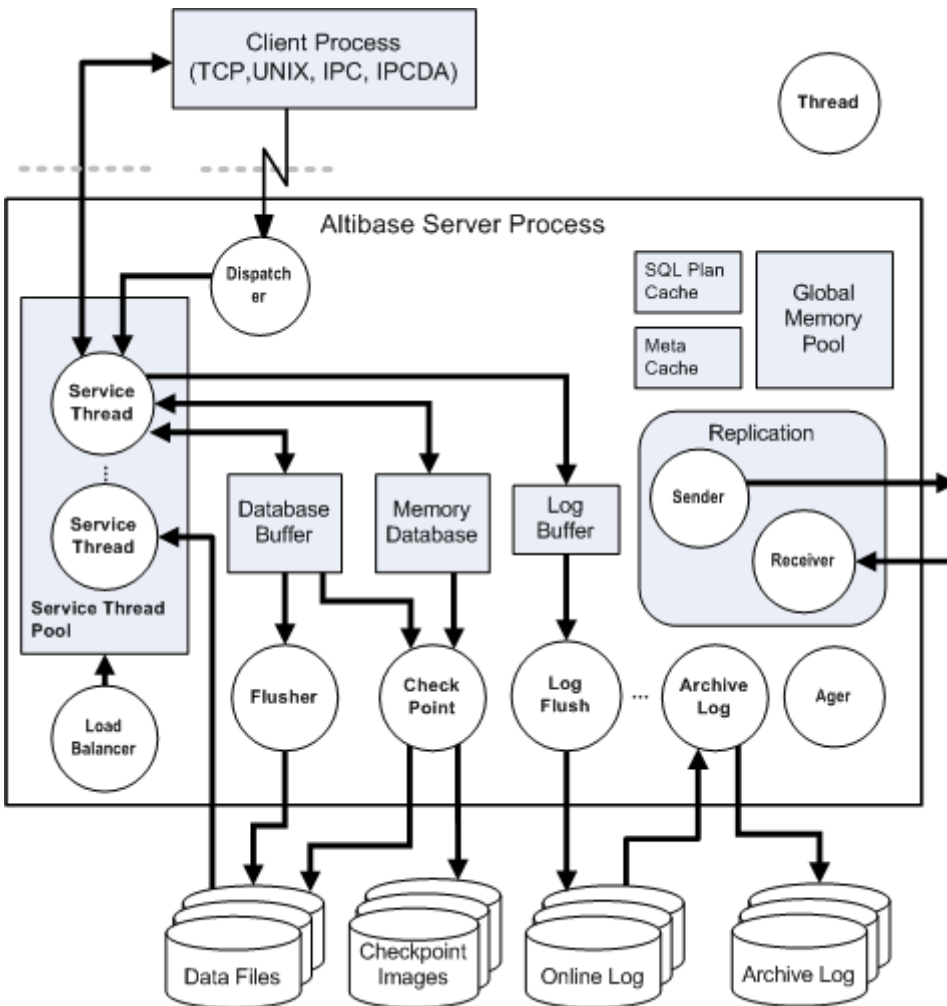
로그 플러시 쓰레드(log flush thread)는 데이터베이스 내의 모든 트랜잭션이 생성한 로그를 관리하고, 로그 버퍼에 모여진 다량의 로그 데이터를 로그 디스크에 반영하는 기능을 한다. 디스크에 완전히 반영(sync)된 로그는 데이터베이스 시스템의 장애 및 재해 발생 시에 데이터베이스를 안전하게 복구하는데 사용된다.

버퍼 플러시 쓰레드(Flusher)

버퍼 풀의 모든 메모리가 사용 중이면 디스크 I/O를 발생시켜서 수행중인 트랜잭션에 성능 상의 기복(Jitter) 현상을 일으키게 된다. 버퍼 플러시 쓰레드는 주기적으로 버퍼를 체크하여 일정 양 이상의 가용 버퍼 메모리를 항상 유지하도록 하며, 사용하지 않는 페이지를 디스크에 내리고 메모리를 사용 가능하게 만드는 역할을 한다. 버퍼 플러시 쓰레드를 Flusher라고도 부른다.

아카이브로그 쓰레드

매체 오류에 대한 복구를 지원하기 위해 주기적으로 온라인 로그 파일들을 프로퍼티에 지정된 위치로 복사하는 쓰레드이다. 복사할 경로는 ARCHIVE_DIR 프로퍼티로 지정하면 된다. Altibase가 아카이브 모드로 운영될 때만 동작한다.



[그림 1-3] Altibase 프로세스의 내부 구조

데이터베이스의 물리적 구조

Altibase의 데이터베이스는 물리적으로 로그앵커 파일, 로그 파일, 데이터 파일로 구성되어 있다.

로그앵커 파일

로그앵커 (loganchor) 파일은 데이터 파일과 로그와의 관계를 나타내는 중요한 정보를 포함한다. 이는 로그를 기준으로 한 시점에서 데이터 파일의 총체적인 정보를 나타낸다. 이 파일은 데이터 파일과 함께 중요한 백업 대상이다.

로그 파일

로그 파일("리두 로그 파일"로 불리기도 함)은 트랜잭션의 원자성(Atomicity)과 영속성(Durability)을 유지하기 위해 사용된다. 원자성은 트랜잭션의 철회(rollback)를 통해 트랜잭션 수행 이전의 상태로 복귀할 수 있도록 하는 것이고, 영속성은 정상적으로 종료(commit)된 트랜잭션이 다양한 데이터베이스 장애로부터 원래의 내용을 복구할 수 있도록 하는 것이다.

로그 파일은 prepare 로그 파일과 active 로그 파일, archive 로그 파일로 구분할 수 있다. active 로그 파일은 실행중인 트랜잭션의 로그가 기록되는 로그파일이다. Prepare 로그 파일은 로그 기록 성능을 향상시키기 위해 미리 만들어지는 로그 파일로 실제 로그가 기록되기 전까지는 비어있는 상태이다. archive 로그 파일은 복구를 위하여 백업된 로그 파일로 기록이 완료된 로그 파일들이다.

로그 파일은 현재의 데이터베이스 상태를 가지는 매우 중요한 파일로서, 만일 현재 로그 파일이 손상을 입었을 경우 당시 작업의 유무에 관계없이 데이터베이스 전체가 손상을 입게 된다. 기존 로그 파일은 일반적으로 데이터 파일이 손상될 경우 백업 파일과 함께 데이터베이스를 복구하는데 사용된다.

데이터 파일

데이터 파일 중 SYS_TBS_MEM_DATA에는 기본으로 생성되는 시스템 메모리 테이블스페이스가 저장되며, SYS_TBS_MEM_DIC에는 메타 테이블들이, system001.dbf 파일에는 기본으로 생성되는 디스크 테이블스페이스 (SYS_TBS_DISK_DATA)가 저장된다. 또한 temp001.dbf 파일에는 쿼리 수행 시 중간 결과들이 저장되는 임시 테이블스페이스가 저장되며, undo001.dbf 파일에는 다중버전 기법(MVCC: Multi-Version Concurrency Control)에서 사용되는 이전 이미지 정보들이 저장되는 언두 테이블스페이스가 저장된다.

Altibase는 페이지 단위로 데이터파일 내에서 저장 공간을 관리한다. 페이지는 데이터베이스에 의해 사용되는 데이터의 가장 작은 단위이다.

페이지는 데이터베이스를 관리하기 위한 정보를 담고 있는 카탈로그 페이지와 사용자 데이터를 저장하는 데이터 페이지로 나누어 진다. 카탈로그 페이지는 현재 생성된 데이터베이스에 대한 상세한 명세를 담고 있으며, Altibase의 구동 및 종료 시 데이터베이스의 일관성 검사에 사용된다.

카탈로그 페이지는 데이터베이스에서 사용되는 자기 자신을 제외한 나머지 데이터 페이지에 대한 리스트 및 사용정보를 담고 있으며, 백업 데이터베이스의 가장 첫 번째 페이지에 위치하고 있는 매우 중요한 페이지 영역이다.

데이터 페이지는 실제로 사용자 데이터가 저장되는 영역이며, 페이지 헤더와 페이지 본체(body)로 나뉘어진다. 페이지 헤더는 서로 간의 리스트를 유지하기 위한 링크

정보와 타입, 그리고 자기 자신의 페이지 번호로 구성되어 있다. 페이지 본체는 여러 개의 슬롯으로 분할된다. 이 슬롯이 실제 데이터가 저장되는 최종 저장소이다.

데이터베이스 논리적 구조

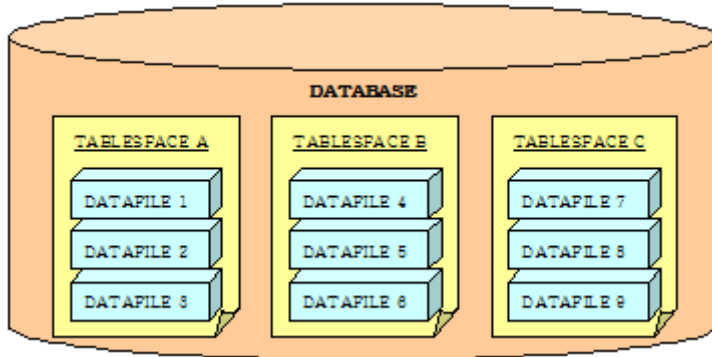
Altibase는 논리적으로 메모리와 디스크 테이블스페이스 내에 데이터를 저장하고, 물리적으로는 해당 테이블스페이스와 관련된 데이터파일 내에 저장한다.

Altibase 데이터베이스를 구성하는 각각의 테이블스페이스는 하나 이상의 데이터파일로 구성된다. 단, 하나의 데이터파일은 하나의 테이블스페이스에만 소속된다.

데이터베이스, 테이블스페이스 그리고 데이터파일은 밀접한 관계가 있으며 이 관계를 정리하면 다음과 같다.

데이터베이스는 논리적으로 테이블스페이스라는 저장 단위로 구성되어 있다. 즉, 테이블스페이스는 데이터베이스의 모든 데이터를 저장하는 논리적인 공간이다. 데이터베이스는 물리적으로 데이터파일이라고 불리는 하나 이상의 파일들로 구성된다. 즉 데이터파일은 데이터베이스의 모든 데이터를 저장하는 물리적 공간이다.

다음 그림은 테이블스페이스와 데이터파일의 관계를 설명한다.



[그림 1-4] 데이터베이스 논리적 구조

Altibase는 데이터베이스 내의 모든 데이터에 논리적 데이터베이스 영역인 테이블스페이스를 할당한다. 데이터베이스 영역 할당의 단위로는 페이지(page), 익스텐트(extent) 그리고 세그먼트(segment)가 있다.

페이지는 가장 작은 논리적 저장 단위로, 모든 데이터는 페이지 내에 저장된다.

페이지의 논리적인 다음 단계는 익스텐트이다. 즉 일정 개수의 연속적인 페이지들이 모여서 익스텐트라는 데이터베이스 영역을 형성한다.

익스텐트 상위의 논리적 데이터베이스 저장 단계를 세그먼트라고 한다. 하나의 세그먼트는 일련의 익스텐트의 집합이며, 한 세그먼트 내의 모든 익스텐트는 같은

테이블스페이스에 저장된다.

자세한 내용은 "6장 테이블스페이스"를 참조한다.

기타 제어 파일들

부트 로그 파일 (altibase_boot.log)

Altibase 서버가 동작된 상태를 기록한다. 이 파일이 기록하고 있는 정보로는 Altibase 구동 및 종료 시 얻어지는 시스템 정보에 대한 세부사항이 있으며, 또한 Altibase의 비정상 종료 시 Altibase의 오류 발생 상태를 기록한다.

프로퍼티 파일 (altibase.properties)

Altibase 서버의 환경 설정을 위한 파일이며 Altibase 서버의 운용 방식과 튜닝에 관한 모든 구성 요소를 포함하고 있다.

에러 메시지 파일

데이터 저장 관리 모듈, 질의 처리 모듈, Altibase 서버 메인 모듈, 그리고 함수 실행이나 데이터 타입과 관련된 오류 메시지를 수록한 파일이다.

2.Altibase 구성요소

이 장에서는 Altibase의 주요 구성요소에 대해서 설명한다. 사용자는 Altibase 패키지 설치 후에 실행 바이너리 부문과 프로그래밍 라이브러리 부문 등의 구성요소에 대해서 확인할 수 있다.

Altibase 디렉토리

Altibase를 설치하면 다음의 디렉토리가 생성된다. Altibase의 홈 디렉토리는 환경 변수 ALTIBASE_HOME에 지정된다. 홈 디렉토리는 bin, conf, lib, include, msg, dbs, logs, sample, install, altiComp, trc, admin, 그리고 arch_logs 디렉토리를 포함하고 있다.

각 디렉토리의 역할과 포함하는 내용에 관하여 설명한다.

admin 디렉토리

Altibase의 시스템 정보와 관련된 view를 생성하는 adminview.sql 파일과 프로시저, 테이블 정보를 볼 수 있는 프로시저를 생성하는 SQL 파일들이다.

arch_logs 디렉토리

복구를 위해 로그 파일을 백업하는 백업 디렉토리이다. 이 디렉토리의 위치 및 디렉토리 이름은 프로퍼티 파일에 명시할 수 있다.

altiComp 디렉토리

이중화 동작 시 발생한 데이터베이스간의 데이터 불일치를 해결하는 Altibase 유틸리티인 altiComp의 예제 스크립트 파일이 들어있는 디렉토리이다.

자세한 설명은 *Utilities Manual*의 *altiComp*를 참조한다.

bin 디렉토리

Altibase의 실행 파일을 포함한 Altibase 관리도구와 사용자 지원 도구들의 실행 파일이 존재하는 디렉토리이다.

bin 디렉토리에는 다음과 같은 파일이 존재한다.

```
aexport, altiAudit, altibase, altierr, altimon.sh, altipasswd, altiPofile,  
altiComp, checkServer, dumpbi, dumpct, dumpdb, dumpddf, dumpla, dump1f, iloader,  
isql, killCheckServer, server, apre
```

iloader, isql, apre에 대한 자세한 설명은 *iLoader User's Manual*, *iSQL User's Manual*, *Precompiler User's Manual*을 각각 참조하고, 나머지 유틸리티에 대한 자세한 설명은 *Utilities Manual*을 참조한다.

conf 디렉토리

conf 디렉토리에는 아래의 파일들이 존재한다.

- altibase_user.env: Altibase 운영을 위한 환경변수 설정 파일
- altibase.properties: Altibase용 설정 파일. 설정 가능한 프로퍼티에 대한 설명은 *General Reference*를 참조하기 바란다.
- license: Altibase 라이선스 파일
- dblink.conf: 데이터베이스 링크용 설정 파일. 설정 가능한 프로퍼티에 대한 설명은 *General Reference*를 참조하기 바란다.
- aexport.properties: aexport용 설정 파일. 설정 가능한 프로퍼티에 대한 설명은 *Utilities Manual*을 참조하기 바란다.

dbs 디렉토리

기본 프로퍼티를 이용할 경우 데이터베이스의 파일들이 생성되는 디렉토리이다. 이 디렉토리의 위치 및 디렉토리 명은 프로퍼티에 명시되어 있다.

SYS_TBS_MEM_DATA 파일에는 기본으로 생성되는 시스템 메모리 테이블스페이스가, SYS_TBS_MEM_DIC 파일에는 메타 테이블이, system001.dbf 파일에는 기본으로 생성되는 디스크 테이블스페이스, temp001.dbf 파일에는 쿼리 수행 시 필요한 임시 결과들이 저장된다.

undo001.dbf 파일에는 SQL문 수행과 복구에 필요한 이전 이미지 정보들이 저장된다.
.dwf 파일은 더블 라이트 버퍼 파일로서 디스크 페이지가 임시로 저장된다.

include 디렉토리

Altibase CLI 라이브러리 등을 이용하여 응용 프로그램을 작성할 때 필요한 헤더 파일을 수록한 디렉토리이다.

alaAPI.h

Altibase 로그 분석기(ALA)에서 사용하는 API 헤더 파일이다.

sqlcli.h

클라이언트 응용 프로그램을 작성할 때 필요한 헤더 파일이다.

sqltypes.h

ODBC 응용 프로그램 개발시 필요한 기초 데이터 타입에 대한 정보를 담고 있다.

sqlucode.h

유니코드 정의 헤더 파일이다.

ulpLibInterface.h

C/C++ 전처리기(Precompiler)로 응용 프로그램 개발시 오류 처리 SQL 문장 구조에 대한 정보를 담고 있다.

install 디렉토리

Altibase 응용프로그램 작성에 필요한 makefile을 위한 매크로 설정 등이 포함된 altibase_env.mk 파일과 README 파일이 있다.

lib 디렉토리

응용 프로그램 작성에 필요한 라이브러리를 수록한 디렉토리이며 다음과 같은 파일을 갖고있다. 각각의 라이브러리를 이용하여 응용 프로그램을 작성하는 방법은 *Getting Started Guide*에서 설명한다.

Altibase.jar

Altibase를 자바 응용프로그램에서 사용하기 위한 JDBC 드라이버이다. 순수 자바 언어로 구현된 Type 4 드라이버이다. 자세한 내용은 *JDBC User's Manual*을 참조한다.

libapre.a

내장 SQL 프로그램을 작성할 때 필요한 라이브러리이다. 내장 SQL 프로그램 작성에 관한 자세한 내용은 *Precompiler User's Manual*을 참조한다.

libodbccli.a

Altibase CLI 응용프로그램 작성을 위한 라이브러리이다. 자세한 내용은 *CLI User's Manual*을 참조한다.

libalticapi.a

Altibase ACI 애플리케이션 작성을 위한 라이브러리이다. 자세한 내용은 *ACI User's Manual*을 참조한다.

libaltibase_odbc-64bit-ul64.so

유닉스 계열 운영체제에서 사용할 수 있는 Altibase의 ODBC 드라이버이다. 설치 패키지와 운영 체제에 따라 파일 확장자와 파일 이름이 다를 수 있다. 자세한 내용은 *ODBC User's Manual*을 참조한다.

그 외

- libchksvr.a: Altibase CheckServer API용 라이브러리. *API User's Manual*을 참조한다.
- libiload.a: Altibase iLoader API용 라이브러리. *API User's Manual*을 참조한다.
- libaltibaseMonitor.a: Altibase Monitoring API용 라이브러리. *Monitoring API Developer's Guide*를 참조한다.
- libsesc.a: 하위 호환성을 위해 제공되며, libapre.a와 동일하다.

logs 디렉토리

로그앵커 파일들과 로그 파일들이 존재하는 디렉토리다.

이 디렉토리의 위치 및 디렉토리 명은 프로퍼티 파일에 명시할 수 있다. 로그앵커 파일명과 로그 파일명은 Altibase 시스템에 의해 자동으로 결정된다. 로그 앵커를 가진 파일 시스템의 오류에 대비하기 위해서는 프로퍼티를 변경하여 여러 개의 로그 앵커 파일들을 각각 서로 다른 파일 시스템에 두어 관리하는 것이 좋다.

msg 디렉토리

오류 메시지를 수록한 파일들을 포함하는 디렉토리다. 다음과 같은 파일이 있다. 각 모듈에 대해 US7ASCII와 KO16KSC5601 캐릭터셋용의 두 메시지 파일이 존재하지만, 파일 안의 오류 메시지는 동일하게 영문만 제공된다.

E_SM_US7ASCII.msb

자료 저장 관리 모듈에서 발생할 수 있는 오류 메시지를 수록한 파일이다.

E_QP_US7ASCII.msb

질의 처리 모듈에서 발생할 수 있는 오류 메시지를 수록한 파일이다.

E_MM_US7ASCII.msb

Altibase 서버의 메인 모듈에서 발생할 수 있는 오류 메시지를 수록한 파일이다.

E_CM_US7ASCII.msb

Altibase 통신 모듈에서 발생할 수 있는 오류 메시지를 수록한 파일이다.

E_RP_US7ASCII.msb

Altibase 이중화 모듈에서 발생할 수 있는 오류 메시지를 수록한 파일이다.

E_ST_US7ASCII.msb

Altibase 공간 데이터 처리 모듈에서 발생할 수 있는 오류 메시지를 수록한 파일이다.

E_DK_US7ASCII.msb

Altibase 데이터베이스 링크 모듈에서 발생할 수 있는 오류 메시지를 수록한 파일이다.

E_ID_US7ASCII.msb, E_MT_US7ASCII.msb

함수 실행이나 데이터 타입과 관련된 오류 메시지를 수록한 파일이다.

sample 디렉토리

Altibase의 응용 프로그램을 샘플로 제공한 디렉토리다.

JDBC, CLI, C/C++ 전처리기(precompiler) 라이브러리를 이용하여 작성된 프로그램과 Makefile이 수록되어 있다.

trc 디렉토리

Altibase 운영 상태를 기록한 파일들이 존재한다. 서버내의 각 모듈은 해당하는 트레이스 파일에 기록한다.

altibase_boot.log

Altibase 서버가 동작된 상태를 기록하고 있다. 이 파일이 기록하고 있는 정보로는 Altibase 구동 및 종료시 생성되는 시스템 정보에 대한 세부사항이 있다.

altibase_error.log

서버에서 발생하는 오류 메시지가 기록되는 파일이다. 또한 Altibase의 비정상 종료시 Altibase 프로세스의 콜 스택이 기록된다.

altibase_trc.log

Altibase를 시작한 이후부터 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일이다. 이 파일에는 프로세스 내에서 스레드 별로 발생하는 경고 메시지 등이 순차적으로 저장된다.

altibase_dump.log

Altibase 프로세스가 비정상적으로 종료하는 시점의 작업 메모리가 덤프되는 파일이다. Altibase 프로그램의 오류를 진단하고 디버깅하는 데 사용된다.

altibase_sm.log

저장관리자 모듈에서 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일들이다.

altibase_rp.log

이중화 모듈에서 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일들이다.

altibase_qp.log

질의 처리 모듈에서 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일들이다.

altibase_mm.log

메인 모듈에서 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일들이다.

altibase_cm.log

통신 모듈에서 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일들이다.

altibase_lb.log

로드밸런서에서 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일들이다.

altibase_snmp.log

SNMP에서 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일들이다.

altibase_dk.log

데이터베이스 링크 모듈에서 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일들이다.

altibase_ipc.log

IPC로 접속시 생성된 자원(resource)들에 대한 정보가 기록되는 파일이다.

altibase_ipcda.log

IPCDA로 접속 시 생성된 자원(resource)들에 대한 정보가 기록되는 파일이다.

altibase_xa.log

XA 인터페이스를 이용해서 Altibase에 수행되는 글로벌 트랜잭션에서 발생하는 경고 메시지나 트레이스 메시지 등이 기록되는 파일이다.

killCheckServer.log

killCheckServer 유틸리티의 실행 결과가 기록되는 파일이다.

실행 바이너리

여기에 설명된 외의 이들 바이너리 파일에 대한 더 자세한 정보는 *Utilities Manual*을 참고하기 바란다..

aexport

Altibase 버전을 업그레이드할 때 필요한 일련의 작업들을 수행할 수 있는 도구로, 업그레이드를 위한 SQL 스크립트 파일, iSQL 실행 쉘 파일, iLoader 실행 쉘 파일을 자동으로 만든다.

altibase

클라이언트-서버 구조로 운영할 때의 서버이다.

altierr

오류 코드에 대한 세부 내용을 검색하여 출력한다.

altimon.sh

Altibase의 동작 상태를 모니터링하는 쉘 스크립트 프로그램이다.

altiProfile

SQL 문의 통계정보(수행 횟수, 수행 시간)를 수집하는 도구이다.

altipasswd

sys 계정의 패스워드를 변경하기 위한 도구이다.

altiComp

altiComp는 두 데이터베이스를 테이블 단위로 비교, 검사하여 불일치하는 정보를 출력하는 기능과 불일치가 발생한 경우 두 데이터베이스를 일치시키는 기능 두 가지를 제공한다.

이에 대한 자세한 내용은 *Utilities Manual*을 참조한다.

checkServer

Altibase의 상태를 체크하여 비정상 종료시 수행해야 할 일을 스크립트 파일로 만들어 실행할 수 있도록 한다.

dumpla

Altibase 로그엔커 파일의 내용을 출력 및 검사한다.

dumplf

Altibase 로그 파일의 내용을 출력 및 검사한다.

iload

데이터베이스의 특정 테이블을 로드 및 언로드할 수 있는 도구이다. 이 도구에 대한 자세한 내용은 *iLoader User's Manual*을 참조한다.

isql

대화형으로 Altibase에 질의를 수행할 수 있는 도구이다. 이 도구에 대한 자세한 내용은 *iSQL User's Manual*을 참조한다.

killCheckServer

실행중인 checkServer를 종료시킨다.

server

Altibase의 구동 및 종료, 재시작 등의 동작을 수행할 수 있도록 작성된 쉘 스크립트 프로그램이다.

apre

내장 SQL문을 사용하여 응용 프로그램을 작성한 후, 작성된 응용 프로그램을 전처리(precompile)하기 위한 전처리 실행 파일이다.

자세한 설명은 Precompiler User's Manual을 참조한다.

Altibase 라이브러리

Altibase의 응용 프로그램을 작성할 때 필요한 구성 요소들로서, 다음과 같은 것들이 있다.

- C 또는 C++ 언어로 프로그램을 작성할 때 필요한 라이브러리
- Altibase CLI 인터페이스를 제공하는 라이브러리 (libodbcli.a)
- 자바 언어로 프로그래밍할 때 필요한 자바 클래스 라이브러리 (Altibase.jar)
- 프로그래밍에 필요한 헤더 파일들

이에 대해서는 Getting Started Guide에서 자세히 설명한다.

3.데이터베이스 생성

Altibase 설치 후에 데이터베이스 관리자는 사용자 데이터 발생량을 예측하여 데이터베이스를 생성하고 관리해야 한다. 이 장에서는 데이터베이스 생성시에 알고 있어야 할 주요사항들에 대해서 설명하고 있다.

데이터베이스 생성

Altibase의 데이터베이스는 데이터의 논리적 저장 단위인 테이블스페이스로 구성된다. Altibase는 데이터를 논리적으로는 테이블스페이스에, 물리적으로는 테이블스페이스에 대응하는 데이터파일에 저장한다. 데이터베이스 서버를 구동하기 전에, 데이터베이스를 미리 생성시켜 놓아야 한다.

여기에서는 테이블스페이스와 로깅 시스템의 종류 및 데이터베이스 생성 방법에 관하여 설명한다.

테이블스페이스의 종류

Altibase의 데이터베이스는 여러 개의 테이블스페이스로 구성된다. 테이블스페이스는 그 사용처와 데이터를 저장하는 방법에 따라서 여러 종류로 분류된다.

CREATE DATABASE 구문을 실행하면, 체크포인트 이미지와 데이터 파일이 기본으로 \$ALTIBASE_HOME/dbs/ 디렉토리에 생성된다.

Note: 사용자가 테이블스페이스를 생성하거나 테이블스페이스에 파일을 추가할 때 명시하는 파일의 확장자와 파일의 경로에는 제한이 없다.

Altibase에서 제공하는 기본 테이블스페이스는 아래와 같다.

메모리 테이블스페이스

메모리 테이블스페이스는 메모리에 존재한다. 딕셔너리 테이블들과 메모리 테이블들, 그리고 이에 관련된 다양한 데이터베이스 객체들이 저장되는 테이블스페이스이다.

디스크 테이블스페이스

디스크 테이블들과 디스크 인덱스들이 저장되는 테이블스페이스이다. 이는 다시 시스템 데이터 테이블스페이스와 사용자 데이터 테이블스페이스로 구분된다.

언두 테이블스페이스(Undo Tablespace)

디스크 테이블에 존재하는 레코드들의 다중버전 동시성 제어 (MVCC: Multi-Versioned Concurrency Control)를 위해 변경 이전 이미지를 일정 기간 동안 저장해두는 테이블스페이스 이다.

임시 테이블스페이스(Temporary Tablespace)

질의를 처리하는 과정에서 발생하는 임시 테이블들과 인덱스들을 저장하는 테이블스페이스이다. 데이터 테이블스페이스와 마찬가지로 시스템 임시 테이블스페이스와 사용자 임시 테이블스페이스로 나뉜다.

휘발성 테이블스페이스(Volatile Tablespace)

디스크 입출력을 하지 않고, 메모리에 객체를 저장하는 테이블스페이스이므로 보다 빠른 성능이 보장된다. 데이터베이스 서비스 종료시 모든 휘발성 데이터 객체들이 사라진다. 휘발성 테이블스페이스의 크기는 시스템의 사용 가능한 물리적 메모리 공간을 초과할 수 없다.

로깅 시스템

데이터베이스 내의 데이터들은 어떤 상황 하에서도 영속성 (Durability)을 가져야 한다. Altibase는 다음의 두 가지 파일들로 로깅 시스템을 구성하여 데이터의 영속성을 보장한다.

로그 파일

트랜잭션 수행 중에 발생할 수 있는 비정상 종료에 대비하여 시스템 복구 (system recovery)를 할 수 있도록 작성되는 로그 레코드들을 기록하는 파일들이다. 로그 파일의 이름은 logfile**이다 (**은 로그 파일의 일련 번호이다).

로그 앵커(Log Anchor) 파일

테이블스페이스들에 대한 정보와 데이터파일들의 위치, 체크 포인트 관련 정보 등 서버 운용에 관련된 중요한 데이터가 저장된 파일이다. 서버가 정상적으로 구동 되려면 이 파일의 내용이 유효하여야 하며, 그렇지 않을 경우에는 서버를 구동 시킬 수 없다. 또한 로그 앵커 파일은 데이터베이스 복구시에도 사용된다.

최초 데이터베이스 생성시 로그 파일과 로그 앵커 파일들은 \$ALTIBASE_HOME/logs/ 위치에 생성된다.

Altibase는 이 로그 앵커 파일들을 3개로 유지하며, 데이터베이스 생성 시 로그 파일들과 같은 위치에 생성되지만, 3개의 로그 앵커 파일들을 서로 다른 파일 시스템에 두기를 권장하고 있다. 로그 앵커 파일의 위치에 관련된 프로퍼티는 LOGANCHOR_DIR 이다.

이 프로퍼티에 대한 자세한 설명은 *Getting Started Guide*을 참고하기 바란다.

데이터베이스 생성 준비

데이터베이스를 생성하려면 Altibase 패키지에 제공되는 iSQL유틸리티를 사용한다.

먼저 iSQL 유틸리티를 SYSDBA 모드로 실행한다.

```
$ isql -u sys -p manager -sysdba
```

이는 Altibase가 실행되어 있지 않은 경우에는 데이터베이스에 접속하지 않고 isql을 관리자 모드 (admin mode)로 띄우는 것이며, 실행 결과는 아래와 같다.

```
-----
Altibase Client Query utility.
Release Version 7.1.0.0.1
Copyright 2000, ALTIBASE Corporation or its subsidiaries.
All Rights Reserved.
-----
ISQL_CONNECTION = TCP, SERVER = 127.0.0.1, PORT_NO = 20300
iSQL(sysdba)>
```

위의 상태가 되면 일단 CREATE DATABASE 명령을 수행하기 위해 서버 프로세스를 구동 시켜야 한다. 서버의 구동은 다음과 같은 순서로 이루어 진다.

1. 1단계: Pre-Process 단계

서버를 구동하기 이전 단계로, Altibase는 데이터베이스 메모리를 초기화한다.
데이터베이스의 생성은 Process 단계에서 가능하며, 이 단계에서 Process 단계로 가려면 다음의 명령을 실행한다.

```
iSQL> startup process
Trying Connect to Altibase.. Connected with Altibase.
TRANSITION TO PHASE: PROCESS
Command execute success.
```

1. 2단계: Process 단계

CREATE DATABASE 구문으로 데이터베이스를 생성하거나 Altibase 프로퍼티들을 조회하고 변경할 수 있는 단계이다.

2. 3단계: Control 단계

데이터베이스 파일이 모두 로드되어 있는 상태의 단계이다. 또한 재시작 복구(restart recovery)를 위한 준비도 완료된 단계이다. 재시작 복구에 대한 설명은 10장의 “데이터베이스 복구” 절을 참고한다.

3. 4단계: Meta 단계

복구가 완료된 단계이다. 이 단계에서는 메타 데이터의 업그레이드와 온라인 로그의 리셋(reset)이 가능하다.

4. 5단계: Service 단계

사용자에게 서비스를 제공할 준비가 된 최종 단계이다.

데이터베이스 생성

Process 단계에서 데이터베이스를 생성하기 위한 CREATE DATABASE 명령은 아래와 같이 수행한다. CREATE DATABASE 구문의 자세한 사용법은 *SQL Reference*를 참조한다.
여기서는 기본 옵션을 사용해서 데이터베이스를 생성하는 예를 보여주고 있다.

```
iSQL> create database mydb initsize=50M noarchivelog character set ksc5601 national character set utf16;
DB Info (Page Size      = 32768)
      (Page Count      = 1537)
      (Total DB Size = 50364416)
      (DB File Size   = 1073741824)
      Creating MMDb FILES      [SUCCESS]
      Creating Catalog Tables [SUCCESS]
      Creating DRDB FILES      [SUCCESS]
      [SM] Rebuilding Indices [Total Count:0] [SUCCESS]
DB Writing Completed. All Done.
Create success.
```

데이터베이스 서버 종료

데이터베이스의 생성이 완료되면 이를 위해 띄웠던 서버를 종료하거나, 서비스 단계로 진행할 수 있다. 서버의 종료는 다음과 같이 수행한다.

```
iSQL(sysdba)> shutdown abort  
iSQL(sysdba)>
```

서버를 종료하면 isql은 다시 서버에 접속하지 않은 Pre-Process 상태가 되며, 서버 프로세스도 종료된다.

shutdown 명령의 옵션으로는 abort외에 immediate와 normal이 더 있으나, 이들은 서버가 service 단계일 때만 수행이 가능하다.

데이터베이스 생성 관련 프로퍼티

CREATE DATABASE 구문을 수행할 때 지정하지 않은 속성들은 Altibase 프로퍼티 파일의 설정에 의해 결정되는데, 그 파일은 \$ALTIBASE_HOME/conf/altibase.properties이다. 관련있는 프로퍼티들은 아래와 같다. 표에서 물음표 (“?”)는 환경변수 ALTIBASE_HOME에 설정된 경로를 가리킨다.

아래 표에 나열한 데이터베이스 초기화와 관련된 Altibase 프로퍼티에 대해 완벽히 이해하기 바란다.

프로퍼티 이름	설명	기본값
DB_NAME	생성할 데이터베이스의 이름	mydb
MEM_DB_DIR	데이터베이스 파일들이 위치할 디렉토리.	?/dbs
SERVER_MSGLOG_DIR	Altibase 운용 중 발생하는 서버의 메시지를 기록하는 파일 (altibase_boot.log)이 위치하는 디렉토리	?/trc
MEM_MAX_DB_SIZE	전체 메모리 테이블스페이스들의 최대 크기	4G
LOGANCHOR_DIR	로그 앵커 파일들이 위치할 디렉토리. 최대 3개까지 지정할 수 있다	?/logs
LOG_DIR	로그 파일들이 위치할 디렉토리	?/logs
LOG_FILE_SIZE	로그 파일 하나의 크기	10M

프로퍼티 이름	설명	기본값
EXPAND_CHUNK_PAGE_COUNT	한 번에 할당하는 메모리 테이블스페이스 페이지의 개수	3200
TEMP_PAGE_CHUNK_COUNT	한 번에 할당하는 메모리 테이블스페이스 임시 페이지의 개수	128
SYS_DATA_TBS_EXTENT_SIZE	시스템 데이터 테이블스페이스의 익스텐트 한 개의 크기	256K
SYS_DATA_FILE_INIT_SIZE	CREATE DATABASE 구문 실행 시 생성되는 시스템 테이블스페이스를 위한 데이터 파일의 최초 크기	100M
SYS_DATA_FILE_MAX_SIZE	시스템 테이블스페이스의 데이터 파일의 최대 크기	2G
SYS_DATA_FILE_NEXT_SIZE	시스템 테이블스페이스의 데이터 파일이 자동 확장될 때의 확장 크기	1M
SYS_TEMP_TBS_EXTENT_SIZE	임시 테이블스페이스의 익스텐트 한 개의 크기	256K
SYS_TEMP_FILE_INIT_SIZE	CREATE DATABASE 실행 시 생성되는 임시 테이블스페이스를 위한 데이터 파일의 최초 크기	100M
SYS_TEMP_FILE_MAX_SIZE	임시 테이블스페이스의 데이터 파일의 최대 크기	2G
SYS_TEMP_FILE_NEXT_SIZE	임시 테이블스페이스의 데이터 파일이 자동 확장될 때의 확장 크기	1M
SYS_UNDO_TBS_EXTENT_SIZE	언두 테이블스페이스의 익스텐트 한 개의 크기	128K
SYS_UNDO_FILE_INIT_SIZE	CREATE DATABASE 실행 시 생성되는 언두 테이블스페이스를 위한 데이터 파일의 최초 크기	100M

프로퍼티 이름	설명	기본값
SYS_UNDO_FILE_MAX_SIZE	언두 테이블스페이스의 데이터 파일의 최대 크기	2G
SYS_UNDO_FILE_NEXT_SIZE	언두 테이블스페이스의 데이터 파일이 자동 확장될 때의 확장 크기	1M
USER_DATA_TBS_EXTENT_SIZE	사용자 데이터 테이블스페이스의 익스텐트 한 개의 크기	256K
USER_DATA_FILE_INIT_SIZE	CREATE DATABASE 실행 시 생성되는 사용자 테이블스페이스를 위한 데이터 파일의 최초 크기	100M
USER_DATA_FILE_MAX_SIZE	사용자 데이터 테이블스페이스의 데이터 파일의 최대 크기	2G
USER_DATA_FILE_NEXT_SIZE	사용자 데이터 테이블스페이스의 데이터 파일이 자동 확장될 때의 확장 크기	1M
USER_TEMP_TBS_EXTENT_SIZE	사용자 임시 테이블스페이스의 익스텐트 한 개의 크기	256K
USER_TEMP_FILE_INIT_SIZE	CREATE DATABASE 실행 시 생성되는 사용자 임시 테이블스페이스의 데이터 파일의 최초 크기	100M
USER_TEMP_FILE_MAX_SIZE	사용자 임시 테이블스페이스의 데이터 파일의 최대 크기	2G
USER_TEMP_FILE_NEXT_SIZE	사용자 임시 테이블스페이스의 데이터 파일이 자동 확장될 때의 확장 크기	1M
ADD_EXTENT_NUM_FROM_TBS_TO_SEG	세그먼트가 확장될 때 새로 할당되는 익스텐트의 개수	1
ADD_EXTENT_NUM_FROM_SYSTEM_TO_TBS	테이블스페이스가 확장될 때 새로 할당되는 익스텐트의 개수	4

프로퍼티에 대한 보다 자세한 내용은 *General Reference*를 참조하기 바란다.

4.Altibase 구동 및 종료

데이터베이스를 생성 후 서비스를 제공하기 위해서는 서버를 서비스 단계까지 구동하여야 한다. 이 장에서는 데이터베이스 구동과 종료 시에 참고할 사항들을 설명하고 있다.

Altibase 구동

Altibase 서버를 구동하는 방법은 두 가지가 있다.

- 데이터베이스 관리자가 sys 계정으로 서버에 로그인 시 -sysdba 관리자로 서버에 접속하여 서버 구동
- 서버 스크립트 명령으로 서버 구동

Altibase를 구동시키기 위해서는 데이터베이스 생성 시와 마찬가지로 우선 isql을 -sysdba 옵션으로 실행해야 한다.

다음은 iSQL을 -sysdba 옵션으로 실행하는 것을 보여준다.

```
$ isql -u sys -p manager -sysdba
-----
Altibase Client Query utility.
Release Version 7.1.0.0.1
Copyright 2000, Altibase Corporation or its subsidiaries.
All Rights Reserved.
-----
ISQL_CONNECTION = TCP, SERVER = 127.0.0.1, PORT_NO = 20300
iSQL(sysdba)>
```

Note: STARTUP 명령어는 Altibase (isql 포함)를 설치한 계정으로만 수행이 가능하다.

Altibase를 구동하면, Altibase의 상태는 아래의 단계대로 순차적으로 진행된다.

1. PRE_PROCESS
2. PROCESS
3. CONTROL
4. META
5. SERVICE

STARTUP 명령어는 아래의 단계 옵션과 함께 사용할 수 있다.

STARTUP [PROCESS | CONTROL | META | SERVICE];

SERVICE 상태가 되어야 SYS사용자를 제외한 일반 사용자들이 데이터베이스에 접근할 수 있다.

Note: Altibase의 상태는 다음 상태로 진행만 할 수 있으며, 이전 상태로 되돌아갈 수는 없다.

SERVICE 상태로 전이시키는 예는 아래와 같다.

```
iSQL> startup service;
Trying Connect to Altibase..... Connected with Altibase.
TRANSITION TO PHASE: PROCESS
TRANSITION TO PHASE: CONTROL
TRANSITION TO PHASE: META
  [SM] Checking Database Phase:  *.*.*[SUCCESS]
  [SM] Recovery Phase - 1: Preparing Database...[SUCCESS]
  [SM] Recovery Phase - 2: Loading Database : Dynamic Memory Version
                        Serial Bulk Loading
                        . is 8192k: *..[SUCCESS]
  [SM] Recovery Phase - 3: Skipping Recovery & Starting Threads...[SUCCESS]
                        Refining Disk Table [SUCCESS]
  [SM] Garbage Collection: ..... [SUCCESS]
  [SM] Rebuilding Indices [Total Count:61] *****.....
..... [SUCCESS]
TRANSITION TO PHASE: SERVICE
      No IPC Initialize: Disabled
--- STARTUP Process SUCCESS ---
Command execute success.
```

구동의 각 단계에서 사용자가 할 수 있는 일은 다음과 같다.

단계	가능한 작업
PRE-PROCESS	PROCESS 단계로 전이할 수 있다.
PROCESS	CREATE DATABASE구문으로 데이터베이스를 생성하거나 DROP DATABASE 구문으로 데이터베이스를 삭제할 수 있다. 제한된 개수의 성능 뷰들을 조회할 수 있다. 프로퍼티 값들을 변경시킬 수 있다. CONTROL 단계로 전이할 수 있다.
CONTROL	미디어 복구 (Media Recovery)를 수행할 수 있다. META 단계로 전이할 수 있다. CONTROL 단계에서 불완전 복구를 한 경우 META 단계로 전이할 때 온라인 로그를 리셋(reset)해야 한다.

단계	가능한 작업
META	메타 데이터 (Dictionary table)를 업그레이드할 수 있다. SERVICE 단계로 전이할 수 있다.
SERVICE	SYS사용자를 제외한 일반 사용자로부터 접속을 받을 수 있다. SHUTDOWN NORMAL/IMMEDIATE/ABORT 를 수행할 수 있다.

Altibase 종료

현재 구동중인 Altibase 서버를 종료하려면 SHUTDOWN 구문을 사용한다. 아래의 옵션이 가능하다.

```
SHUTDOWN [NORMAL | IMMEDIATE | ABORT];
```

SHUTDOWN NORMAL과 SHUTDOWN IMMEDIATE는 Altibase가 SERVICE 상태일 때만 수행 가능하며, SHUTDOWN ABORT는 어떤 상태에서도 수행 가능하다.

Note: SHUTDOWN 명령어는 Altibase (isql 포함)를 설치한 유닉스 계정으로만 수행이 가능하다.

SHUTDOWN NORMAL

서버를 정상적으로 종료하는 방식이다. 서버는 모든 클라이언트들이 서버로부터 접속을 끊을 때까지 종료 작업을 대기한다. 서버 종료시 내부적으로 다음의 작업이 수행된다.

- 클라이언트-서버간 통신 세션을 감지하는 스레드의 종료
- 서비스 스레드의 종료
- 자료 저장 관리자의 종료
- 마지막으로 Altibase 서버 프로세스가 완전히 종료되면, Altibase 서버를 종료

이 방식으로 Altibase를 종료했을 때 다음과 같은 메시지가 출력된다.

```
iSQL(sysdba)> shutdown normal;
Ok..Shutdown Proceeding....
```

```
TRANSITION TO PHASE : Shutdown Altibase
[RP] Finalization : PASS
shutdown normal success.
```

SHUTDOWN IMMEDIATE

SHUTDOWN IMMEDIATE를 실행하면, Altibase 서버는 먼저 현재 연결된 세션들을 강제로 단절시킨 다음, 현재 실행 중이던 트랜잭션들을 철회(rollback) 시키고 Altibase 서버를 종료한다.

이 방식으로 Altibase를 종료했을 때 다음과 같은 메시지가 출력된다.

```
iSQL(sysdba)> shutdown immediate
Ok..Shutdown Proceeding....

TRANSITION TO PHASE : Shutdown Altibase
  [RP] Finalization : PASS
shutdown immediate success.
```

서버 스크립트 명령을 이용하여 서버를 종료할 수도 있다.

```
$ server stop
-----

Altibase Client Query utility.
Release Version 7.1.0.0.1
Copyright 2000, Altibase Corporation or its subsidiaries.
All Rights Reserved.

-----

ISQL_CONNECTION = TCP, SERVER = 127.0.0.1, PORT_NO = 20300
Alter success.
Alter success.
Alter success.
Ok..Shutdown Proceeding....

TRANSITION TO PHASE : Shutdown Altibase
  [RP] Finalization : PASS
shutdown immediate success.
```

SHUTDOWN ABORT

SHUTDOWN ABORT는 Altibase 서버를 강제로 죽인다. 이 방법으로 Altibase 서버를 종료하면, 데이터베이스가 완전하지 못한 상태가 되어 다음에 Altibase 서버를 구동할 때 데이터베이스 복구 과정을 거쳐야 할 수도 있다.

이 방식으로 Altibase를 종료했을 때 다음과 같은 메시지가 출력된다.

```
iSQL(sysdba)> shutdown abort
iSQL(sysdba)>
```

또는 서버 스크립트 명령을 이용할 수 있다.

```
$ server kill
```

```
-----  
Altibase Client Query utility.  
Release Version 7.1.0.0.1  
Copyright 2000, Altibase Corporation or its subsidiaries.  
All Rights Reserved.  
-----
```

```
ISQL_CONNECTION = TCP, SERVER = 127.0.0.1, PORT_NO = 20300
```

```
$
```

5.데이터베이스 객체 및 권한

이장에서는 Altibase 데이터베이스 내의 객체 관리 및 권한 관리 방법에 대해서 설명한다.

데이터베이스 객체 개요

데이터베이스 객체는 특정 스키마에 속하여 관리되는 스키마 객체와 스키마와 관계없이 데이터베이스 전체에서 관리되는 비스키마 객체로 구분할 수 있다. 이 장에서는 스키마 객체와 비스키마 객체를 구분하고 각각에 포함되는 데이터베이스 객체에 대해 설명한다.

스키마 객체

스키마란 데이터 또는 객체들의 논리적 집합으로 한 데이터베이스 사용자는 하나의 스키마를 소유하고 SQL문에 의해 관리된다. 이러한 스키마에 포함되는 객체를 스키마 객체라고 하고 Altibase는 다음과 같은 스키마 객체를 제공한다.

테이블 (Table)

테이블은 가장 기본적인 데이터 저장 단위로서 테이블은 칼럼들로 구성된 레코드들의 집합이다. Altibase의 테이블은 데이터의 저장 공간에 따라 메모리 테이블과 디스크 테이블로 구별된다. 그리고 시스템이 생성하고 관리하는 시스템 테이블과 일반 사용자가 생성하는 일반 테이블로 구별될 수도 있다.

이중화 대상 테이블의 경우 테이블 관리가 특별하며 대용량 테이블의 경우에도 주의가 요하는 사항들이 있다.

이에 대해서는 아래의 "테이블" 절에서 자세히 설명한다.

파티션드 테이블(Partitioned Table)

테이블의 데이터를 여러 조각(각 조각을 파티션이라고 한다)으로 나누어 서로 다른 테이블스페이스에 저장하는 경우, 이 테이블을 파티션드 테이블(Partitioned Table)이라고 한다. 대용량 테이블의 경우 파티션드 테이블을 활용하면 데이터 관리가 용이할 것이다.

파티션드 테이블에 대한 자세한 내용은 "7장 파티션드 객체"를 참조한다.

파티션드 인덱스(Partitioned Index)

인덱스가 파티션되는 여부에 따라 파티션드 인덱스(partitioned index) 또는 논파티션드 인덱스(non partitioned index)로 분류한다. 논파티션드 인덱스는 파티션으로 분할되지 않은 인덱스를 의미하며, 파티션드 인덱스는 파티션드 테이블과 마찬가지로 파티션 조건에 따라 분리한 인덱스를 의미한다.

파티션드 인덱스에 대한 자세한 내용은 "7장 파티션드 객체"를 참조한다.

임시 테이블 (Temporary Table)

하나의 세션 또는 트랜잭션이 유지되는 동안에 데이터를 일시적으로 보관하기 위해 임시 테이블을 사용할 수 있다. 임시 테이블을 이용하면 복잡한 질의를 사용할 때 수행 속도를 높일 수 있다.

임시 테이블은 휘발성 테이블스페이스에만 생성할 수 있다.

큐 테이블(Queue Table)

Altibase는 메시지 큐잉 기능을 이용하여 데이터베이스와 사용자 프로그램간의 비동기 데이터 통신을 지원한다. 이때 사용되는 큐 테이블은 데이터베이스 객체의 하나로써 일반 테이블과 마찬가지로 DDL과 DML로 제어할 수 있다.

큐 테이블의 개념과 기능에 대해서는 아래의 "큐" 절에서 자세히 설명한다.

제약조건 (Constraint)

제약조건이란 테이블의 데이터 삽입 또는 변경 시 데이터의 일관성을 유지할 수 있도록 부과하는 조건이다.

제약조건의 대상에 따라 칼럼 제약조건과 테이블 제약조건으로 구별할 수 있다. Altibase는 아래의 제약조건을 지원한다.

- NOT NULL / NULL 제약조건
- CHECK 제약조건
- 유일 키 (unique key) 제약조건
- 주 키 (primary key) 제약조건
- 외래 키 (foreign key) 제약조건

- TIMESTAMP 제약조건

인덱스 (Index)

인덱스는 테이블의 특정 칼럼들에 대해 색인을 생성하여 그 테이블의 레코드들에 대한 빠른 접근이 가능하도록 한다. 즉, 인덱스를 사용하여 DML문의 처리 성능을 향상시킬 수 있다.

뷰 (View)

뷰는 실제 데이터 자체는 포함하지 않고, 하나 이상의 테이블, materialized view 또는 뷰를 기반으로 한 논리적 테이블 (logical table)이다.

Materialized View

Materialized view란 쿼리의 결과를 데이터로 저장하고 있는 데이터베이스 객체이다. 하나 이상의 테이블, 뷰, 및 다른 materialized view에 기반하여 데이터를 구성할 수 있다.

시퀀스 (Sequence)

Altibase는 유일 키를 생성하기 위한 키생성자인 시퀀스를 제공한다.

시노님 (Synonym)

테이블, 시퀀스, 뷰, 저장 프로시저 및 저장 함수에 대한 별칭 (alias)을 부여하여 객체 사용에 대한 투명성을 보장할 수 있는 시노님을 제공한다.

저장 프로시저 및 저장 함수 (Stored Procedure or Function)

저장 프로시저 (Stored Procedure)란 SQL문들과 흐름 제어문, 할당문, 오류 처리 루틴 등으로 구성된 데이터베이스 객체이다. 이를 이용해서 전체 업무 절차를 프로그래밍하여 하나의 모듈로 만든 후 데이터베이스에 영구적으로 저장해 두고, 모듈 이름만을 호출하여 전체 업무 절차를 서버에서 한번에 수행하도록 하는 데이터베이스 객체이다.

하나의 리턴 값을 가지지 않느냐와 가지느냐에 따라 저장 프로시저와 저장 함수로 구별된다.

타입 세트 (Type Set)

타입 세트 (Type Set)란 저장 프로시저 및 저장 함수에서 사용하는 사용자 정의 타입들을 한 곳에 모아서 관리하도록 해 주는 데이터베이스 객체이다.

이에 대한 보다 자세한 내용은 *Stored Procedures Manual*에서 자세히 설명한다.

데이터베이스 트리거 (Database Trigger)

트리거란 테이블에 데이터가 삽입, 삭제, 또는 갱신될 때 시스템에 의해 작동되어 특정 작업 절차를 자동으로 수행할 수 있도록 하는 저장 프로시저의 한 종류이다. 사용자는 테이블에 대해 제약조건과 트리거를 정의하여 데이터의 일관성을 유지할 수 있다.

데이터베이스 링크 (Database Link)

데이터베이스 링크는 지역적으로 분리되어 있으나 네트워크로 연결된 데이터 서버들을 연동하여 그 데이터들을 통합해서 하나의 결과를 생성할 수 있게 한다.

이에 대해서는 *Database Link User's Manual*에 더 자세히 기술되어 있다.

외부 프로시저 및 외부 함수 (External Procedure or Function)

외부 프로시저 또는 외부 함수 객체는 사용자 정의 C/C++ 함수와 일대일로 대응하는 데이터베이스 객체이다. 사용자 정의 함수의 실행은 외부 프로시저 또는 외부 함수 객체를 통해 이루어진다. 리턴 값을 가지는 여부에 따라 외부 프로시저와 외부 함수로 구별된다.

자세한 내용은 *C/C++ External Procedures Manual*을 참고하도록 한다.

라이브러리 (Library)

외부 프로시저와 연결된 사용자 정의 C/C++ 함수를 포함하는 동적 라이브러리 파일을 Altibase 서버가 식별할 수 있도록 해야 한다. 이를 위해 Altibase는 동적 라이브러리 파일에 일대일로 대응하는 라이브러리 객체라는 데이터베이스 객체를 제공한다.

자세한 내용은 *C/C++ External Procedures Manual*을 참고하도록 한다.

비스키마 객체

특정 스키마에 소속되지 않고 전체 데이터베이스 수준에서 관리되는 객체를 비스키마 객체라고 한다. Altibase는 다음과 같은 비스키마 객체를 제공한다.

디렉토리 (Directory)

저장프로시저의 파일 제어 기능은 운영 체제의 텍스트 파일에 대한 읽기 및 쓰기 기능을 제공한다. 이 기능을 이용하여 사용자는 저장프로시저 실행에 대한 별도의 메시지 등을 파일에 남길 수도 있으며, 파일로 결과를 보고하거나 파일로부터 데이터를 읽어와 테이블에 삽입하는 등 다양한 작업을 수행할 수 있다. 디렉토리 객체는 이러한 저장프로시저에서 접근하는 파일들이 저장되어 있는 디렉토리에 대한 정보를 관리하는데 사용된다.

디렉토리 객체에 대한 자세한 기능은 *SQL Reference*를 참고한다.

저장프로시저 내에서의 파일 제어 방법은 *Stored Procedures Manual*을 참고한다.

이중화 (Replication)

이중화는 시스템이 자동으로 한 지역서버에서 원격 서버로 데이터를 전송하고 복제하여 다른 서버들간의 테이블 데이터를 동일하게 유지해 줄 수 있도록 하는 객체이다.

이중화 관리에 대해서는 *Replication Manual*을 참조한다.

테이블스페이스 (Tablespace)

테이블스페이스는 가장 큰 논리적 데이터 저장 단위로 데이터베이스는 여러 개의 테이블스페이스로 나뉘어져 관리된다.

테이블스페이스는 저장공간을 기준으로 크게 메모리 테이블스페이스와 디스크 테이블스페이스로 구분된다. 시스템 테이블스페이스는 데이터베이스 생성시에 자동으로 만들어지며 사용자가 삭제할 수 없다. 또 사용자는 필요에 따라 사용자 테이블스페이스를 자유롭게 생성하거나 삭제할 수 있다.

테이블스페이스 관리에 대한 자세한 내용은 “6장 테이블스페이스”를 참조한다.

사용자 (User)

사용자 계정은 Altibase 접속을 위해 필요하며, 스키마의 소유자이기도 하다. 시스템에 의해 생성되어 전체 시스템의 관리자인 시스템 사용자와 일반 사용자로 구분된다. 일반 사용자의 경우 데이터베이스에 접근하여 데이터를 조작하기 위해서는 적절한 권한이 필요하다.

사용자 권한에 대해서는 아래의 "권한" 절에서 자세히 설명한다.

작업 (Job)

작업(Job)은 저장 프로시저에 실행 일정을 더한 것이다. JOB 객체를 생성할 때 실행할 저장 프로시저와 실행 시각, 실행 반복 간격 등의 일정을 설정할 수 있다. 생성된 JOB이 자동으로 돌아가도록 하기 위해서는 JOB_SCHEDULER_ENABLE 프로퍼티를 1로 설정해야 한다. JOB의 생성, 변경 및 삭제와 작업 스케줄러에 대한 관리는 SYS 사용자에게 의해서만 가능하다.

이에 대해서는 아래의 "작업(Job)" 절에서 자세히 설명한다.

테이블

테이블은 가장 기본적인 데이터 저장 단위이다. 테이블은 칼럼들로 구성되며 레코드들을 포함한다. 이 절에서는 테이블과 관련된 용어를 정의하고 테이블 관리

개념과 방법들에 대해 설명한다.

메모리 테이블과 디스크 테이블

테이블의 저장 공간에 따라 메모리 테이블과 디스크 테이블로 분류된다. 테이블 생성 시 그 테이블이 속한 테이블스페이스가 메모리 테이블스페이스인지 디스크 테이블스페이스인지에 따라서 메모리 테이블 또는 디스크 테이블이 된다.

시스템 테이블과 사용자 테이블

또한 테이블은 시스템이 내부적으로 생성하고 관리하는 시스템 테이블과 사용자에게 의해 생성되고 관리되는 사용자 테이블로 분류된다.

데이터 디렉터리로 알려진 시스템 테이블은, 데이터베이스 객체 정보를 저장하는 메타 테이블과 시스템 프로세스 정보를 저장하는 프로세스 테이블로 분류된다. 프로세스 테이블은 다시 고정 테이블 (fixed table)과 성능 뷰 (performance view)로 분류된다.

메타 테이블과 성능 뷰에 대해서는 *General Reference*를 참고한다.

대용량 메모리 테이블

대용량 메모리 테이블에 대해 SQL문 수행 시 다음과 같은 주의를 요한다.

대용량 메모리 테이블에 DDL 수행하기

대용량 테이블에 DDL 문을 수행하고자 하는 경우, 그 테이블에 ADD COLUMN 또는 DROP COLUMN을 직접 수행하는 것 보다 iLoader 유틸리티를 이용해서 데이터를 다운로드 받고 그 테이블은 삭제하고 새로운 스키마로 그 테이블을 재 생성한 후에 다운로드 받은 데이터를 iLoader 유틸리티를 이용해서 업로드하는 방식을 권장한다.

대용량 메모리 테이블에 DML 수행하기

데이터 크기가 크지 않은 테이블에 대해 DML을 수행하는 것은 운영자가 잘못된 데이터 조작을 하지 않는 한 Altibase의 성능이나 운용 상에 있어서 크게 문제를 발생시키지 않는다. 그러나 하나의 UPDATE 또는 DELETE 문 수행이 아주 많은 수의 레코드에 영향을 미친다면, 그 DML을 수행하는 트랜잭션은 오랜 시간 동안 진행 중인 상태로 있을 수 있다. 이처럼 오랜 시간 동안 진행 중인 트랜잭션 (long-duration transaction)이 존재하는 경우 Altibase를 운용함에 있어 다음과 같은 심각한 문제를 야기할 수 있다.

테이블에 대한 배타적 접근

트랜잭션의 수행 시간이 길어지면 그 트랜잭션이 획득하고 있는 잠금 (lock) 때문에 그 테이블에 접근하고자 하는 다른 트랜잭션들은 모두 수행을 멈추게 된다. 또한

변경되는 레코드들의 전체 양이 LOCK_ESCALATION_MEMORY_SIZE 프로퍼티에 지정한 값 이상이 되면 lock escalation이 발생하므로, 검색만 하는 트랜잭션이라 하더라도 그 테이블에 대해 접근할 수 없는 경우가 발생할 수도 있다.

Altibase 메모리 사용량 증가

Altibase에서 사용되는 모든 레코드 (버전)에는 garbage collector가 삭제해야 할 레코드들을 구분하기 위해 SCN (System Commit Number)이 사용된다. 커밋되지 않은 트랜잭션이 사용하고 있는 SCN보다 작은 SCN을 가지는 레코드에 대해서만 garbage collector는 삭제 작업을 수행한다. 그러므로, 장기간 수행 중인 트랜잭션이 존재하면 garbage collector는 자신이 처리해야 할 레코드가 없는 것으로 간주하고 레코드 삭제 작업을 수행하지 않는다.

따라서, bulk update/delete를 장기간 수행하는 트랜잭션이 존재하는 경우 garbage collector의 동작이 멈추게 되어 불필요한 버전이 계속 쌓이는 현상이 발생하며, 이에 따라 Altibase 메모리 사용량뿐만 아니라 데이터베이스의 크기도 증가하게 된다.

로그 파일의 축적

트랜잭션이 생성하는 로그 파일들은 이중화 또는 재시작 복구를 위해 필요한 로그들을 제외하고는 체크포인트 수행시 디스크에서 삭제된다. 재시작 복구에 필요한 로그 파일은 체크포인트 검사 시 수행 중이던 트랜잭션들이 생성한 로그 파일들 중 가장 오래된 로그 파일을 의미한다.

따라서, 장시간 수행 중인 트랜잭션이 존재하면 체크포인트가 발생하더라도 재시작 복구를 위해 로그 파일들은 지워지지 않는다. 따라서 로그 파일이 저장되는 파일 시스템에 더 이상 로그를 저장하지 못하는 상태가 될 수 있다.

페이지 리스트 다중화

메모리 테이블의 경우 여러 개의 트랜잭션이 동시에 수행될 때 하나의 페이지 리스트에서 페이지를 할당받는 과정에서 병목 현상이 발생할 수 있다. 이러한 병목을 제거하기 위하여 테이블에서 사용하는 페이지 리스트를 다중화할 수 있다.

이중화된 테이블

Altibase에서 이중화 대상인 테이블에 대하여 DDL 문의 실행이 가능하지만, 먼저 반드시 다음과 같이 프로퍼티를 설정해야 한다.

- REPLICATION_DDL_ENABLE 프로퍼티를 1로 설정한다.
- ALTER SESSION SET REPLICATION으로 설정할 수 있는 REPLICATION 세션 프로퍼티를 NONE 이외의 값으로 설정한다.

이중화 테이블 관리에 대한 자세한 내용은 *Replication Manual*을 참조한다.

생성

테이블은 CREATE TABLE문을 사용하여 생성할 수 있다.

테이블 생성 시에는 칼럼 정의, 제약조건, 테이블이 저장될 테이블스페이스, 테이블에 삽입할 수 있는 최대 레코드 수, 테이블을 위한 저장 관리자 내 페이지에 대한 공간 활용율, 테이블이나 파티션에 대한 접근 모드 등을 명시할 수 있다.

예제

```
CREATE TABLE book(  
  isbn    CHAR(10) CONSTRAINT const1 PRIMARY,  
  title    VARCHAR(50),  
  author   VARCHAR(30),  
  edition  INTEGER DEFAULT 1,  
  publishingyear INTEGER,  
  price    NUMBER(10,2),  
  pubcode  CHAR(4)) MAXROWS 2 TABLESPACE user_data;
```

```
CREATE TABLE dept_c002  
  AS SELECT * FROM employees  
  WHERE dno = 4002;
```

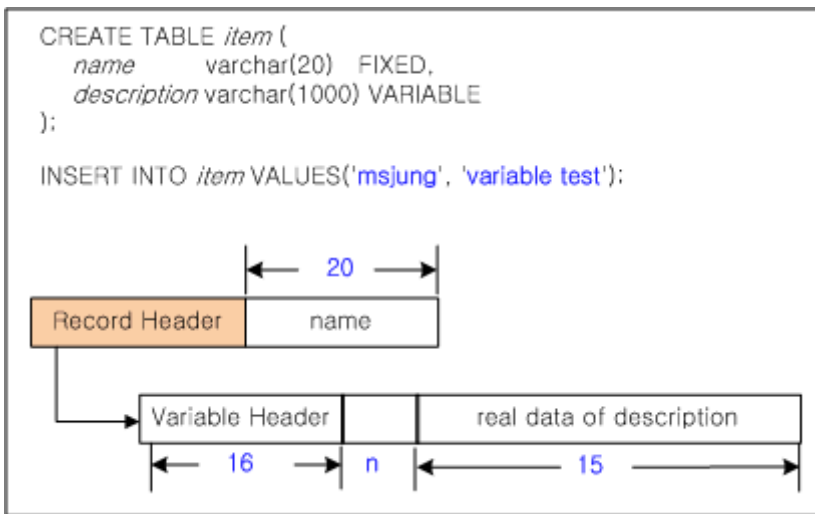
메모리 테이블에서 칼럼 정의 시 주의 사항

VARCHAR 데이터 타입의 경우 FIXED 또는 VARIABLE 속성을 사용자가 지정할 수 있다.

사용자가 이 속성을 지정하지 않았다면, 데이터의 길이가

MEMORY_VARIABLE_COLUMN_IN_ROW_SIZE 프로퍼티에 지정한 값 이하인 데이터의 경우 FIXED영역에 저장되고 그렇지 않을 경우 VARIABLE 영역에 저장된다. FIXED영역에 저장될 경우에는 비록 데이터 타입이 VARCHAR일지라도 CHAR 데이터 타입처럼 저장공간을 해당 길이만큼 미리 할당받으며, VARIABLE영역에 저장될 경우에는 데이터 길이만큼만 저장공간에 할당된다. VARCHAR 데이터 타입의 데이터 비교 방식은 FIXED 또는 VARIABLE 속성에 관계 없이 non-blank padding 비교 방식을 따른다.

다음 그림은 FIXED 또는 VARIABLE로 선언된 칼럼이 레코드 내에서 저장되는 방식을 보여준다. FIXED일 경우에는 비록 데이터 타입이 VARCHAR일지라도 CHAR 데이터 타입처럼 메모리상에 미리 해당 길이만큼 할당받으며 VARIABLE일 경우에는 데이터 길이만큼만 메모리 상에 할당된다.



[그림 5-1] VARCHAR 칼럼 구조

테이블 item의 칼럼 name은 VARCHAR(20) FIXED로 선언되었기 때문에 실제 삽입되는 값인 “msjung”의 길이가 6이라 하더라도 레코드 내에서 20바이트만큼 그 공간을 할당받는다.

테이블 item의 칼럼 description은 VARCHAR(1000) VARIABLE로 선언되었기 때문에 실제 삽입되는 값인 “variable test”의 길이인 13바이트만큼 그 공간을 할당받는다. 할당받는 공간은 레코드 내의 연속적인 공간이 아니라 위 그림처럼 별도의 공간¹이다.

[¹] VARCHAR 데이터 타입의 VARIABLE속성으로 선언된 칼럼에 데이터를 저장할

때마다 데이터 크기만큼 매번 메모리 할당을 받게 되는 경우 성능에 영향을 줄 수 있다. 그러므로 Altibase는 4K, 8K, 16K 등 내부적으로 정해진 길이의 슬롯을 미리 준비하며, VARCHAR 데이터 타입의 VARIABLE칼럼에 데이터 입력 시 서버는 실제 데이터를 저장할 수 있는 최적의 크기를 갖는 슬롯을 선택하여 데이터를 저장한다.

VARCHAR 데이터 타입의 VARIABLE속성으로 선언된 칼럼은 실제 데이터가 저장된 곳의 위치 정보를 record header에 유지한다. 별도로 저장된 슬롯마다 16 바이트의 variable header와 n개 칼럼의 위치를 저장하기 위한 (n+1)*2의 추가 공간이 필요하다. 따라서 위 그림의 예제에서 description 칼럼의 값을 저장하기 위해 실제 사용되는 공간은 35바이트이다.

변경

ALTER TABLE문, RENAME문을 사용하여 다음과 같은 테이블 정의를 변경할 수 있다.

- 테이블 이름
- 새로운 칼럼 추가

- 기존 칼럼 삭제
- 칼럼 기본 값
- 칼럼 이름
- 제약 조건 추가
- 제약 조건 삭제
- 메모리 테이블 COMPACT
- 최대 허용 레코드 수
- 인덱스 활성화 및 비활성
- 테이블 또는 파티션 접근 모드

예제

```
ALTER TABLE book
  ADD COLUMN (isbn CHAR(10) PRIMARY KEY,
  edition INTEGER DEFAULT 1);
```

```
ALTER TABLE book
  DROP COLUMN isbn;
```

```
ALTER TABLE department
  RENAME COLUMN dno TO dcode;
```

ALTER TABLE 문에 대한 자세한 설명은 *SQL Reference*를 참고한다.

삭제

테이블은 DROP TABLE문을 사용하여 삭제할 수 있다.

예제

```
DROP TABLE employees;
```

TRUNCATE

테이블의 레코드는 DELETE문을 사용해 삭제할 수도 있지만 TRUNCATE TABLE문을 이용해 삭제할 수도 있다. DELETE문은 내부적으로 레코드를 건건이 삭제하는 것인 반면, TRUNCATE TABLE문은 내부적으로 DROP TABLE문을 수행하고 같은 정의의 테이블을 재생성하는 DDL문이다.

따라서 TRUNCATE TABLE문을 수행하면 그 테이블에 대해 테이블 수준의 잠금 (lock)이 잡히며, TRUNCATE TABLE문이 성공적으로 수행된 이후에는 ROLLBACK문을 사용해도 삭제된 데이터를 복구할 수 없다.

데이터 조작

테이블의 레코드들은 다음의 DML문을 사용하여 조작할 수 있다.

- INSERT
- DELETE
- UPDATE
- SELECT

위에서 언급한 바와 같이 Altibase 운용 중에 대용량의 데이터에 대해 bulk UPDATE/DELETE를 수행하는 것은 위험하기 때문에, Altibase CLI 혹은 전처리기 (APRE C/C++)를 이용하여 응용프로그램 작성 시 각 레코드에 대해 UPDATE/DELETE 작업 수행 후 커밋하는 것이 바람직하다.

다음은 bulk UPDATE/DELETE을 피하고 레코드 각각에 대해 UPDATE작업을 수행하는 C/C++ Precompiler 프로그램의 예이다.

<p>(a) isql 상에서 bulk-update 수행</p> <pre>iSQL >update t1 set col1=2 where col1 > 1000;</pre>	<p>(b) APRE C/C++를 이용하여 레코드별 update 수행</p> <pre>..... EXEC SQL DECLARE update_cursor CURSOR FOR select col1 from t1 where col1 > 1000; EXEC SQL OPEN update_cursor; while (1) { EXEC SQL FETCH update_cursor INTO :t1_col; if (sqlca.sqlcode == SQL_NO_DATA) break; EXEC SQL update t1 set col1=2 where col1=:t1_col; }</pre>
---	--

관련 SQL문

테이블에 대해 다음과 같은 SQL문이 지원된다. 이에 대한 자세한 설명은 *SQL Reference*를 참조한다.

- CREATE TABLE
- ALTER TABLE
- RENAME TABLE

- TRUNCATE TABLE
- LOCK TABLE
- INSERT
- DELETE
- UPDATE
- SELECT

임시 테이블

하나의 세션 또는 트랜잭션이 유지되는 동안에 데이터를 일시적으로 보관하기 위해 임시 테이블을 사용할 수 있다. 임시 테이블(Temporary Table)을 이용하면 복잡한 질의를 사용할 때 수행 속도를 높일 수 있다. 따라서 임시 테이블은 응용 프로그램에서 여러 개의 DML 작업을 실행할 때 생기는 결과 집합을 일시적으로 저장할 때 유용하다.

임시 테이블은 휘발성 테이블스페이스에 생성할 수 있으며, 세션이나 트랜잭션이 종료되면 임시 테이블은 자동으로 truncate된다.

임시 테이블의 정의는 모든 세션에서 볼 수 있지만, 임시 테이블의 데이터는 테이블에 데이터를 입력한 세션에서만 볼 수 있다.

임시 테이블에도 인덱스를 생성할 수 있다. 인덱스 또한 임시적이며 인덱스의 데이터는 인덱스가 생성된 테이블의 데이터와 동일한 세션 또는 트랜잭션 레벨에서 유효하다.

일반 테이블과 달리 임시 테이블과 거기에 생성된 인덱스는 객체 생성 시에 자동으로 세그먼트가 할당되지 않는다. 대신 처음으로 INSERT(또는 CREATE TABLE AS SELECT)가 수행될 때 세그먼트가 할당된다.

트랜잭션에 한정되는 임시 테이블은 동시에 하나의 트랜잭션만 허용한다.

임시 테이블의 데이터는 일시적이기 때문에, 임시 테이블의 데이터는 백업이나 시스템 장애 시 복구가 불가능하다. 따라서 사용자는 장애에 대비하기 위하여 임시 테이블의 데이터를 보존할 수 있는 대안을 직접 개발해야 한다.

제약 사항:

- 임시 테이블은 휘발성 테이블스페이스에만 생성할 수 있다.

생성

임시 테이블 생성은 CREATE [GLOBAL] TEMPORARY TABLE 구문을 사용한다. ON COMMIT 절은 테이블의 데이터가 트랜잭션에 한정되는지 또는 세션에 한정되는지를 지정한다.

ON COMMIT 절에 대한 자세한 설명은 *SQL Reference*를 참고하기 바란다.

또한 임시 테이블은 휘발성 테이블스페이스에만 생성이 가능하므로, TABLESPACE 절에 임시 테이블이 생성될 휘발성 테이블스페이스를 지정해야 한다.

예제

<질의> 트랜잭션에 한정되는 임시 테이블을 생성한다:

```
CREATE VOLATILE TABLESPACE my_vol_tbs SIZE 12M AUTOEXTEND ON MAXSIZE 1G;

CREATE TEMPORARY TABLE temp1(i1 INTEGER, i2 VARCHAR(10))
    ON COMMIT DELETE ROWS
    TABLESPACE my_vol_tbs;
```

변경

세션에 한정되는 임시 테이블은 세션에 바인딩 되지 않은 경우에만 DDL 작업(ALTER TABLE, DROP TABLE, CREATE INDEX 등)이 허용된다.

트랜잭션에 한정되는 임시 테이블은 바인딩 여부에 상관 없이 DDL 작업이 허용된다.

하지만 Altibase는 DDL 작업을 수행하기 전에 내부적으로 커밋을 먼저 하기 때문에, 임시 테이블에 DDL 작업을 수행하면 그 테이블의 데이터는 사라진다.

예제

<질의> 임시 테이블이 세션에 바인딩 되어 있는 상태에서 DDL 작업이 실패하는 것을 보여준다.

```
CREATE VOLATILE TABLESPACE my_vol_tbs SIZE 12M AUTOEXTEND ON MAXSIZE 1G;
CREATE TEMPORARY TABLE temp1(i1 INTEGER, i2 VARCHAR(10))
    ON COMMIT PRESERVE ROWS
    TABLESPACE my_vol_tbs;
INSERT INTO temp1 VALUES (1, 'ABC');
```

```
iSQL> ALTER TABLE temp1 ADD CONSTRAINT temp1_pk PRIMARY KEY (i1);
[ERR-31363 : Cannot execute DDL when a temporary table is in use.]
```

삭제

임시 테이블은 DROP TABLE 문을 사용하여 삭제할 수 있다.

예제

<질의> 임시 테이블 temp1을 삭제한다.

```
DROP TABLE temp1;
```

데이터 조작

일반 테이블과 동일하게 INSERT, UPDATE 또는 DELETE 구문을 사용해서 임시 테이블의 데이터를 조작할 수 있다.

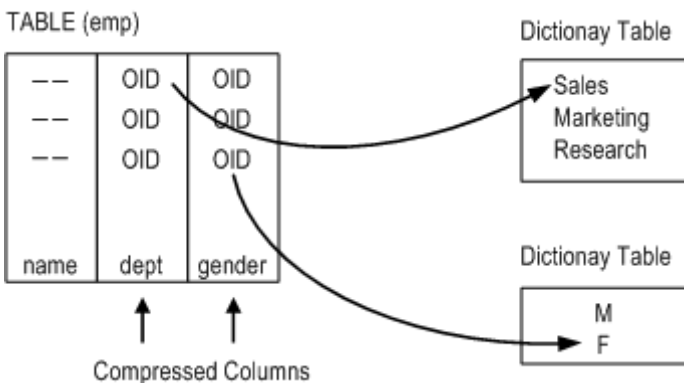
관련 SQL문

임시 테이블에 대해 다음과 같은 SQL문이 지원된다. 이에 대한 자세한 설명은 *SQL Reference*를 참조한다.

- CREATE TABLE
- ALTER TABLE
- RENAME TABLE
- TRUNCATE TABLE
- LOCK TABLE
- INSERT
- DELETE
- UPDATE
- SELECT

압축 테이블

압축 테이블이란, 압축 칼럼이 속해 있는 테이블을 의미한다. 사용자가 압축 칼럼을 포함하는 테이블을 생성하면, Altibase 서버는 딕셔너리 테이블과 빠른 검색을 위한 유니크 인덱스를 자동으로 생성한다. 딕셔너리 테이블은 데이터가 실제로 저장되는 테이블을 가리키며, 압축 칼럼마다 하나의 딕셔너리 테이블이 생성된다. 압축 칼럼에 데이터가 입력되거나 변경될 때 실제 데이터는 딕셔너리 테이블에 삽입되며, 압축 칼럼에는 실제 데이터가 저장된 위치를 가리키는 값(아래 그림에서 OID)이 저장된다.



[그림 5-2] 압축 칼럼과 딕셔너리 테이블 간의 관계

압축 테이블은 칼럼 값이 중복되지 않도록 별도의 테이블에 데이터를 저장하여 관리함으로써 메모리 사용량을 줄일 수 있다. 하지만, 중복되는 칼럼 값이 적을 경우 압축을 위한 부가적인 데이터 저장 공간만 차지하여 오히려 메모리 사용량이 늘어날 수 있으므로 주의가 필요하다.

압축 테이블이 메모리 테이블이든 디스크 테이블이든 상관없이 딕셔너리 테이블은 메모리 테이블스페이스에 생성된다.

제약 사항

- 압축 테이블은 메모리 테이블스페이스와 디스크 테이블스페이스에만 생성할 수 있다.
- 디스크 압축 테이블의 경우, 압축 칼럼에 OID가 저장된 후에는 해당 트랜잭션을 롤백하더라도 딕셔너리 테이블에 삽입된 데이터는 삭제되지 않고 그대로 유지된다.

생성

압축 테이블 생성은 일반 테이블과 동일하게 CREATE TABLE 구문을 사용한다. 다만 COMPRESS 절에 압축할 칼럼의 이름을 명시한다. COMPRESS 절에 대한 자세한 설명은 *SQL Reference*를 참고하기 바란다.

예제

<질의> department와 position 칼럼을 압축하여 emp 압축 테이블을 생성한다.

```
CREATE TABLE emp (  
    name          CHAR( 20 ),  
    department    CHAR( 20 ),  
    position      CHAR( 20 )  
) COMPRESS ( department, position );
```

변경

일반 테이블과 동일하게 ALTER TABLE문, RENAME문을 사용하여 테이블 정의를 변경할 수 있다. 그리고 COMPRESS 절을 사용하여 압축 칼럼을 추가할 수 있다.

예제

<질의> emp 테이블에 salary와 join_date 칼럼을 추가하되, join_date 칼럼을 압축 칼럼으로 추가하라.


```
ALTER TABLE emp
  ADD COLUMN (salary NUMBER, join_date DATE)
  COMPRESS (join_date);
```

재구축

ALTER TABLE *table_name* REORGANIZE 구문은 딕셔너리 테이블에서 참조되지 않는 데이터를 제거하여 저장 공간을 정리한다.

압축 테이블에 DELETE 또는 UPDATE 문을 수행하여도 연결된 딕셔너리 테이블의 데이터는 삭제되거나 변경되지 않고, 새로운 데이터만 삽입된다. 따라서 압축 테이블에 DELETE 또는 UPDATE 문을 빈번히 수행하면 딕셔너리 테이블에 참조되지 않는 데이터가 쌓이게 된다. 따라서 불필요한 데이터가 메모리 저장 공간을 차지하지 않도록 재구축하는 것이다.

예제

<질의> 압축 테이블 emp에 데이터를 삽입하고 삭제한 후, 테이블을 재구축하는 예제이다.

```
CREATE TABLE emp (
  name          CHAR( 20 ),
  department    CHAR( 20 )
) COMPRESS ( department );
INSERT INTO emp VALUES ( 'Park', 'Personel' );
INSERT INTO emp VALUES ( 'Yun', 'Sales' );
INSERT INTO emp VALUES ( 'Min', 'Personel' );
DELETE FROM emp WHERE name = 'Yun';

ALTER TABLE emp REORGANIZE COLUMN ( department );
```

삭제

압축 테이블은 DROP TABLE 문을 사용하여 삭제할 수 있다.

예제

<질의> 압축 테이블 temp1을 삭제한다.

```
DROP TABLE temp1;
```

데이터 조작

일반 테이블과 동일하게 INSERT, UPDATE 또는 DELETE 구문을 사용해서 압축 테이블의 데이터를 조작할 수 있다.

관련 SQL문

압축 테이블에 대해 다음과 같은 SQL문이 지원된다. 이에 대한 자세한 설명은 *SQL Reference*를 참조한다.

- CREATE TABLE
- ALTER TABLE
- RENAME TABLE
- TRUNCATE TABLE
- LOCK TABLE
- INSERT
- DELETE
- UPDATE
- SELECT

큐

Altibase 메시지 큐잉 기능은 데이터베이스와 사용자 프로그램간의 비동기 데이터 통신을 지원한다. 이 때 사용되는 큐 테이블은 데이터베이스 오브젝트의 하나로써 다른 데이터베이스 테이블과 마찬가지로 DDL과 DML구문으로 제어할 수 있다.

생성

사용자가 CREATE QUEUE 문장을 이용해서 큐를 생성하면, 데이터베이스에는 사용자가 명시한 이름의 테이블이 생성된다. 이를 큐 테이블이라고 부른다. 큐 테이블은 다음과 같은 구조를 가진다.

Column name	Type	Length	Default	Description
MSGID	BIGINT	8	-	메시지 식별자로 Altibase에 의해 자동으로 부여됨
CORRID	INTEGER	4	0	사용자가 지정한 메시지 식별자
MESSAGE	VARCHAR	Message length	-	메시지 텍스트
ENQUEUE_TIME	DATE	8	SYSDATE	메시지가 큐에 들어온 시간

큐 테이블의 이름이나 칼럼명은 사용자가 임의로 변경할 수 없으며, MSGID 칼럼에는 자동으로 주요 키 (Primary Key)가 생성된다.

유일한 값의 MSGID를 생성하기 Altibase 내부적으로 [QUEUE이름]_NEXT_MSG_ID라는 이름의 시퀀스가 생성된다. 사용자가 해당 시퀀스에 대한 정보를 조회하려면 SYS_TABLES_ 메타 테이블을 사용하면 된다.

시퀀스는 큐 테이블이 삭제될 때까지 유지되어야 하기 때문에 DROP SEQUENCE 문장으로 삭제되지 않는다.

큐 테이블은 SYS_TABLES_ 메타 테이블에 TABLE_TYPE이 'Q'로 저장된다. 사용자는 필요에 따라서 CREATE INDEX 문장을 이용해서 큐 테이블에 인덱스를 생성할 수 있다.

예제

```
CREATE QUEUE Q1(40);
```

변경

CREATE QUEUE 문장으로 생성된 큐 테이블은 ALTER TABLE 등의 문장으로 구조를 변경할 수 없다. 오직 DROP QUEUE 문장으로 삭제될 수만 있다. 단, 사용자는 ENQUEUE/DEQUEUE, DELETE, SELECT 등의 문장으로 데이터 조작은 가능하다.

삭제

큐 테이블은 DROP QUEUE 문을 사용하여 삭제할 수 있다.

예제

```
DROP QUEUE Q1;
```

데이터 삭제

큐에 적재된 메시지만 모두 삭제하고자 하는 경우에는 TRUNCATE TABLE 문장을 이용할 수 있다.

예제

```
TRUNCATE TABLE Q1;
```

데이터 조작

큐 테이블의 레코드들은 다음과 같은 DML문을 사용하여 조작될 수 있다.

- ENQUEUE
- DEQUEUE
- DELETE
- SELECT

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference* 을 참조한다.

- CREATE QUEUE
- DROP QUEUE
- ENQUEUE
- DEQUEUE

제약조건

제약 조건은 테이블의 데이터 삽입 또는 변경에 제한을 두는 것이다. 이 절에서는 제약조건의 종류와 데이터 일관성을 유지할 수 있도록 제약조건을 관리하는 방법에 대해 설명한다.

종류

Altibase는 다음과 같은 종류의 제약조건을 지원한다.

NOT NULL/NULL

NOT NULL은 칼럼에 NULL이 삽입되는 것을 막는 제약조건이다. NOT NULL은 칼럼 단위로 정의할 수 있다. NULL을 명시하면 NULL값을 허용한다. 칼럼에 대해 NOT NULL 을 명시하지 않으면 기본적으로 NULL값을 허용한다.

CHECK 제약조건

사용자는 CHECK 제약조건을 명시하여 데이터가 무결성 규칙을 갖도록 할 수 있다. CHECK 제약조건은 하나 또는 두 개 이상의 칼럼에 대해 결과가 참, 거짓, 또는 알 수 없음(NULL)이 될 수 있는 조건을 명시하는 것이다. DML 구문 수행으로 변경되는 칼럼 값이 CHECK 제약조건의 조건 검사에서 거짓으로 평가되면, 그 구문은 실패로 처리된다.

CHECK 제약조건의 검사조건에는 아래와 같은 몇 가지 제한 사항이 있다:

- SYNONYM, 부질의(subquery), 시퀀스, LEVEL 또는 ROWNUM 등의 모든 의사칼럼(Pseudo Column), 및 SYSDATE 또는 USER_ID 같은

비결정적(Non-deterministic) SQL 함수가 포함될 수 없다.

- PRIOR 연산자를 사용할 수 없다.
- LOB 타입의 데이터를 사용할 수 없다.

한 칼럼에 여러 개의 CHECK 제약조건을 정의할 수 있다. 단, CHECK 제약조건을 평가 순서는 사용자가 지정할 수 없다. Altibase는 CHECK 조건의 상호 배타적 여부를 확인하지 않으므로, 각 조건이 서로 충돌하지 않게 설계해야 한다.

CHECK 조건에는 아래와 같이 년도 또는 월을 기술하지 않은 DATE 상수가 허용되므로, 사용에 주의가 필요하다:

- 년도를 기술하지 않으면 현재 년도로 설정
- 월을 기술하지 않으면 현재 월로 설정

UNIQUE

하나 이상의 칼럼에 대해 정의할 수 있는 제약조건으로 칼럼 또는 칼럼의 조합에 대해 중복 값이 삽입되는 것을 방지한다. 유일 키 제약조건을 정의하면 내부적으로 유일 키 인덱스가 생성된다.

PRIMARY KEY

프라이머리 키 제약조건은 유일 키 제약조건에 NOT NULL 제약조건까지 합쳐진 제약조건이다. 하나 이상의 칼럼에 대해 프라이머리 키 제약조건을 정의할 수 있다. 프라이머리 키가 생성될 때 내부적으로 유일 키 인덱스가 생성된다. 프라이머리 키에 포함되는 어떤 칼럼에도 NULL값을 삽입할 수 없다.

FOREIGN KEY

참조 무결성 제약조건 (referential integrity constraint)으로 참조 관계에 있는 테이블 간의 데이터 일관성을 유지할 수 있도록 해 주는 제약조건이다.

TIMESTAMP

칼럼의 값에 레코드의 삽입 또는 갱신 시 시스템 시간 값을 설정하는 제약조건이다. 주로 이중화 대상 테이블의 한 칼럼에 대해 이 제약조건을 정의한다.

칼럼 제약조건과 테이블 제약조건

칼럼 정의 시 하나의 칼럼에 대해 정의한 제약조건을 칼럼 제약조건이라 하고 여러 개의 칼럼들에 대해 하나의 제약조건을 테이블 정의 하단 부분에 정의하는 것을 테이블 제약조건이라고 한다.

NULL/NOT NULL 제약조건과 TIMESTAMP 제약조건은 칼럼 제약조건으로만 정의할 수 있고, 그 외 제약조건들은 칼럼 제약조건 또는 테이블 제약조건으로 정의할 수 있다.

생성

제약조건은 테이블 생성 (CREATE TABLE문) 또는 테이블 변경 (ALTER TABLE문) 시 정의할 수 있다.

제약조건 정의 시 제약조건의 이름을 사용자가 명시할 수 있으며 명시하지 않을 경우 시스템에 의해 자동으로 제약조건의 이름이 부여된다. 인덱스를 필요로 하는 제약조건의 경우 시스템에 의해 자동으로 이름이 부여되어 인덱스가 생성된다.

예제

```
CREATE TABLE inventory(  
    subscriptionid CHAR(10),  
    isbn CHAR(10),  
    storecode CHAR(4),  
    purchasedate DATE NOT NULL,  
    quantity INTEGER CHECK(quantity < 1000),  
    paid CHAR(1),  
    PRIMARY KEY(subscriptionid, isbn),  
    CONSTRAINT fk_isbn FOREIGN KEY(isbn, storecode) REFERENCES book(isbn, storecode))  
TABLESPACE user_data;
```

```
ALTER TABLE book  
ADD CONSTRAINT const1 UNIQUE(bno);
```

삭제

ALTER TABLE문을 이용해 정의된 제약조건을 삭제할 수 있다.

예제

```
ALTER TABLE book DROP UNIQUE(bno);
```

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference* 을 참조한다.

- CREATE TABLE
- ALTER TABLE

인덱스

인덱스는 테이블 내 레코드들에 대한 빠른 접근이 가능하도록 한다. 이 절에서는 인덱스의 Altibase가 지원하는 인덱스의 종류와 속성, 인덱스 객체의 관리 및 활용

방법에 대해 설명한다.

인덱스 종류

Altibase는 BTREE, RTREE 의 두 가지 인덱스를 지원한다. RTREE 는 다차원 인덱스로서 공간 질의 시 이용된다.

B-tree 인덱스

공간 데이터 타입인 GEOMETRY 타입의 칼럼을 제외한 모든 타입의 칼럼에는 B-Tree 인덱스가 생성된다. B-Tree는 전통적으로 DBMS에서 사용해온 인덱스 구조로써 현재까지 많은 연구를 통해 여러가지 변이를 가지는데, Altibase는 이중 B+-Tree 형태의 인덱스를 지원한다.

B+-Tree는 인덱스의 최하위 레벨에 존재하는 리프 노드 (Leaf Node)들과, 최상위 레벨에 존재하는 루트 노드 (Root Node), 그리고 리프와 루트의 사이에 존재하는 인터널 노드 (Internal Node)들로 구성된다. 키 값들은 모두 리프 노드에만 존재하며, 루트와 인터널 노드는 좌측 자식 노드와 우측 자식 노드의 중간 값인 세퍼레이터 (Separator) 키들을 가진다.

R-tree 인덱스

공간 데이터 타입인 GEOMETRY 칼럼에는 R-Tree 인덱스가 생성된다.

R-Tree인덱스를 사용하여 대상 객체 검색시 Altibase는 다음의 과정을 수행한다.

1. 각 공간 객체를 감싸는 최소 사각형인 MBR (Minimum Bounding Rectangle)을 이용하여 일차로 조건 필터링 (Filtering)을 수행한다.
2. 이 결과로 남은 객체에 대해 정확한 인덱스 검색 조건을 체크하는 리파인먼트 (Refinement)를 수행한다.

R-Tree의 삽입, 삭제, 노드 스플릿 (Split), 노드 머지 (Merge) 알고리즘은 MBR을 기준으로 한다는 점만 제외하고는 B-Tree와 유사하다.

인덱스 속성

인덱스를 생성할 때 키 칼럼 구성 방법, 키 칼럼의 속성 등에 의해 해당 인덱스는 아래와 같은 인덱스 속성을 가진다.

유일 키 인덱스 (Unique Index)

인덱스 칼럼에 대해 중복 값을 허용하지 않는 인덱스이다.

유일 키 (Unique Key)와 프라이머리 키 (Primary Key)

유일 키와 프라이머리 키 모두 중복 값을 허용하지 않는 것은 공통이다. 하지만 널 (NULL)의 허용 여부에 따라 유일 키와 프라이머리 키로 구별된다. 프라이머리 키의 경우 널 (NULL)을 허용하지 않는다.

중복 키 인덱스 (Non-unique Index)

인덱스 칼럼에 대해 중복 값을 허용하는 인덱스이다. 유일 키 옵션을 지정하지 않으면 기본적으로 중복 값을 허용하는 인덱스로 생성된다.

단일 키 인덱스 (Non-composite Index)

인덱스 대상 칼럼이 하나인 인덱스이다.

복합 키 인덱스 (Composite Index)

여러 개의 칼럼들의 조합에 대해 하나의 인덱스를 생성하는 경우 복합 키 인덱스라고 한다.

직접 키 인덱스(Direct Key Index)

일반 인덱스는 인덱스 노드에 레코드의 포인터만 저장하지만, 직접 키 인덱스는 인덱스 노드에 레코드 포인터와 함께 실제 레코드도 저장하기 때문에 인덱스 스캔 비용을 줄일 수 있다.

인덱스 관리

인덱스는 테이블의 레코드들에 대한 접근을 빠르게 하기 위해서 사용된다. 인덱스는 해당 테이블로부터 물리적, 논리적으로 독립적인 객체이기 때문에 테이블에 관계없이 생성, 삭제 또는 수정할 수 있다.

테이블의 레코드들이 수정되면 해당 인덱스들도 수정이 된다. 그러므로 필요한 경우에만 인덱스를 생성하고, 테이블에 대한 접근 유형에 따라 인덱스를 변경하거나 삭제하여 최적화된 인덱스를 관리한다.

인덱스 생성

인덱스는 테이블에 존재하는 하나 이상의 칼럼에 대해 생성된다. 인덱스는 테이블 제약조건을 통해서 자동으로 생성될 수도 있고, CREATE INDEX문을 사용해 사용자가 명시적으로 생성할 수도 있다.

예제

테이블 제약조건에 의한 인덱스 생성

```
CREATE TABLE TB1 (C1 INTEGER PRIMARY KEY, C2 INTEGER UNIQUE);
```


테이블 제약조건 변경에 의한 인덱스 생성

```
ALTER TABLE TB1 ADD PRIMARY KEY (C1);
ALTER TABLE TB1 ADD UNIQUE (C2);
```

인덱스 생성시 칼럼 정렬 지정

```
CREATE INDEX TB1_IDX1 ON TB1 (C1 ASC, C2 DESC);
```

인덱스 타입 지정

```
CREATE INDEX TB1_IDX1 ON TB1 (C1) INDEXTYPE IS BTREE ;
```

UNIQUE 인덱스 생성

```
CREATE UNIQUE INDEX TB1_IDX ON TB1 (C1) ;
```

디스크 B-tree 인덱스의 생성 옵션 (NOLOGGING, NOFORCE)

디스크 B-tree 인덱스 생성 시 인덱스 생성과 관련된 로그를 기록하도록 하여 시스템 오류 발생시 복구에 사용할 수 있다. 디스크 B-tree 인덱스를 생성할 때 기록되는 로그 양과 인덱스 생성 소요 시간을 단축하려면, NOLOGGING 옵션을 사용하면 된다.

NOLOGGING 옵션을 사용하면 인덱스가 구축된 후 인덱스의 모든 페이지를 디스크에 즉시 반영해서 인덱스 생성 후에 시스템 고장이 발생하더라도 인덱스의 일관성을 보장할 수 있게 된다.

그러나 NOLOGGING 옵션으로 인덱스를 생성할 때 인덱스 페이지들을 즉시 디스크에 반영하지 않는 NOFORCE 옵션을 함께 명시하면, 인덱스를 구축하는 데 필요한 시간이 감소됨에도 불구하고 시스템이나 미디어 고장이 발생했을 경우 인덱스 일관성이 깨질수 있다. NOLOGGING과 NOFORCE 옵션을 모두 지정해서 생성된 인덱스의 연속성을 보장하기 위해서는 수동으로 미디어 백업을 수행해야 한다.

	인덱스 생성시간	일관성 및 연속성
LOGGING	인덱스 생성 시간 + 로깅 시간	시스템 고장 및 미디어 고장 시 복구가능
NOLOGGING FORCE	인덱스 생성 시간 + 인덱스 페이지를 디스크에 기록하는 시간	시스템 고장 시 복구 가능하지만, 미디어 고장 시 일관성이 깨어질 수 있음
NOLOGGING NOFORCE	인덱스 생성 시간	시스템 고장 및 미디어 고장 시 일관성이 깨어질 수 있음

예제

로깅을 하지 않고 인덱스를 생성하고 인덱스를 디스크에 반영

```
CREATE INDEX TB1_IDX1 ON TB1(C1) NOLOGGING;
```

또는

```
CREATE INDEX TB1_IDX1 ON TB1(C1) NOLOGGING FORCE;
```

로깅을 하지 않고 (NOLOGGING) 인덱스를 생성한 후 디스크에 반영하지 않음 (NOFORCE)

```
CREATE INDEX TB1_IDX1 ON TB1(C1) NOLOGGING NOFORCE;
```

인덱스 변경

인덱스 활성화 여부 속성을 ALTER INDEX문을 사용하여 변경할 수 있다.

인덱스 삭제

인덱스의 삭제는 DROP INDEX문을 사용하여 명시적으로 삭제하거나 관련 제약조건을 삭제하여 묵시적으로 삭제될 수 있다.

예제

```
DROP INDEX emp_idx1;
```

인덱스 활용

상향식 인덱스 생성

Altibase는 상향식으로 인덱스를 구축 (Bottom-Up Index Building)한다. 그러므로 데이터를 업로드한 후 인덱스를 생성하는 것이 효율적이다. 테이블에 인덱스가 생성되어 있는 상태에서 대량의 데이터를 삽입할 경우, 레코드가 삽입될 때마다 인덱스에도 반영되므로 성능이 느려진다.

디스크 인덱스의 일관성

NOLOGGING 옵션으로 생성된 디스크 테이블 인덱스의 경우 시스템 고장이나 미디어 고장 발생 시 인덱스의 일관성을 보장할 수 없는 경우가 발생한다. 이런 경우, 디스크 인덱스의 일관성을 V\$DISK_BTREE_HEADER 성능 뷰로 확인해야 한다. 만약 IS_CONSISTENT가 'F'인 인덱스가 존재한다면 해당 인덱스를 삭제하고 필요할 경우 재생성하여 사용하라.

함수 기반 인덱스 (Function-based Index)

함수 기반 인덱스는 함수 또는 수식의 결과 값을 기반으로 생성하는 인덱스이다. 함수 기반 인덱스 생성에 사용된 것과 동일한 수식이 포함된 질의를 수행할 경우, 이 함수 기반 인덱스가 사용되어 빠른 질의 처리를 기대할 수 있다.

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference* 을 참조한다.

- CREATE TABLE
- ALTER TABLE
- CREATE INDEX
- ALTER INDEX
- DROP INDEX

뷰

뷰(View)란 하나 이상의 테이블, materialized view 또는 뷰를 기반으로 한 논리적 테이블(logical table)로서, 데이터 자체는 실제로 포함하지 않는다. 이 절에서는 뷰의 관리 방법에 대해 설명한다.

베이스 (base) 테이블과 뷰

베이스 테이블이란 뷰가 접근하여 데이터를 읽어 오는 객체 (테이블, materialized view 또는 뷰)이다. 하나의 뷰에 여러 개의 베이스 테이블이 연관될 수 있다.

생성

뷰는 CREATE VIEW문을 사용하여 생성할 수 있다.

예제

```
CREATE VIEW avg_sal AS
  SELECT DNO, AVG(salary) emp_avg_sal
  -- salary average of each department
  FROM employees
  GROUP BY dno;
```

변경

이미 존재하는 뷰에 대해 뷰의 생성 구문 즉, 뷰 밑에 있는 SELECT쿼리 구문을 변경하고자 하는 경우에는 CREATE OR REPLACE VIEW문을 사용할 수 있다.

예제

```
CREATE OR REPLACE VIEW emp_cus AS
SELECT DISTINCT o.eno, e.e_lastname, c.c_lastname
FROM employees e, customers c, orders o
WHERE e.eno = o.eno AND o.cno = c.cno;
```

컴파일

뷰는 베이스 테이블들을 참조하므로 베이스 테이블에 DDL문이 발생하여 베이스 테이블의 정의가 변경되는 경우 관련 뷰들은 수행이 불가능한 무효한 상태가 될 수 있다. 이런 경우 ALTER VIEW문을 COMPILE 옵션과 함께 사용해서 재컴파일하면 유효한 상태로 만들 수 있다.

예제

```
ALTER VIEW avg_sal COMPILE;
```

삭제

뷰는 DROP VIEW문을 사용하여 삭제할 수 있다.

예제

```
DROP VIEW avg_sal;
```

데이터 조작

일반 테이블과 마찬가지로 뷰에 대해서도 SELECT문으로 데이터를 조회할 수 있고, INSERT, UPDATE 또는 DELETE문으로 데이터를 변경할 수 있다. 뷰에 대한 DML (INSERT, UPDATE, DELETE) 수행으로 베이스 테이블의 데이터를 변경할 수 있는 뷰를 변경 가능 뷰(Updatable View)라고 말한다. 변경 가능 뷰는 베이스 테이블의 행과 뷰의 행이 일대일 관계이어야 한다, 그러나 아래의 요소를 포함하는 뷰는 변경할 수 없다:

- 집계 함수, 분석 함수
- DISTINCT, ROWNUM 연산자
- GROUP BY, HAVING절
- UNION 또는 UNION ALL 등의 집합 연산자
- select list에 부질의 또는 칼럼 연산
- FROM 절에 변경 불가능 뷰
- WHERE 절의 부질의가 FROM 절의 테이블을 참조
- CONNECT BY 또는 START WITH 절

예제

변경 가능한 뷰 simple_emp를 생성한 후, 이 뷰에 UPDATE를 수행한다. UPDATE 수행 전후의 salary 값이 변경된 것을 알 수 있다.

```
CREATE VIEW simple_emp AS
  SELECT eno, e_lastname, salary
  FROM employees;
```

```
iSQL> select * from simple_emp where eno=20;
```

ENO	E_LASTNAME	SALARY
-----	------------	--------

20	Blake	
----	-------	--

1 row selected.

```
iSQL> update simple_emp set salary=2000 where eno=20;
```

1 row updated.

```
iSQL> select * from simple_emp where eno=20;
```

ENO	E_LASTNAME	SALARY
-----	------------	--------

20	Blake	2000
----	-------	------

1 row selected.

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference*을 참조한다.

- CREATE VIEW
- ALTER VIEW
- DROP VIEW
- SELECT
- INSERT
- DELETE
- UPDATE

Materialized View

Materialized view란 쿼리의 결과를 데이터로 저장하고 있는 데이터베이스 객체이다. 하나 이상의 테이블, 뷰, 및 다른 materialized view에 기반하여 데이터를 구성할 수 있다. Materialized view는 이중화 될 수 없다.

이 절에서는 materialized view의 관리 방법에 대해 설명한다.

베이스 (base) 테이블과 Materialized view

베이스 테이블이란 `materialized view`가 접근하여 데이터를 읽어 오는 객체 (테이블, `materialized view` 또는 뷰)이다. 여러 개의 베이스 테이블이 하나의 `materialized view`에 연관될 수 있다.

Altibase는 읽기 전용 `materialized view`만 지원한다. `Updatable materialized view`와 `Writable materialized view`는 지원하지 않는다.

생성

`Materialized view`는 `CREATE MATERIALIZED VIEW`문을 사용하여 생성할 수 있다. 일반 테이블과 마찬가지로 데이터가 저장될 테이블스페이스를 지정할 수는 있으나, 생성 시에 칼럼 정의와 제약 조건은 지정할 수 없다. 단, 생성 후에 "`ALTER TABLE mview_name ...`" 구문을 사용해서 칼럼 정의를 변경하거나 제약 조건을 추가할 수 있다.

예제

```
CREATE MATERIALIZED VIEW avg_sal
  TABLESPACE sys_tbs_mem_data
  BUILD IMMEDIATE
  REFRESH FORCE ON DEMAND
  AS SELECT DNO, AVG(salary) emp_avg_sal
  FROM employees
  GROUP BY dno;
```

변경

`ALTER MATERIALIZED VIEW`문을 사용해서 `materialized view`의 refresh 방법과 시기를 변경할 수 있다.

`Materialized view`의 정의 변경은 데이터를 실제 저장하고 있는 `materialized view`와 동일한 이름의 테이블 정의를 변경해서 가능하다. 단, 이렇게 테이블 정의를 변경하면, `materialized view`의 refresh 작업이 실패할 수 있다.

예제

<예제> `Materialized view`의 리프레쉬 방법을 변경하라.

```
ALTER MATERIALIZED VIEW avg_sal REFRESH COMPLETE;
```

<예제> `Materialized view`의 정의를 변경하라.

```
ALTER TABLE avg_sal ADD PRIMARY KEY (dno);
```

Refresh

REFRESH MATERIALIZED VIEW 저장 프로시저를 사용해서 사용자가 직접 materialized view의 데이터를 최신 데이터로 갱신할 수 있다.

예제

```
EXEC REFRESH MATERIALIZED_VIEW('SYS', 'AVG_SAL');
```

삭제

Materialized view는 DROP MATERIALIZED VIEW문을 사용하여 삭제할 수 있다.

예제

```
DROP MATERIALIZED VIEW avg_sal;
```

TRUNCATE

TRUNCATE TABLE문을 사용해서 materialized view에 저장된 데이터를 삭제할 수 있다.

데이터 조작

Altibase는 materialized view에 대해서 SELECT만 지원한다.

관련 SQL문

Materialized view에 대해 다음과 같은 SQL문을 제공하며, 이에 대한 자세한 설명은 *SQL Reference*을 참조한다.

- CREATE MATERIALIZED VIEW
- ALTER MATERIALIZED VIEW
- DROP MATERIALIZED VIEW

Materialized view를 위해 실제 데이터가 유지되는 테이블에 대해 다음과 같은 SQL문을 제공하며, 이에 대한 자세한 설명은 *SQL Reference*을 참조한다.

- ALTER TABLE
- TRUNCATE TABLE
- LOCK TABLE
- SELECT

시퀀스

Altibase는 연속된 숫자 값 생성자로서 시퀀스 (sequence) 객체를 제공한다. 시퀀스의 다음 값들은 일관된 성능 보장을 위해 캐싱할 수 있다.

시퀀스의 용도

시퀀스 생성자는 디스크 I/O 또는 트랜잭션 잠금의 오버헤드 없이 연속된 유일한 숫자를 생성하기 위해서 다중 사용자 환경에서 특히 유용하다. 예를 들어, 두 사용자가 동시에 orders 테이블에 새로운 레코드를 삽입한다고 가정하자. Order_id 칼럼에 입력될 유일한 주문 번호를 생성하기 위해 시퀀스를 사용하면, 어느 사용자도 다음의 가능한 주문 번호를 입력하기 위해서 다른 사용자를 기다리지 않아도 된다. 시퀀스는 각 사용자를 위해서 유일한 값을 자동으로 생성한다.

시퀀스 (sequence)는 DML 문으로 임의의 칼럼에 입력할 키 값을 생성하기 위해 주로 사용된다. [sequence 이름].NEXTVAL과 [sequence 이름].CURRVAL은 시퀀스에 접근하기 위해 사용된다.

- [sequence 이름].NEXTVAL은 시퀀스의 다음 값을 구하기 위해 사용된다.
- [sequence 이름].CURRVAL은 시퀀스의 현재 값을 구하기 위해 사용된다.

시퀀스 생성 후 그 시퀀스에 대해서 최초로 수행하는 연산이 [sequence 이름].CURRVAL일 수 없다. [sequence 이름].CURRVAL을 사용하기 위해서는 시퀀스 생성 이후 반드시 [sequence 이름].NEXTVAL을 먼저 사용해야 한다.

시퀀스의 다음 값에 접근할 때마다 시퀀스의 값은 내부적으로 명시한 증감분 (increment by)만큼 증가한다. 시퀀스의 증감분은 시퀀스 생성시 명시적으로 그 값이 주어지지 않는 경우 기본적으로 1이다.

INSERT문에서의 시퀀스 사용

시퀀스를 사용하여 키를 생성하여 레코드를 삽입하는 예제이다.

예제

```
create sequence seq1;  
insert into t1 values (seq1.nextval);
```

위에 예제에서, 시퀀스 생성시 초기값은 1이므로 t1테이블에는 1이 입력되며 시퀀스의 다음 값은 1 증가한 2가 될 것이다.

생성

CREATE SEQUENCE문을 사용하여 시퀀스를 생성할 수 있다. 시퀀스 생성 구문에 사용되는 옵션들은 다음과 같다.

- START WITH
시퀀스의 시작 값
- INCREMENT BY
시퀀스의 증감 분
- MAXVALUE
시퀀스의 최대값
- MINVALUE
시퀀스의 최소값

- CYCLE
이 옵션은 시퀀스가 최대값 또는 최소값에 도달했을 때 다음 값을 계속 생성하는 것을 보장한다. 시퀀스는 오름차순 시퀀스인 경우는 최소값부터, 내림차순 시퀀스인 경우는 최대값부터 다시 순환된다.

- CACHE
Altibase는 시퀀스 값을 보다 빠르게 액세스하기 위하여 미리 생성하여 메모리에 캐시한다. 이렇게 캐시되는 시퀀스 값의 개수는 CACHE 옵션으로 지정할 수 있다. 시퀀스 캐시는 시퀀스가 처음 참조될 때 채워지며 다음 시퀀스 값을 요청할 때마다 캐시된 시퀀스에서 반환된다. 캐시된 마지막 시퀀스 값을 사용한 이후에 발생한 시퀀스 요청에 대해서는 새로 시퀀스 값을 생성하여 메모리에 캐시하고 그 캐시의 첫번째 값을 반환한다. 시퀀스 생성시 기본 CACHE 값은 20이다.

예제

기본적인 시퀀스 생성하기 (1부터 시작하며 1씩 증가)

```
CREATE SEQUENCE seq1;
```

짝수를 생성하고 0에서 100까지 순환하는 시퀀스 생성하기

```
CREATE SEQUENCE seq1
START WITH 0
INCREMENT BY 2
MAXVALUE 100
CYCLE ;
```

변경

ALTER SEQUENCE문을 사용하여 START WITH의 값을 제외한 모든 시퀀스 옵션을 변경할 수 있다.

예제

```
ALTER SEQUENCE seq1  
  INCREMENT BY 1  
  MINVALUE 0  
  MAXVALUE 100;
```

삭제

DROP SEQUENCE문을 사용하여 명시한 시퀀스를 삭제할 수 있다.

예제

```
DROP SEQUENCE seq1;
```

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference* 을 참조한다.

- CREATE SEQUENCE
- ALTER SEQUENCE
- DROP SEQUENCE

시노님

Altibase는 테이블, 뷰, 시퀀스, 저장 프로시저 및 저장 함수에 대한 별칭(alias)으로써 시노님을 제공한다.

시노님의 장점

다음과 같은 경우 데이터베이스 시노님을 사용하면 많은 이점이 있다.

- 특정 객체를 생성한 사용자와 객체의 원래 이름을 숨기고 싶은 경우
- SQL문의 사용을 단순화하고자 하는 경우
- 사용자 변경에 따른 응용프로그램의 변경을 최소화하고자 하는 경우

생성

CREATE SYNONYM문을 사용하여 시노님을 생성할 수 있다.

예제

테이블 dept의 별칭으로 my_dept 시노님을 생성하라.

```
CREATE SYNONYM my_dept FOR dept;
```

삭제

DROP SYNONYM문을 사용하여 명시한 시노임을 삭제할 수 있다.

예제

시노임 my_dept를 삭제하라.

```
DROP SYNONYM my_dept;
```

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference*을 참조한다.

- CREATE SYNONYM
- DROP SYNONYM

저장 프로시저 및 저장 함수

저장 프로시저 (Stored Procedure)란 SQL문들과 흐름 제어문, 할당문, 오류 처리 루틴 등의 조합하여 전체 업무 절차를 하나의 모듈로 프로그래밍한 것이다. 이 모듈은 데이터베이스에 데이터베이스 객체로서 영구적으로 저장되어, 모듈 이름을 호출하여 전체 업무 절차를 서버에서 한번에 수행할 수 있다. 이 절에서는 저장 프로시저 관리 방법에 대해 설명한다.

저장 프로시저와 저장 함수는 리턴값 존재 유무에 따라 구별된다. 그 외의 모든 점은 동일하므로, 특별한 언급이 없는 한 모든 설명들은 공통 사항이다.

또한 이 절에서는 저장 프로시저 관리 방법을 보여주는 간단한 예제를 제공한다.

저장 프로시저의 용어와 개념, 자세한 관리 방법에 대해서는 *Stored Procedures Manual*을 참조한다.

종류

저장 프로시저 (Stored Procedure)

저장 프로시저는 입력 인자, 출력 인자, 입출력 인자를 가지고 바디(body) 내에 정의된 조건에 따라 여러 SQL문을 한번에 수행하는 데이터베이스 객체이다. 리턴값을 가지지 않으며 출력 인자와 입출력 인자들을 통해 클라이언트에게 값을 전달한다.

하나의 리턴 값을 갖지 않기 때문에 다른 SQL문의 expression 내에 피연산자로 사용될 수 없다.

저장 함수 (Stored Function)

저장 함수는 하나의 리턴 값을 가지는 것을 제외하면 저장 프로시저와 동일하다. 저장 프로시저와 달리 하나의 리턴 값을 가지므로 시스템 제공 함수들처럼 다른 SQL문의 expression 내에 피연산자로 사용될 수 있다.

타입 세트 (Type Set)

저장 프로시저 내에서 사용되는 사용자 정의 타입들을 정의한 집합이다. 이는 주로 저장 프로시저끼리 파라미터 또는 리턴값으로 사용자 정의 타입을 주고받을 때 사용된다.

저장 프로시저 관련 SQL 구문

저장 프로시저의 SQL 구문 종류를 살펴보면, 다음과 같다.

종류	관련 문장	설명
생성	CREATE [OR REPLACE] PROCEDURE 문	새로운 저장 프로시저를 생성하거나 이미 생성된 저장 프로시저의 재정의하는 SQL문이다.
	CREATE [OR REPLACE] FUNCTION 문	새로운 저장 함수를 생성하거나 이미 생성된 저장 함수의 재정의하는 SQL문이다.
	CREATE [OR REPLACE] TYPESET 문	타입 세트를 생성 또는 변경하는 SQL문이다.
변경	ALTER PROCEDURE 문	저장 프로시저 생성 이후 관련 객체들의 정의가 변경된다면 현재 저장 프로시저의 실행 계획은 최적화된 상태가 아닐 것이다. 이 경우 이를 재 컴파일하여 최적화된 실행 계획을 재 생성하는 SQL문이다.
	ALTER FUNCTION 문	저장 함수 생성 이후 관련 객체들의 정의가 변경된다면 현재 저장 함수의 실행 계획은 최적화된 상태가 아닐 것이다. 이 경우 이를 재 컴파일하여 최적화된 실행 계획을 재 생성하는 SQL문이다.
삭제	DROP	생성된 저장 프로시저를 삭제하는 SQL문이다.

	PROCEDURE 문	
	DROP FUNCTION 문	생성된 저장 함수를 삭제하는 SQL문이다.
	DROP TYPESET 문	생성된 타입 세트를 삭제하는 SQL문이다.
실행	EXECUTE 문	저장 프로시저 또는 저장 함수를 실행하는 SQL문이다.
	[함수 이름]	SQL문 내에서 built-in function과 같은 저장 함수를 호출한다.

[표 5-1] 저장 프로시저문의 종류

생성

저장 프로시저는 CREATE PROCEDURE문을 사용하여 생성할 수 있다.

예제

```

CREATE PROCEDURE proc1
(p1 IN INTEGER, p2 IN INTEGER, p3 IN INTEGER)
AS
    v1 INTEGER;
    v2 t1.i2%type;
    v3 INTEGER;
BEGIN
    SELECT *
    INTO v1, v2, v3
    FROM t1
    WHERE i1 = p1 AND i2 = p2 AND i3 = p3;

    IF v1 = 1 AND v2 = 1 AND v3 = 1 THEN
        UPDATE t1 SET i2 = 7 WHERE i1 = v1;
    ELSIF v1 = 2 AND v2 = 2 AND v3 = 2 then
        UPDATE t1 SET i2 = 7 WHERE i1 = v1;
    ELSIF v1 = 3 AND v2 = 3 AND v3 = 3 then
        UPDATE t1 SET i2 = 7 WHERE i1 = v1;
    ELSIF v1 = 4 AND v2 = 4 AND v3 = 4 then
        UPDATE t1 SET i2 = 7 WHERE i1 = v1;
    ELSE    -- ELSIF v1 = 5 AND v2 = 5 AND v3 = 5 then
        DELETE FROM t1;
    END IF;

    INSERT INTO t1 VALUES (p1+10, p2+10, p3+10);
END;
/

```

변경

기존의 저장 프로시저의 이름은 유지하면서 저장 프로시저의 파라미터 또는 본체를 변경하고자 할 때는 CREATE OR REPLACE PROCEDURE문을 사용하여 저장 프로시저를 재 생성해야 한다.

예제

```

CREATE OR REPLACE PROCEDURE proc1
(p1 IN INTEGER, p2 IN INTEGER, p3 IN INTEGER)
AS
    v1 INTEGER;
    v2 t1.i2%type;
    v3 INTEGER;
BEGIN
    .
    .
    .
END;
/

```

저장 프로시저에서 참조되는 테이블, 시퀀스, 다른 저장 프로시저 또는 저장 함수의 정의가 변경되어 생성 시의 정의와 달라지면 현재 이 저장 프로시저의 실행 계획으로는 저장 프로시저를 실행할 수 없게 된다. 이 경우 현재 이 저장 프로시저는 무효한 (invalid) 상태라고 한다.

예를 들면 처음 저장 프로시저 생성 시 존재하던 인덱스가 삭제된 경우 이전 실행 계획은 인덱스를 통해 테이블에 접근하도록 계획되어 있으므로 이전 실행 계획을 사용해 테이블에 접근할 수 없게 된다.

ALTER PROCEDURE문은 무효한 저장 프로시저를 재 컴파일하여 유효한 (valid) 상태의 실행 계획을 재 생성하는 데 사용된다.

예제

```
ALTER PROCEDURE proc1 COMPILE;
```

삭제

저장 프로시저는 DROP PROCEDURE문을 사용하여 삭제할 수 있다.

예제

```
DROP PROCEDURE proc1;
```

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *Stored Procedures Manual*을 참조한다.

- CREATE PROCEDURE
- CREATE FUNCTION
- CREATE TYPESET
- ALTER PROCEDURE
- ALTER FUNCTION
- DROP PROCEDURE
- DROP FUNCTION
- DROP TYPE SET
- EXECUTE
- [함수 이름]

트리거

트리거란 테이블에 데이터가 삽입, 삭제, 또는 갱신될 때 시스템에 의해 작동되어 특정 작업 절차를 자동으로 수행하는 저장 프로시저의 한 종류이다. 이 절에서는 트리거 관리 방법에 대해 설명한다.

트리거 구성 요소

다음의 트리거 구성요소는 트리거 작동 시점, 트리거 작동 여부, 트리거 작업을 결정한다.

- 트리거 이벤트 (trigger event)
수행시 트리거 작동을 유발하는 SQL문을 트리거 이벤트라고 한다.
- 트리거 조건 (trigger condition (WHEN 절))
이는 트리거를 작동시키기 위해 만족시켜야 하는 SQL 조건이다.
- 트리거 액션 (trigger action)
이는 트리거 조건이 TRUE일 때 트리거가 수행하는 저장 프로시저의 본체(body)이다.

트리거 이벤트

트리거 작동을 유발시키는 이벤트로서 다음 세 DML 구문 중 하나를 명시할 수 있다.

- DELETE
해당 테이블의 데이터를 삭제하는 DELETE 구문 수행 시마다 트리거가 동작된다.
- INSERT
해당 테이블의 데이터를 삽입하는 INSERT 구문 수행 시마다 트리거가 동작된다.
- UPDATE
해당 테이블의 데이터를 변경하는 UPDATE 구문 수행 시마다 트리거가 동작된다.
UPDATE 트리거 이벤트에 OF 절이 있을 경우, OF 절에 지정된 칼럼의 데이터가 변경될 때만 트리거가 동작된다.

Note: 데이터베이스의 무결성을 위해 이중화에 의한 테이블의 변경은 트리거 이벤트로 처리되지 않는다.

생성

트리거는 CREATE TRIGGER문을 사용하여 생성할 수 있다.

예제


```
CREATE TRIGGER del_trigger
AFTER DELETE ON orders
REFERENCING OLD ROW old_row
FOR EACH ROW
AS BEGIN
    INSERT INTO log_tbl VALUES(old_row.ono, old_row.cno, old_row.qty, old_row.arrival_date, sysdate);
END;
/
```

변경

존재하는 트리거를 비활성화 시키거나 무효한 상태의 트리거를 재컴파일하는 경우 ALTER TRIGGER문을 사용할 수 있다. 트리거는 처음 생성될 때, 기본적으로 자동 활성화된다. ALTER TRIGGER 구문에 DISABLE과 ENABLE 절을 이용하여 트리거를 비활성화하고 활성화 시킬수 있다.

예제

```
ALTER TRIGGER del_trigger DISABLE;
```

삭제

DROP TRIGGER문을 사용해 트리거를 삭제할 수 있다.

예제

```
DROP TRIGGER del_trigger;
```

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference* 을 참조한다.

- CREATE TRIGGER
- ALTER TRIGGER
- DROP TRIGGER

트리거는 저장 프로시저의 한 종류이므로 트리거 본체(body)에 대해서는 *Stored Procedures Manual*을 참조한다.

작업(Job)

Altibase는 저장 프로시저에 실행 일정을 더한 JOB 객체를 제공한다. JOB 객체를 생성할 때 실행할 저장 프로시저와 실행 시각, 실행 반복 간격 등을 설정할 수 있다.

SYS 사용자만이 JOB을 생성하거나 변경 또는 삭제할 수 있으며, 한 개의 JOB에는 한 개의 프로시저만 등록할 수 있다.

생성된 JOB이 일정에 맞춰 돌아가기 위해서는 JOB 스케줄러가 동작하도록 해 주어야 한다. 작업 스케줄러의 동작은 JOB_SCHEDULER_ENABLE 프로퍼티로 제어할 수 있다.

이 절에서는 우선 작업 스케줄러를 제어하는 방법을 설명하고, 작업(Job) 객체의 생성, 변경 및 삭제하는 방법을 설명한다.

작업 스케줄러(Job Scheduler)의 시작 및 종료

JOB 객체를 생성하는 것은 저장 프로시저를 실행할 일정을 등록만 한 것이고, 실제로 일정에 맞춰서 작업이 실행되게 하려면 작업 스케줄러가 동작하도록 해야 한다.

작업 스케줄러 시작

작업 스케줄러를 사용하려면 먼저 아래의 프로퍼티의 값을 변경한다. 만약 JOB_SCHEDULER_ENABLE 프로퍼티의 값을 1로 설정하여도 JOB_THREAD_COUNT 프로퍼티의 값이 0이면 작업 스케줄러가 동작하지 않는다.

- JOB_SCHEDULER_ENABLE (0 비활성(기본값), 1: 활성)
- JOB_THREAD_COUNT (기본값: 0)

작업 스케줄러 종료

작업 스케줄러를 종료하려면 JOB_SCHEDULER_ENABLE 프로퍼티의 값을 0으로 변경한다.

제약사항

- 작업 스케줄러는 SYS 사용자만 제어할 수 있다.
- JOB_THREAD_COUNT 프로퍼티의 값을 변경한 후에는 서버를 재시작한다.

예제

<질의> 등록된 JOB이 실행되도록 작업 스케줄러를 시작하라.

```
iSQL> ALTER SYSTEM SET job_scheduler_enable = 1;  
Alter success.
```

<질의> 작업 스케줄러의 동작을 종료하라.

```
iSQL> ALTER SYSTEM SET job_scheduler_enable = 0;
Alter success.
```

작업 생성

CREATE JOB 문을 사용하여 JOB을 생성한다. CREATE JOB 문에서 실행할 저장 프로시저와 실행 시각, 실행 주기를 설정할 수 있다. 등록될 프로시저의 사용자 이름이 생략되면, SYS 사용자로 간주한다.

JOB을 생성하면 기본적으로 DISABLE 상태이며, JOB이 실행 주기에 따라 동작하려면 ENABLE 상태로 변경해야 한다.

자세한 설명은 SQL Reference의 CREATE JOB 구문을 참조하기 바란다.

제약사항

- 작업(JOB)을 생성하기 전에 JOB_SCHEDULER_ENABLE, JOB_THREAD_COUNT 프로퍼티의 값이 0이 아닌 값으로 설정되어 있어야 한다. 프로퍼티에 대한 자세한 설명은 General Reference를 참조하기 바란다.
- 한 개의 작업에 한 개의 프로시저만 등록할 수 있다.

예제

<질의> proc1 프로시저가 현재 시간에 처음 시작하여 1시간 주기로 작업을 실행 후, 3일 후에 끝나도록 JOB을 생성하라.

```
iSQL> CREATE JOB job1
EXEC proc1
START sysdate
END sysdate + 3
INTERVAL 1 HOUR;
Create success.
```

<질의> job1을 생성할 때 ENABLE 옵션을 설정하지 않았다면, job1의 상태는 DISABLE이다. 스케줄러에서 job1을 수행할 수 있도록 ENABLE 상태로 변경하라.

```
iSQL> ALTER JOB job1 SET ENABLE;
Alter success.
```

작업 변경

ALTER JOB문을 사용하여 JOB 구문의 정의를 변경할 수 있다. 자세한 설명은 SQL Reference의 ALTER JOB 구문을 참조하기 바란다.

예제

<질의> 이름이 job1인 JOB의 시작 시간을 2013년 1월 1일로 변경하라.

```
ALTER JOB job1 SET START to_date('20130101','YYYYMMDD');
```

작업 삭제

DROP JOB문을 사용하여 명시한 작업을 삭제할 수 있다.

예제

<질의> JOB 객체 job1을 제거하라.

```
DROP JOB job1;
```

작업 로그 확인하기

마지막으로 실행된 JOB의 프로시저 수행이 실패했다면, 그 에러 코드가 SYS_JOBS_메타 테이블의 ERROR_CODE 칼럼에 저장된다. 그리고 에러 메시지 등의 자세한 정보는 QP_MSGLOG_FILE 프로퍼티에 설정된 트레이스 로그 파일(기본: \$ALTIBASE_HOME/trc/altibase_qp.log)로 저장된다. 단, QP모듈에 대한 TRCLEVEL 2가 설정되어 있는 경우에만 트레이스 로그가 기록되므로, 아래의 쿼리를 이용해서 TRCLEVEL 2의 FLAG를 확인하도록 한다.

```
iSQL> SELECT * from V$TRACELOG
WHERE MODULE_NAME='QP' AND DESCRIPTION!='---';
```

MODULE_NAME	TRCLEVEL	FLAG	POWLEVEL	DESCRIPTION
QP	1	X	1	PSM Error Line Trace Log
QP	2	O	2	DDL Trace Log
QP	99	SUM	2	Total Sum of Trace Log Values

만약, TRCLEVEL 2의 FLAG가 'X'이면, 아래의 구문으로 트레이스 로깅 레벨을 변경할 수 있다.

```
ALTER SYSTEM SET qp_msglog_flag = <기존값 + 2>;
```

기존 값은 TRCLEVEL 칼럼 값이 99인 레코드의 POWLEVEL 칼럼 값을 조회해서 확인할 수 있다.

관련 프로퍼티 및 메타 테이블

작업 스케줄러와 관련된 프로퍼티는 아래와 같다.

- JOB_SCHEDULER_ENABLE
- JOB_THREAD_COUNT
- JOB_THREAD_QUEUE_SIZE

생성된 JOB에 대한 정보는 SYS_JOBS_ 메타 테이블에서 확인할 수 있다. 프로퍼티와 메타 테이블에 대한 상세한 설명은 *General Reference*를 참조한다.

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference* 을 참조한다.

- CREATE JOB
- ALTER JOB
- DROP JOB

데이터베이스 사용자

데이터베이스 생성 후 초기 데이터베이스 내에는 시스템 관리자인 SYSTEM_와 SYS 사용자만이 존재한다. 이 사용자들은 DBA (데이터베이스 관리자)이므로 일반 스키마를 구축하여 스키마 객체를 관리하기 위해서는 일반 사용자를 생성해야 한다. 이 절에서는 사용자를 생성하고 관리하는 방법에 대해 설명한다.

SYSTEM_와 SYS 사용자

SYSTEM_와 SYS 사용자는 데이터베이스 생성시 시스템에 의해 생성되는 시스템 관리자로 일반 사용자와는 구별된다.

시스템 관리자로는 메타 테이블의 소유자로 메타 테이블에 대한 DDL문과 DML문 수행 권한을 가지고 있는 SYSTEM_ 사용자와, DBA로 일반 테이블에 대해 모든 권한을 가지고 있으며 시스템 수준의 모든 작업을 수행할 수 있는 권한을 기본적으로 가지고 있는 SYS 사용자가 있다.

이들 사용자는 DDL 구문을 사용하여 임의로 변경되거나 삭제될 수 없다.

생성

CREATE USER문을 사용하여 사용자를 생성할 수 있다. 이 구문을 실행하려면 CREATE USER 시스템 권한이 있어야 한다. 사용자 생성 시 비밀번호를 지정하여야 하고, 부가적으로 사용자를 위한 기본 테이블스페이스를 설정할 수 있다.

예제

```
CREATE USER dlr IDENTIFIED BY dlr123
DEFAULT TABLESPACE user_data
TEMPORARY TABLESPACE temp_data
ACCESS sys_tbs_memory ON;
```

변경

ALTER USER문을 사용하여 사용자 비밀번호와 해당 사용자의 테이블스페이스 설정을 변경할 수 있다.

예제

사용자 비밀번호 변경

```
ALTER USER dlr IDENTIFIED BY dlr12345;
```

기본 데이터 테이블스페이스 변경

```
ALTER USER dlr DEFAULT TABLESPACE dlr1_data;
```

임시 테이블스페이스 변경

```
ALTER USER dlr TEMPORARY TABLESPACE dlr1_tmp;
```

특정 테이블스페이스 접근 허용 여부 변경

```
ALTER USER dlr ACCESS dlr2_data ON;
```

삭제

사용자를 삭제하고자 하는 경우 DROP USER문을 사용하면 된다. 해당 사용자의 소유로 되어 있는 모든 객체까지 한꺼번에 삭제하고자 할 경우 CASCADE 옵션을 이용하라. 해당 사용자의 스키마 내에 객체가 존재할 때 CASCADE 옵션을 사용하지 않는 경우 DROP USER문 수행 시 오류가 발생한다.

예제

```
DROP USER dlr CASCADE;
```

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference* 을 참조한다.

- CREATE USER
- ALTER USER
- DROP USER

권한과 롤

사용자가 데이터베이스 객체 또는 데이터에 접근하기 위해서는 적절한 권한이 필요하다. 이 절에서는 시스템 권한, 객체 권한 및 롤과 이를 관리하는 방법에 대해서 설명한다.

종류

Altibase는 시스템 권한, 객체 권한 및 롤을 지원한다.

시스템 권한 (System Privilege)

시스템 권한은 일반적으로 DBA가 관리한다. 시스템 권한이 있는 사용자는 데이터베이스에 특정한 작업을 수행하거나 모든 스키마에 있는 객체들에 접근할 수 있다.

Altibase가 지원하는 전체 시스템 접근 권한 목록은 다음과 같다. 각 권한에 대한 자세한 설명은 *SQL Reference* 을 참조한다.

시스템 권한	이름
DATABASE	ALTER SYSTEM
	ALTER DATABASE
	DROP DATABASE
INDEX	CREATE ANY INDEX
	ALTER ANY INDEX
	DROP ANY INDEX
LIBRARY	CREATE LIBRARY
	CREATE ANY LIBRARY
	ALTER ANY LIBRARY

	DROP ANY LIBRARY
PROCEDURE	CREATE PROCEDURE
	CREATE ANY PROCEDURE
	ALTER ANY PROCEDURE
	DROP ANY PROCEDURE
	EXECUTE ANY PROCEDURE
SEQUENCE	CREATE SEQUENCE
	CREATE ANY SEQUENCE
	ALTER ANY SEQUENCE
	DROP ANY SEQUENCE
	SELECT ANY SEQUENCE
SESSION	CREATE SESSION
	ALTER SESSION
TABLE	CREATE TABLE
	CREATE ANY TABLE
	ALTER ANY TABLE
	DELETE ANY TABLE
	DROP ANY TABLE
	INSERT ANY TABLE
	LOCK ANY TABLE
	SELECT ANY TABLE
	UPDATE ANY TABLE
TABLESPACE	CREATE TABLESPACE
	ALTER TABLESPACE
	DROP TABLESPACE
	MANAGE TABLESPACE

USER	CREATE USER
	ALTER USER
	DROP USER
VIEW	CREATE VIEW
	CREATE ANY VIEW
	DROP ANY VIEW
MISCELLANEOUS	GRANT ANY PRIVILEGES
TRIGGER	CREATE TRIGGER
	CREATE ANY TRIGGER
	ALTER ANY TRIGGER
	DROP ANY TRIGGER
MATERIALIZED VIEW	CREATE MATERIALIZED VIEW
	CREATE ANY MATERIALIZED VIEW
	ALTER ANY MATERIALIZED VIEW
	DROP ANY MATERIALIZED VIEW
ROLE	CREATE ROLE
	DROP ANY ROLE
	GRANT ANY ROLE
SYNONYM	CREATE ANY SYNONYM
	CREATE PUBLIC SYNONYM
	CREATE SYNONYM
	DROP ANY SYNONYM
	DROP PUBLIC SYNONYM
JOB	ALTER ANY JOB
	CREATE ANY JOB

	DROP ANY JOB
DIRECTORY	CREATE ANY DIRECTORY
	DROP ANY DIRECTORY
DATABASE LINK	CREATE DATABASE LINK
	CREATE PUBLIC DATABASE LINK
	DROP PUBLIC DATABASE LINK

객체 권한 (Object Privilege)

객체 권한은 객체의 소유자가 관리한다. 이들 권한은 객체에 접근하고 조작하는 것을 관리한다.

Altibase가 지원하는 객체 접근 권한 목록은 다음과 같다.

Object privilege	Table	Sequence	PSM/ External Procedure	View	directory	External Library
ALTER	O	O				
DELETE	O					
EXECUTE			O			O
INDEX	O					
INSERT	O					
REFERENCES	O					
SELECT	O	O		O	O	
UPDATE	O				O	

롤 (Role)

롤은 권한들의 묶음이다. 여러 개의 권한을 사용자들에게 부여할 때 롤을 사용하는 것이 용이하다. 롤에 대한 자세한 설명과 제약에 대해서는 SQL Reference를 참조한다.

권한 부여

GRANT문을 사용하여 특정 사용자 또는 롤에게 명시적으로 권한을 부여할 수 있다.

SYSTEM_와 SYS 사용자의 경우 DBA로서 모든 권한을 갖고 있으며, 일반 사용자 또는 롤에게 임의의 권한을 부여할 수 있다.

일반 사용자의 경우 CREATE USER문을 수행하여 사용자를 생성하면 다음의 권한들이 시스템에 의해 자동으로 부여된다.

- CREATE SESSION
- CREATE TABLE
- CREATE SEQUENCE
- CREATE PROCEDURE
- CREATE VIEW
- CREATE TRIGGER
- CREATE SYNONYM
- CREATE MATERIALIZED VIEW
- CREATE LIBRARY

예제

시스템 권한 부여

```
GRANT ALTER ANY SEQUENCE, INSERT ANY TABLE, SELECT ANY SEQUENCE TO uare5;  
GRANT ALTER ANY SEQUENCE, INSERT ANY TABLE, SELECT ANY SEQUENCE TO role1;
```

객체 권한 부여

```
GRANT SELECT, DELETE ON sys.employees TO uare8;  
GRANT SELECT, DELETE ON sys.employees TO role2;
```

권한 해제

사용자에게 이미 부여된 권한을 REVOKE문을 사용해 해제할 수 있다.

예제

시스템 권한 해제

```
REVOKE ALTER ANY TABLE, INSERT ANY TABLE, SELECT ANY TABLE,  
DELETE ANY TABLE FROM uare10;  
REVOKE ALTER ANY SEQUENCE, INSERT ANY TABLE FROM role1;
```

객체 권한 해제

```
REVOKE SELECT, DELETE ON sys.employees FROM uare7, uare8;  
REVOKE DELETE ON sys.employees FROM role2;
```

관련 SQL문

다음과 같은 SQL문을 제공하며 이에 대한 자세한 설명은 *SQL Reference* 을 참조한다.

- CREATE ROLE
- DROP ROLE
- GRANT
- REVOKE

6.테이블스페이스

이 장에서는 관리자가 알아야할 테이블스페이스의 개념, 테이블스페이스 구조와 그 사용을 위해서 지원되는 기능에 대해서 설명하고, 효율적인 테이블스페이스 관리를 위해서 관리자들이 알아야 할 정보를 전달한다.

테이블스페이스 정의 및 구조

본 절에서는 테이블스페이스가 무엇인지 살펴본다. 또한, 테이블스페이스와 데이터베이스의 관계를 알아보고, 디스크 테이블스페이스, 메모리 테이블스페이스, 휘발성 테이블스페이스들이 각각 어떤 구조를 갖고 있는지 설명한다.

테이블스페이스의 정의

테이블스페이스는 테이블, 인덱스 등의 데이터베이스 객체들이 저장되는 논리적인 저장소 (Storage)이다. 데이터베이스는 올바른 운영을 위해 기본적으로 하나 이상의 테이블스페이스를 필요로 한다. 시스템 테이블스페이스는 데이터베이스 생성시 자동으로 생성된다. 그리고 사용자가 임의로 사용자 정의 테이블스페이스를 생성할 수 있다.

Altibase는 사용자 정의 테이블스페이스를 데이터베이스 객체가 디스크에 상주하는 디스크 테이블스페이스와 메모리에 상주하는 메모리 테이블스페이스, 메모리에 상주하면서 로깅을 하지 않는 휘발성 테이블스페이스로 구분해 지원한다. 따라서 사용자는 테이블스페이스에 저장되는 데이터의 특성에 따라 디스크, 메모리, 또는 휘발성 테이블스페이스 중에서 어떤 것을 사용할 것인지 결정할 수 있다.

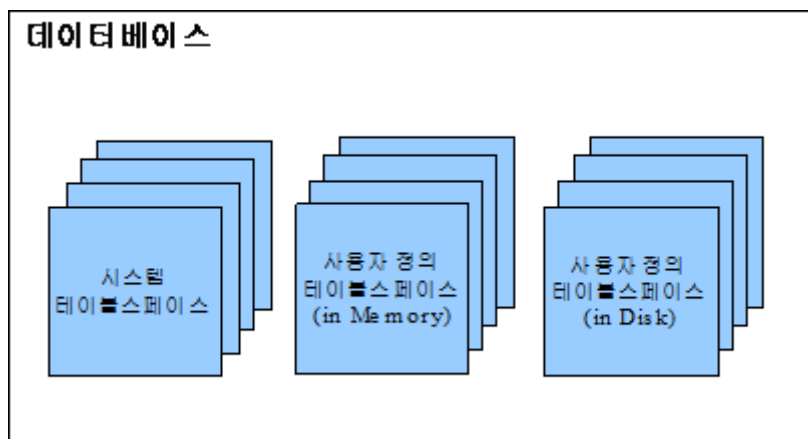
예를 들어 이력 데이터와 같은 대용량 데이터를 위해서는 디스크 테이블스페이스가 적합하다. 또한 접근 빈도가 높은 소용량 데이터는 메모리 테이블스페이스를, 빠른 처리를 위한 임시 데이터 처리는 휘발성 테이블스페이스를 사용하는 것이 적합하다.

데이터베이스와 테이블스페이스의 관계

Altibase 데이터베이스 생성시 4종류의 시스템 테이블스페이스 (시스템 디렉터리 테이블스페이스, 시스템 데이터 테이블스페이스, 시스템 언두 테이블스페이스, 시스템 임시 테이블스페이스) 들이 자동으로 생성된다.

또한 사용자가 테이블스페이스가 필요할 경우 사용자 정의 테이블스페이스 (디스크, 메모리, 또는 휘발성 테이블스페이스)를 생성할 수 있다. 사용자 정의 테이블스페이스는 데이터 특성에 따라 디스크 또는 메모리에 선택적으로 생성할 수 있다.

[그림 6-1]은 데이터베이스와 테이블스페이스의 관계를 보여준다.



[그림 6-1] 테이블스페이스와 데이터베이스의 관계

디스크 테이블스페이스 구조

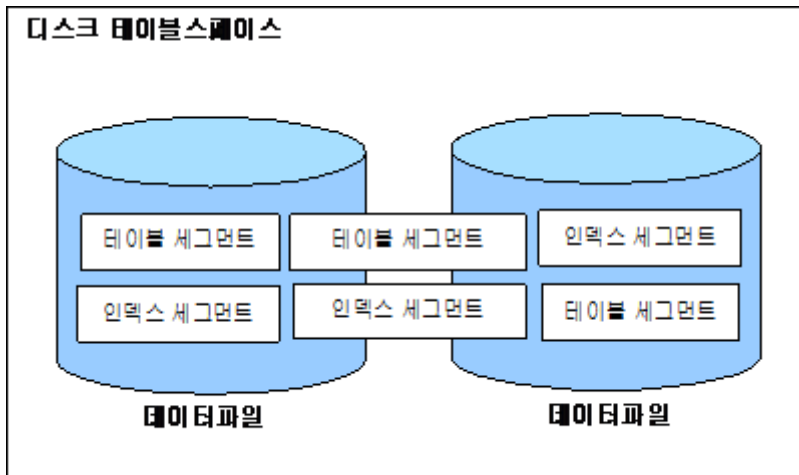
디스크 테이블스페이스는 모든 데이터가 디스크 공간에 저장되는 테이블스페이스이다. 물리적으로는 데이터 파일로 구성되며, 논리적으로 세그먼트, 익스텐트 및 페이지로 구성된다.

물리적 구조

디스크 테이블스페이스는 데이터 파일, 세그먼트와 밀접한 관계를 갖는다. [그림 6-2]는 디스크 테이블스페이스와 데이터 파일 및 세그먼트의 연관 관계를 설명한다.

디스크 테이블스페이스, 데이터 파일 및 세그먼트는 다음과 같은 특징을 갖는다. 디스크 테이블스페이스는 하나 이상의 데이터 파일로 구성되며, 데이터 파일은 운영체제에서 제공되는 파일 형태로 존재한다. 세그먼트는 논리적으로

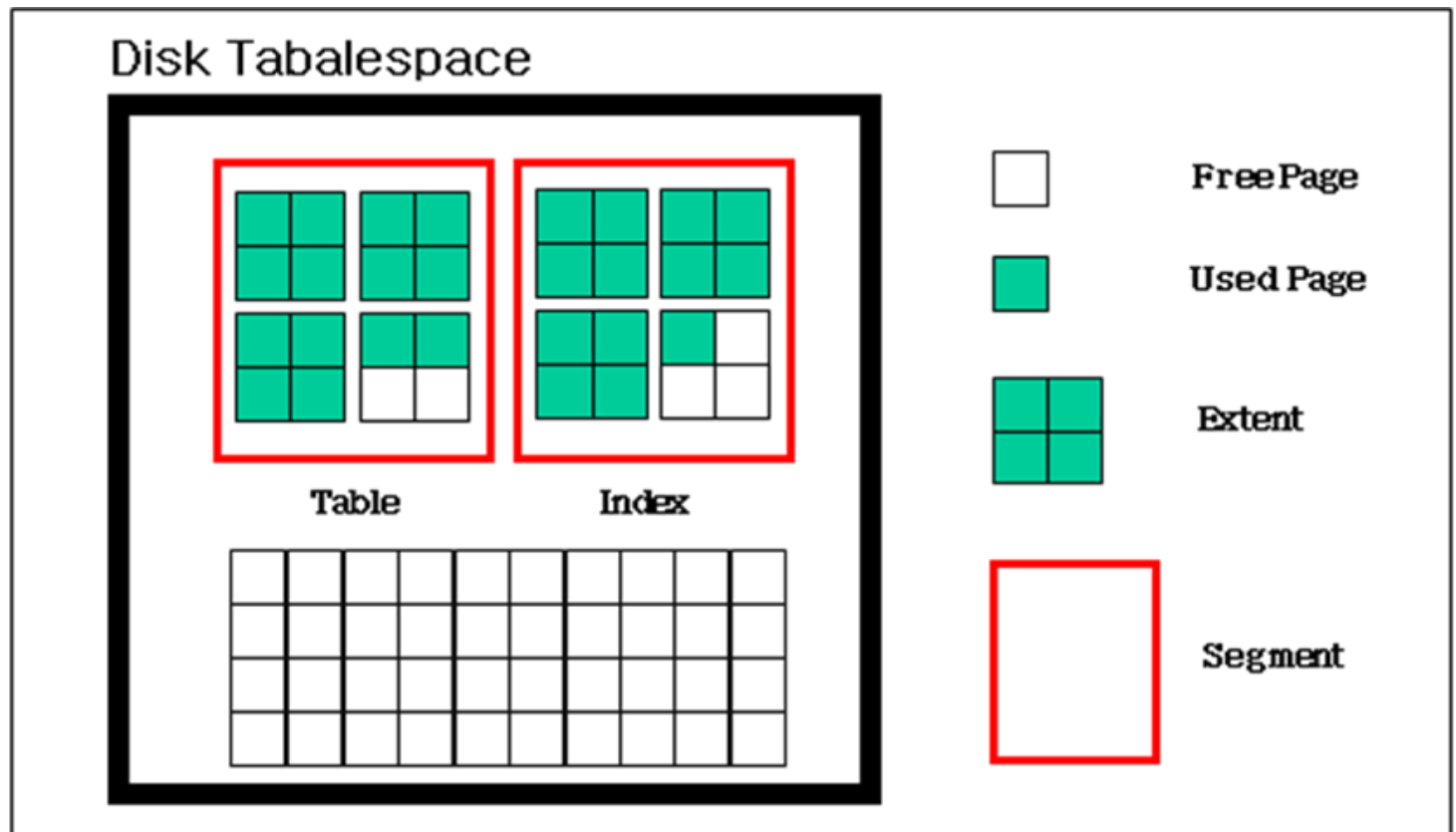
테이블스페이스에 저장되며, 물리적으로 데이터 파일에 저장된다. 세그먼트는 특정 디스크 테이블스페이스에 종속적이며, 세그먼트가 참조하는 세그먼트는 다른 디스크 테이블스페이스에 저장될 수 있다.



[그림 6-2] 디스크 테이블스페이스, 데이터 파일, 세그먼트의 연관 관계

논리적 구조

디스크 테이블스페이스는 논리적으로 세그먼트, 익스텐트 및 페이지로 구성된다. 이들의 관계를 살펴보면 아래 그림과 같다.



[그림 6-3] 디스크 테이블스페이스의 논리적 구조

세그먼트(Segment)

세그먼트는 익스텐트의 집합으로 테이블스페이스 내의 모든 객체가 여기에 저장된다. 세그먼트는 테이블스페이스 내에서 테이블 또는 인덱스를 할당하는 단위이다. 하나의 테이블 또는 인덱스는 논리적으로 하나의 세그먼트로 볼 수 있다. Altibase에서 사용하는 세그먼트의 종류는 다음과 같다.

종 류	설 명
Table 세그먼트	데이터베이스 안에 데이터를 저장하는 가장 기본적인 수단이다. 한 개의 테이블 세그먼트에는 파티션 되지 않은 테이블의 전체 데이터 또는 파티션드 테이블의 한 파티션의 전체 데이터가 저장될 수 있다. 테이블 생성시 Altibase는 테이블스페이스에 테이블 세그먼트를 할당한다.
Index 세그먼트	한 개의 인덱스 세그먼트에는 한 인덱스의 모든 데이터 또는 파티션드 인덱스의 한 파티션의 데이터가 저장될 수 있다. 인덱스 생성시 Altibase는 테이블스페이스에 인덱스 세그먼트를 할당한다.
Undo 세그먼트	데이터베이스의 변경을 발생시키는 트랜잭션에 의해 사용된다. Altibase는 테이블 또는 인덱스를 변경하기 전에 변경 전 값 (즉, before-image)을 언두 세그먼트에 저장해 두어, 트랜잭션 롤백시에 변경을 언두할 수 있다.
TSS 세그먼트	Altibase 내부적으로 관리되는 TSS (Transaction Status Slot)를 관리하기 위한 세그먼트이며, 시스템 언두 테이블스페이스 내에 할당된다.

[표 6-1] 세그먼트 종류

각 세그먼트는 내부적으로 프리 (Free) 익스텐트 리스트와 풀 (Full) 익스텐트 리스트를 관리한다. 프리 익스텐트가 부족하면 테이블스페이스에 익스텐트 추가를 요청한다.

익스텐트(Extent)

디스크 테이블스페이스에서 데이터 오브젝트를 저장하기 위해서 필요한 자원으로 연속된 여러 페이지를 할당하는 단위이다. 데이터를 저장할 때 저장 가능한 프리 페이지 (Free Page)가 부족하면, 테이블스페이스에서 익스텐트 단위로 페이지를 할당받는다.

한 개의 익스텐트는 기본 64개의 페이지 (512KB)로 구성된다. Altibase는 테이블스페이스마다 익스텐트 크기를 다르게 정할 수 있도록 지원한다.

페이지(page)

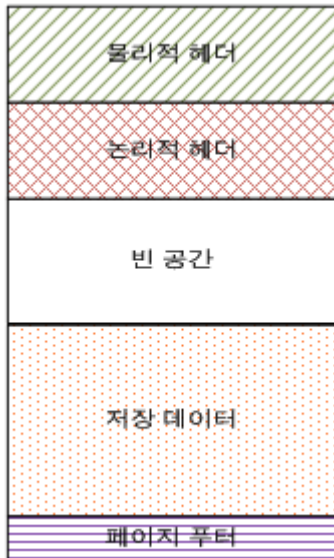
테이블과 인덱스의 레코드가 저장되는 최소 단위를 페이지라고 한다. 또한 I/O의 최소 단위이다. Altibase의 페이지 크기는 8KB이다. (Altibase는 다양한 페이지 크기 (multiple page size)의 사용을 지원하지 않는다.)

페이지에 어떤 정보를 저장하느냐 따라 데이터 페이지, 인덱스 페이지, 언두 페이지 등 여러 종류의 페이지가 있다.

페이지의 일반적인 구조와 데이터 저장 방식을 살펴보면 다음과 같다.

페이지 구조

페이지는 페이지의 기본 정보와 free slot 등을 관리하기 위한 헤더를 가진다. 레코드는 헤더를 제외한 영역에 저장된다. 페이지는 내부적으로 다음 그림과 같이 5개 영역으로 나뉘어진다.



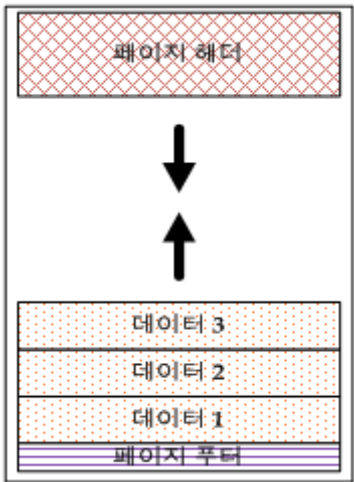
[그림 6-4] 디스크 테이블스페이스의 페이지 구조

- 물리적 헤더 (Physical Header)
모든 데이터 페이지에 공통되는 정보를 가지고 있다.
- 논리적 헤더 (Logical Header)
페이지의 종류에 따라 필요한 정보를 가지고 있다.
- 빈 공간 (Free Space)
새로운 데이터를 저장된다.
- 저장 데이터 (Stored Procedure)
페이지 종류에 따라 로우, 인덱스, 언두 레코드 등이 저장된다.
- 페이지 푸터 (Page Footer)
페이지의 가장 아래쪽에 위치하며, 페이지의 무결성을 검사하기 위한 정보를 가지고 있다.

페이지 레코드 저장 방식

레코드는 페이지의 아래쪽에서 위쪽 (페이지의 시작) 방향으로 채워지며 빈 공간 영역에 저장된다.

페이지의 논리적 헤더는 페이지 아래 방향으로 확장되어 저장된다. 그 크기는 가변적이다.



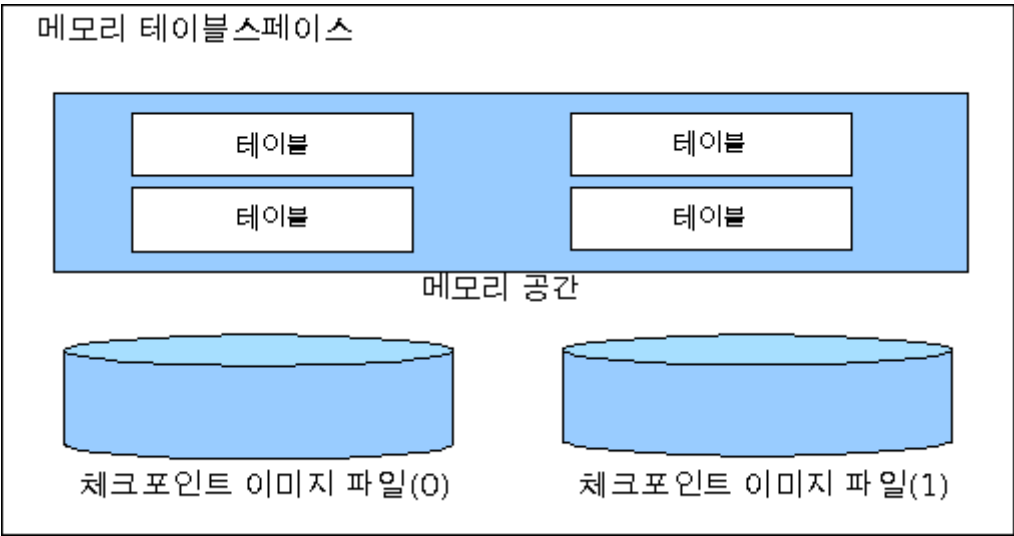
[그림 6-5] 페이지 레코드 저장 방식

메모리 테이블스페이스 구조

메모리 테이블스페이스는 모든 데이터가 메모리 공간에 저장되는 테이블스페이스이다. 물리적 구조는 체크포인트 이미지 파일로 구성되며, 논리적으로는 페이지와 페이지 리스트들로 구성된다.

물리적 구조

메모리 테이블스페이스는 체크포인트 이미지 파일과 밀접한 관계를 갖는다. 다음 그림에서는 메모리 테이블스페이스, 테이블 및 체크포인트 이미지 파일의 연관 관계를 설명한다.



[그림 6-6] 메모리 테이블스페이스, 테이블 및 체크포인트 이미지 파일의 연관 관계 메모리 테이블스페이스,

테이블 및 체크포인트는 다음의 특징을 갖는다.

메모리 테이블스페이스는 디스크 테이블스페이스와 달리 데이터를 데이터 파일에 저장하지 않고, 선형적인 메모리 공간에 저장한다. 선형적인 메모리 공간은 페이지 단위로 분할되고, 이 페이지들의 리스트가 테이블을 구성한다. 디스크 테이블스페이스는 디스크 입출력 비용 및 대용량 테이블 관리를 위하여 페이지 단위가 아닌 익스텐트 단위로 관리한다. 세그먼트는 개념적으로 익스텐트의 리스트를 관리하기 논리적인 단위이다.

그러나 메모리 테이블스페이스의 목적은 대용량 데이터의 관리보다 빠른 접근을 지원하는 것이기 때문에, 세그먼트나 익스텐트의 개념이 필요하지 않다. 따라서 메모리 테이블스페이스의 테이블들은 페이지 리스트를 이용하여 관리된다.

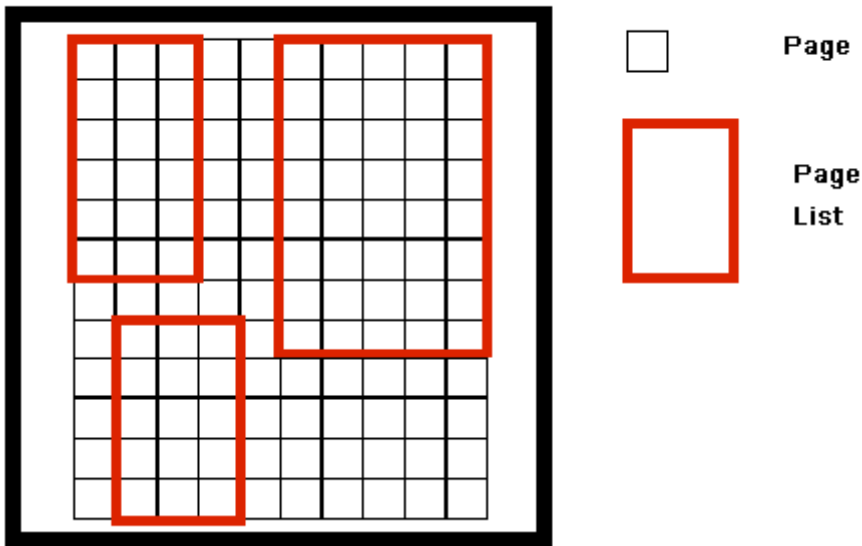
메모리 테이블들은 체크포인트시에 물리적으로 체크포인트 이미지 파일에 저장된다. 체크포인트 이미지 파일의 용도는 디스크 테이블스페이스의 데이터 파일의 그것과는 다르다. 디스크 테이블스페이스의 데이터 파일은 객체들을 저장하기 위한 것인 반면, 메모리 테이블스페이스의 체크포인트 이미지 파일은 객체들을 백업하기 위한 것이다. 체크포인트 이미지 파일은 데이터베이스 운영에 직접적으로 필요하지 않다. 하지만 백업 및 복구 시간을 단축하기 위해서는 반드시 필요하다.

체크포인트 시 메모리 공간의 페이지들은 운영 체제에 의해 지원되는 파일에 저장된다. Altibase의 체크포인트 방식은 일명 “핑퐁 (ping-pong) 체크포인트팅”으로, 이는 두 별의 체크포인트 이미지 파일 (0번, 1번)을 유지하며, 체크포인트 발생시마다 0번과 1번 파일을 번갈아가며 사용한다. 또한 각 체크포인트 이미지 파일은 디스크 입출력 비용의 분산을 목적으로 다수의 작은 파일로 분리될 수 있다.

논리적 구조

메모리 테이블스페이스의 논리적 구성 요소에는 페이지 리스트와 페이지가 있다. 이들의 관계를 살펴보면 다음 그림과 같다.

Memory Tablespace



[그림 6-7] 메모리 테이블스페이스의 논리적 구조

페이지 리스트 (Page List)

페이지 리스트는 메모리 테이블스페이스 내에서 테이블을 구성하는 논리적인 개념이다. 페이지 리스트는 메모리 테이블스페이스의 메모리 공간을 분할한 단위인 페이지의 리스트이다.

메모리 테이블스페이스 객체 중에서 테이블은 페이지 리스트로 유지된다. 인덱스는 데이터베이스의 일관성을 유지하는 대상에 포함되지 않기 때문에 페이지 리스트를 사용하지 않는다. 시스템을 다시 시작할 때 메모리 테이블의 인덱스는 재구축 되는데, 이는 운영중에 인덱스 로깅을 함으로써 발생할 수 있는 부하를 제거한다.

페이지 (Page)

메모리 테이블스페이스의 페이지 구조 및 데이터 저장 방식은 디스크 테이블스페이스의 페이지와 다른 특징을 갖는다.

메모리 테이블스페이스는 디스크 테이블스페이스와 달리 디스크 입출력 비용을 고려할 필요가 없기 때문에 레코드 수정 방식으로 아웃 플레이스 갱신 (out-place update)을 사용한다.

아웃 플레이스 갱신이란 기존 레코드의 이미지를 직접적으로 변경하지 않고, 레코드의 새로운 버전을 위한 공간을 할당받아 처리하는 방식이다. 이러한 갱신 방식은 기존 레코드의 삭제와 새로운 레코드의 삽입 과정으로 이루어지기 때문에 기존 레코드를 재구성하는 비용이 들지 않는다. 또한 기존 레코드의 접근을 직접 할 수 있어 동시성 레벨이 높은 응용 분야에서 빠른 성능을 보장한다.

휘발성 테이블스페이스 구조

휘발성 테이블스페이스의 구조는 모든 데이터가 메모리 공간에 저장되는 메모리 테이블스페이스와 동일하다. 그러나 디스크상의 체크포인트 이미지 파일을 가지지 않는다는 점에서 메모리 테이블스페이스와 차이가 있다. 휘발성 테이블스페이스의 데이터는 메모리에만 상주한다.

휘발성 테이블스페이스에서 일어나는 작업들은 디스크 로깅 작업을 수반하지 않고 체크포인트 대상에서도 제외되기 때문에 디스크 입출력이 전혀 없다. 따라서 빠른 성능을 필요로 하는 경우에 휘발성 테이블스페이스가 유용하다. 논리적으로 페이지 리스트와 페이지로 구성된다.

물리적 구조

휘발성 테이블스페이스는 메모리에 데이터베이스 객체를 상주시킨다는 점에서 메모리 테이블스페이스와 동일하다. 그러나 휘발성 테이블스페이스는 체크포인트 이미지 파일을 갖지 않는다.

논리적 구조

메모리 테이블스페이스와 동일하게 페이지 리스트와 페이지로 구성된다.

테이블스페이스 분류

Altibase가 제공하는 테이블스페이스는 아래 3가지 기준에 의해서 분류된다. 하나의 테이블스페이스는 아래에 분류된 다수의 속성들을 동시에 가질 수 있다.

- 저장 공간에 따른 분류
- 저장 내용에 따른 분류
- 생성 주체에 따른 분류

저장 공간에 따른 분류

Altibase 테이블스페이스는 저장 공간에 따라 다음과 같이 분류된다.

- 메모리 상주 테이블스페이스
- 디스크 테이블스페이스

메모리 상주 테이블스페이스

메모리 상주 테이블스페이스는 로깅 수행 여부 및 디스크 이미지 파일의 존재 여부에 따라 메모리 테이블스페이스와 휘발성 테이블스페이스로 구분된다.

메모리 테이블스페이스는 메모리 기반 객체를 저장하기 위한 테이블스페이스이다. 해당 테이블스페이스 내에 저장되는 모든 객체에 메모리 기반 데이터베이스 기술이 적용됨으로써, 사용자가 실시간으로 데이터에 접근할 수 있다. 그러나 메모리

테이블스페이스의 크기는 시스템의 사용 가능한 물리적 메모리 공간을 초과할 수 없다.

휘발성 테이블스페이스는 디스크 I/O 작업 없이 메모리 기반 객체를 저장하는 테이블스페이스이다. 해당 테이블스페이스 내에 저장되는 모든 객체에 메모리 기반 데이터베이스 기술과 부가적 기술이 적용됨으로써, 사용자가 디스크 I/O 작업 없이 실시간으로 데이터에 접근할 수 있다. 그러나 휘발성 테이블스페이스의 크기는 시스템의 사용 가능한 물리적 메모리 공간을 초과할 수 없고, 데이터베이스 서버 종료시 모든 휘발성 데이터 객체들은 사라진다.

디스크 테이블스페이스

디스크 테이블스페이스는 디스크 기반 객체를 저장하기 위한 테이블스페이스이다. 데이터의 실시간 접근보다는 대용량 데이터를 관리하고 싶은 경우에 사용할 수 있는 테이블스페이스이다. 해당 테이블스페이스에 저장되는 객체들에 대한 접근은 디스크 입출력을 수반한다. 이러한 디스크 입출력 비용이 데이터 접근 시간의 대부분을 차지하기 때문에, 디스크 테이블스페이스는 디스크 입출력 비용을 줄이기 위해서 메모리 버퍼 공간을 사용한다.

저장 내용에 따른 분류

테이블스페이스에 저장되는 내용에 따라 다음과 같이 분류된다.

- 딕셔너리 테이블스페이스 (Dictionary Tablespace)
- 언두 테이블스페이스 (Undo Tablespace)
- 임시 테이블스페이스 (Temporary Tablespace)
- 데이터 테이블스페이스 (Data Tablespace)

딕셔너리 테이블스페이스

딕셔너리 테이블스페이스는 데이터베이스 시스템의 운영상 필요한 메타 데이터를 저장하기 위한 테이블스페이스다. 데이터베이스 생성시 시스템에 의해서 생성되는 테이블스페이스이며, 데이터베이스 내에 하나만 존재한다. 사용자는 딕셔너리 테이블스페이스 내에 객체를 생성할 수 없으며, 시스템만이 메타 데이터 유지 관리를 위한 시스템 객체를 생성할 수 있다. 메타 데이터에 대한 빠른 접근을 위하여 딕셔너리 테이블스페이스는 메모리에 존재한다. 딕셔너리 테이블스페이스가 붕괴(crash)된 경우에는 전체 데이터베이스를 운영할 수 없다(백업 및 매체 복구를 이용하여 데이터베이스를 복구시켜야 한다).

언두 테이블스페이스

언두 테이블스페이스는 디스크 객체에 대한 연산이 남긴 언두 이미지(undo image)를 저장하기 위한 테이블스페이스이다. Altibase의 동시성 제어는 MVCC(Multi-Version

Concurrency Control) 기법이기 때문에 변경 이전의 이미지를 저장할 공간이 필요하다. 이러한 이전 이미지가 언두 테이블스페이스에 저장된다.

언두 테이블스페이스는 시스템에 하나만 존재하며, 데이터베이스 내의 모든 디스크 테이블스페이스에 의해 공유된다. 따라서, 언두 테이블스페이스도 딕셔너리 테이블스페이스와 마찬가지로 시스템 운영상 필수적인 시스템 테이블스페이스이다. 테이블스페이스 단위의 백업이 가능하다.

임시 테이블스페이스

임시 테이블스페이스는 질의 수행중 생성되는 임시 결과를 저장하기 위한 테이블스페이스이다. 따라서, 트랜잭션이 종료하는 시점에 해당 질의가 남긴 임시 테이블스페이스 내의 모든 데이터들은 사라지게 된다.

이러한 종류의 테이블스페이스는 동시성 제어 및 회복을 위한 로깅 등을 하지 않아 빠른 저장 및 검색이 가능하다. 사용자가 임의로 임시 테이블스페이스를 생성할 수 있으며, 다수의 임시 테이블스페이스가 시스템 내에 존재할 수 있다. 임시 테이블스페이스의 백업은 지원되지 않는다.

데이터 테이블스페이스

데이터 테이블스페이스는 사용자 정의 객체를 저장하기 위한 테이블스페이스이다. 다수의 데이터 테이블스페이스가 시스템 내에 존재할 수 있으며, 사용자는 테이블스페이스에 저장되는 데이터의 특성에 따라 메모리, 휘발성 또는 디스크 테이블스페이스로 생성할 수 있다.

생성 주체에 따른 분류

Altibase의 테이블스페이스는 생성 주체에 따라 다음과 같이 분류된다.

- 시스템 테이블스페이스 (System Tablespace)
- 사용자 정의 테이블스페이스 (User-defined Tablespace)

시스템 테이블스페이스

시스템 테이블스페이스는 시스템 운영상 필요한 데이터들을 저장하기 위한 테이블스페이스이다. 시스템 딕셔너리 테이블스페이스, 시스템 언두 테이블스페이스, 시스템 데이터 테이블스페이스 및 시스템 임시 테이블스페이스가 이에 해당한다. 시스템 테이블스페이스는 데이터베이스 생성시 만들어지며, 사용자가 명시적으로 테이블스페이스 삭제하거나 이름을 변경할 수 없다. 또한 테이블스페이스 단위의 백업과 매체 복구 수행이 가능하다.

사용자 정의 테이블스페이스

사용자 정의 테이블스페이스는 사용자 정의 객체들의 내용을 저장하기 위한 테이블스페이스이다. 사용자 정의 테이블스페이스 내에 정의된 객체들의 메타 데이터는 디렉터리 테이블스페이스에 저장된다. 사용자는 명시적으로 사용자 정의 테이블스페이스를 삭제하거나 이름을 변경할 수 있다. 또한 테이블스페이스 단위의 백업 및 복구 수행이 가능하다.

테이블스페이스 목록

데이터베이스가 생성시 다수의 테이블스페이스가 만들어진다. [표 5-2]와 같이 시스템 테이블스페이스, 언두 테이블스페이스, 임시 테이블스페이스, 그리고 사용자가 이용할 수 있는 기본적인 메모리 테이블스페이스와 디스크 테이블스페이스가 생성된다. 추가로 사용자가 'CREATE TABLESPACE'문으로 테이블스페이스들을 추가할 수 있다.

ID	테이블스페이스 종류	저장 공간	테이블스페이스 이름	생성 시점
0	SYSTEM DICTIONARY TABLESPACE	메모리	SYS_TBS_MEM_DIC	CREATE DATABASE
1	SYSTEM MEMORY DEFAULT TABLESPACE	메모리	SYS_TBS_MEM_DATA	CREATE DATABASE
2	SYSTEM DISK DEFAULT TABLESPACE	디스크	SYS_TBS_DISK_DATA	CREATE DATABASE
3	SYSTEM UNDO TABLESPACE	디스크	SYS_TBS_DISK_UNDO	CREATE DATABASE
4	SYSTEM DISK TEMPORARY TABLESPACE	디스크	SYS_TBS_DISK_TEMP	CREATE DATABASE
5이상	USER MEMORY DATA TABLESPACE	메모리	사용자 지정	CREATE MEMORY DATA TABLESPACE
5이상	USER DISK DATA TABLESPACE	디스크	사용자 지정	CREATE DISK DATA TABLESPACE
5이상	USER DISK TEMPORARY TABLESPACE	디스크	사용자 지정	CREATE DISK TEMPORARY TABLESPACE

ID	테이블스페이스 종류	저장 공간	테이블스페이스 이름	생성 시점
5이상	USER VOLATILE DATA TABLESPACE	메모리	사용자 지정	CREATE VOLATILE DATA TABLESPACE

[표 6-2] 테이블스페이스 목록

디스크 테이블스페이스

디스크 테이블스페이스는 모든 데이터가 디스크 공간에 저장되는 테이블스페이스이다. 이 절에서는 디스크의 데이터 페이지를 중심으로 구조 및 로우 데이터의 입력 방식에 대해 살펴본다.

데이터 페이지 구조

Altibase가 데이터베이스의 저장 공간을 관리할 때 사용하는 데이터의 최소 단위를 페이지 (page)라고 한다. 페이지 크기는 8KB이고, 다양한 크기의 페이지 (multiple page size)는 지원되지 않는다.

데이터 페이지 (data page)는 여러 페이지 종류들 중의 한가지로, 로우 데이터 (row data)를 저장한다. 로우 데이터는 페이지 아래부터 채워가며 저장되며, 이 때 빈 공간 영역을 사용한다. 만약 빈 공간의 영역이 충분하지 않다면, 페이지 콤팩트 (page compact) 연산을 수행하여 단편화된 공간을 제거하고 연속된 빈 공간을 확보하도록 한다.



[그림 6-8] 데이터 페이지의 구조

데이터 페이지의 영역은 위의 그림과 같이 6개의 영역으로 구성된다.

- 물리적 헤더 (Physical Header)
이 영역은 페이지 종류에 상관없이 모든 데이터 페이지들에 공통되는 정보를 가지고 있다.
- TTL (Touched Transaction Layer)
이 영역은 MVCC (Multi-Version Concurrency Control) 관련 정보를 가지고 있다.
- 슬롯 디렉토리 (Slot Directory)
이 영역은 로우가 저장된 페이지 내에서의 위치 (offset)에 대한 정보를 가지고 있다.
- 빈 공간 (Free Space)
이 영역은 입력이나 갱신 등의 연산을 할 때 사용할 수 있는 여유 공간이다.
- 로우 데이터 (Row Data)
- 페이지 푸터 (Page Footer)
이 영역은 페이지 구조의 가장 아래쪽에 위치하며, 페이지의 무결성을 확인하기 위한 정보를 가지고 있다.

디스크 테이블스페이스의 공간 관리

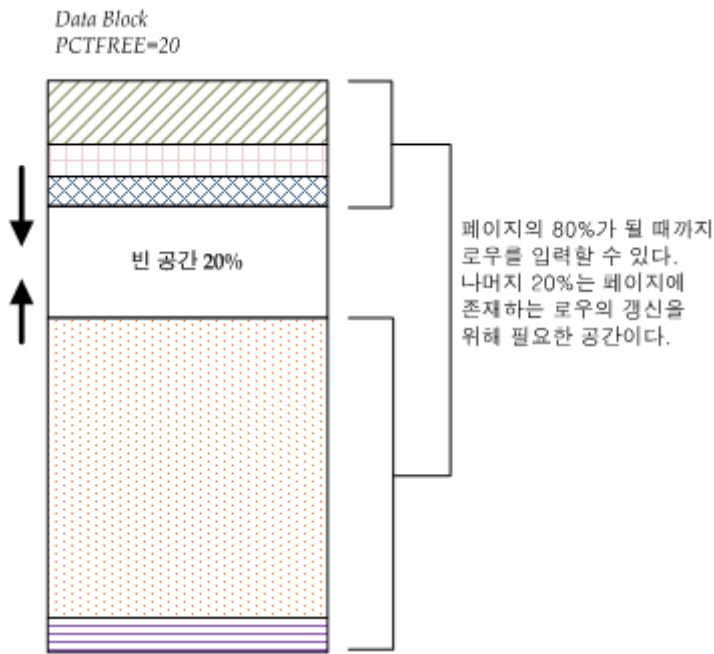
디스크 테이블스페이스는 PCTFREE와 PCTUSED 파라미터를 이용하여 수동적으로 관리될 수 있다.

PCTFREE와 PCTUSED 파라미터를 사용해서 로우 데이터에 대한 입력이나 갱신 연산을 할 때 빈 공간의 사용을 제어할 수 있다. 이들 파라미터의 값은 `altibase.properties` 파일의 PCTFREE와 PCTUSED 프로퍼티의 값으로 지정된다. 또한 테이블 생성(CREATE TABLE...) 또는 변경(ALTER TABLE...) 구문에서 테이블 별로 파라미터 값을 명시할 수도 있다.

PCTFREE

PCTFREE는 페이지에 저장되어 있는 로우들이 갱신될 경우에 대비하여 미리 확보해두는 빈 공간의 최소 비율이다.

예를 들어 PCTFREE 값을 20으로 설정하면, 페이지의 80% 공간까지만 입력 (insert)할 수 있고, 나머지 20%의 공간은 기존의 로우들이 갱신될 때 사용을 위해서 남겨둔다.

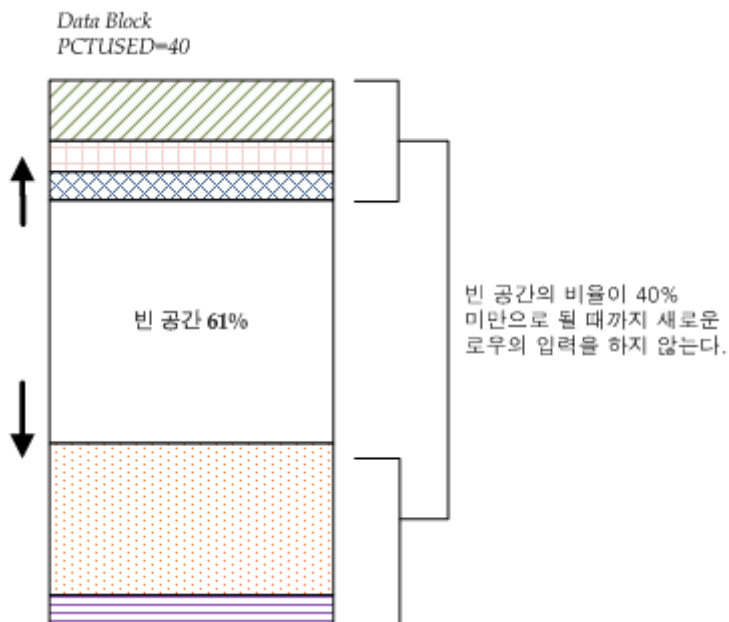


[그림 6-9] PCTFREE 와 페이지 구조

PCTUSED

PCTUSED는 페이지가 갱신만 가능한 상태에서 다시 삽입이 가능 상태로 가기 위해서 감소해야 할 페이지 내 사용 공간의 최소 비율이다.

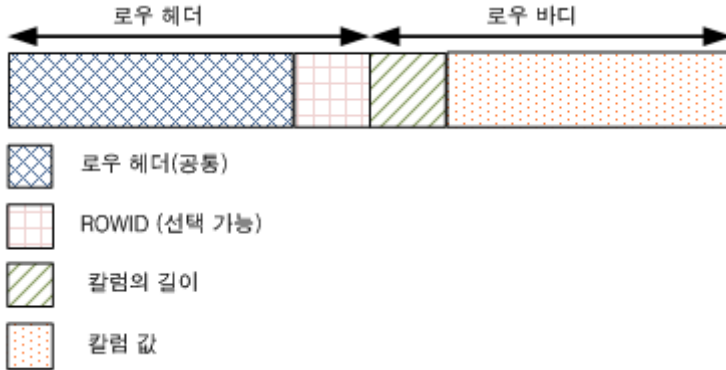
PCTFREE 제한에 걸리게 되면, 해당 페이지에는 사용 공간의 비율이 PCTUSED 보다 낮아지기 전까지 새로운 로우를 입력할 수 없고, 이 페이지 내의 빈 공간은 오직 갱신 연산을 위해서만 사용된다. 이 상태는 사용 공간의 비율이 PCTUSED값 아래로 떨어질 때까지 지속된다.



[그림 6-10] PCTUSED 와 페이지 구조

로우의 구조

로우는 하나 이상의 로우 조각 (row piece)들로 구성된다. 만약 로우 전체가 한 개의 페이지에 저장될 수 있다면, 로우는 하나의 로우 조각으로 저장된다. 그러나 로우 전체를 한 개의 페이지에 저장할 수 없다면, 로우는 여러 개의 로우 조각에 나뉘어서 저장된다. 이들 로우 조각들은 ROWID값에 의해 서로 연결된다 (chained).



[그림 6-11] 로우 조각의 구조

로우 조각은 로우 헤더 (row header)와 로우 바디(row body)로 구성된다.

로우 헤더에는 18 byte 크기의 헤더 정보가 저장된다. 연결된 로우 조각 (chained row piece)일 경우에는 6 byte의 ROWID 정보가 추가적으로 저장된다.

로우 바디에는 칼럼의 길이 (column length), 칼럼 값 (column value)이 쌍을 이뤄서 연속으로 저장된다. 칼럼 값의 길이가 250 byte 이하이면 칼럼 길이의 저장을 위해서 1byte만 필요하고, 칼럼 값의 길이가 250 byte를 초과하면 칼럼 길이의 저장을 위해서 3byte가 필요하다.

공간을 절약하기 위해서 칼럼 값이 널 (NULL)인 경우 칼럼의 길이 (0)만 저장하고 칼럼 값은 저장하지 않는다. 또한 칼럼 값이 널인 칼럼들이 마지막에 연속으로 올 경우에는 칼럼 값뿐 아니라 칼럼 길이도 저장하지 않는다.

칼럼은 테이블 생성 (CREATE TABLE...) 구문에서 나열한 순서대로 저장된다. 이 때 널을 많이 포함하는 칼럼을 마지막에 배치하면 로우를 저장하는데 필요한 공간을 절약할 수 있다.

로우 체이닝 및 마이그레이션

로우의 데이터가 너무 커서 한 개의 페이지에 저장할 수 없을 때 로우 체이닝 (row chaining)과 로우 마이그레이션 (row migration)이 발생한다.

로우 체이닝은 데이터를 입력할 때 데이터의 크기가 너무 커서 로우가 한 페이지에 저장될 수 없을 때 발생한다. 이 경우에는 로우의 데이터가 여러 개의 페이지에

나누어 저장되고, 이들은 서로 ROWID에 의해 연결된다.

로우 마이그레이션은 한 페이지 내에 저장되었던 로우가 갱신 과정에서 로우의 크기가 페이지의 크기를 넘어가는 경우 발생한다. 이 경우 전체 로우는 새로운 페이지로 마이그레이션 되고, 원래 로우는 옮겨진 로우가 저장된 새로운 위치를 가리키게 된다. 그러나 로우 마이그레이션이 발생하더라도 ROWID는 변경되지 않는다.

로우 체이닝 또는 로우 마이그레이션이 발생하면, DML 처리시에 한 페이지를 더 읽어야 하므로 디스크 I/O로 인한 성능저하가 발생한다.

언두 테이블스페이스

언두 테이블스페이스는 데이터베이스에 대한 변경 연산을 철회 (rollback)하는데 필요한 정보를 저장하는 테이블스페이스이다. Altibase는 다중 버전의 동시성 제어 기법 (MVCC)을 사용하기 때문에 변경 이전의 이미지를 저장할 공간이 필요하다.

언두 테이블스페이스는 데이터베이스에 하나만 존재하며, 데이터베이스 내의 모든 디스크 테이블스페이스에 의해 공유된다.

이 절에서는 언두 테이블스페이스의 특징 및 크기 계산 등 언두 테이블스페이스를 어떻게 관리하는지 설명한다.

- 언두 레코드 (Undo Record)
- 언두 테이블스페이스의 특징
- 트랜잭션 세그먼트의 관리
- 세그먼트 공간 재사용
- 언두 테이블스페이스 변경

언두 레코드

데이터베이스는 변경된 트랜잭션을 취소(롤백 또는 언두)하기 위하여 관련 정보들을 유지해야 한다. 이러한 정보들은 주로 트랜잭션이 커밋되기 전에 언두 레코드들로 저장된다.

언두 레코드는 다음과 같은 목적으로 사용된다.

- 트랜잭션 철회 (rollback)
- 데이터베이스 복구
- 읽기 일관성 (Read Consistency) 보장

롤백 구문이 수행되면, 언두 레코드는 커밋되지 않은 트랜잭션에 의한 데이터베이스 변경을 취소하기 위해 사용된다.

또한 언두 레코드는 데이터베이스를 복구하는 동안에도 사용된다. 로그 파일에 기반한 트랜잭션 리두 (redo)에 의해 데이터베이스를 복원한 후, 언두 레코드는 커밋되지 않은 변경에 대해서 취소하기 위해서 사용된다.

그리고 다른 트랜잭션에 의해 변경중인 레코드를 어떤 트랜잭션이 읽을 때, 두 트랜잭션이 동시에 레코드에 접근하여도 레코드가 변경되기 전의 이미지는 언두 레코드에 저장되어 있기 때문에 읽기 일관성을 보장할 수 있다.

언두 테이블스페이스의 특징

언두 테이블스페이스의 특징을 살펴보면 다음과 같다.

- 시스템에 의해 자동으로 관리된다.
- 기본 언두 테이블스페이스 파일은 자동 확장 모드의 undo001.dbf 이다. 데이터 파일의 추가 및 크기 변경이 가능하다.
- 온라인 백업 (Online Backup)의 대상이다.
- TSS 세그먼트와 언두 세그먼트 이외의 데이터베이스 객체는 언두 테이블스페이스에 생성이 불가능하다.
- 언두 테이블스페이스는 시스템 테이블스페이스이므로, 테이블스페이스 오프라인 및 제거가 불가능하다.
- 서버가 재구동될 때마다 언두 테이블스페이스는 재구성 (Reset)된다.

Altibase는 언두 테이블스페이스의 정보 및 공간을 관리할 때 시스템에 의한 관리 방식을 사용한다. 시스템에 의한 관리 방식이란 기본적으로 언두 테이블스페이스의 세그먼트들과 공간들을 서버가 자동으로 관리하는 것을 의미한다.

언두 테이블스페이스는 데이터베이스 생성 과정에서 생성된다. 언두 테이블스페이스는 시스템 테이블스페이스로써, 데이터베이스내에 하나만 존재할 수 있다. 만약 언두 테이블스페이스가 존재하지 않는다면 부트 로그에 에러 메시지가 출력되고 서버 구동이 실패한다.

언두 테이블스페이스내에서는 트랜잭션 세그먼트 (TSS 세그먼트와 언두 세그먼트)가 관리된다. 사용자는 프로퍼티 TRANSACTION_SEGMENT_COUNT를 사용해서 트랜잭션 세그먼트의 개수를 변경할 수 있다. 사용자가 프로퍼티에서 지정한 개수만큼 TSS 세그먼트와 언두 세그먼트가 각각 생성된다. TRANSACTION_SEGMENT_COUNT 프로퍼티를 255로 설정하였다면, 서버 구동시마다 TSS 세그먼트 255개와 언두 세그먼트 255개가 생성된다.

프로퍼티 파일내에서 트랜잭션 세그먼트를 다른 개수로 변경하였다면, 다음 서버 구동시에 명시된 개수만큼 세그먼트들이 생성될 것이다.

트랜잭션 세그먼트의 관리

트랜잭션 세그먼트란 디스크 변경 트랜잭션에 반드시 필요한 한 개의 TSS 세그먼트와 한 개의 언두 세그먼트로 구성된다.

한 트랜잭션 세그먼트는 한 디스크 변경 트랜잭션에 바인딩 (Binding) 되고, 그 트랜잭션이 완료될 때 언바인딩 (Unbinding) 되기 때문에 다른 트랜잭션에 의해 동시에 공유될 수 없다.

V\$TXSEGS를 조회하면, 트랜잭션 세그먼트의 바인딩 여부를 확인할 수 있다. 디스크 변경 트랜잭션에 해당 트랜잭션 세그먼트가 바인딩되면 V\$TXSEGS에 트랜잭션 세그먼트 ID, 트랜잭션 ID에 해당하는 레코드가 생성되고 바인딩이 해제되면 레코드는 삭제된다.

또한 TSS 세그먼트와 언두 세그먼트의 공간은 트랜잭션이 한 번 사용한 공간에 대해서는 어느 정도 시간이 지나면 재사용할 수 있는 구조로 설계되었다. 따라서 언두 트랜잭션의 공간이 필요한 경우에는 무조건 세그먼트를 생성하여 공간을 확장하는 것이 아니라 기간이 만료된 세그먼트가 다시 사용된다.

TSS 세그먼트의 재사용 단위는 1M이며, 언두 세그먼트는 2M이다.

다음은 언두 테이블스페이스와 관련된 사용자 프로퍼티를 나타낸다.

- SYS_UNDO_FILE_INIT_SIZE
언두 테이블스페이스의 데이터 파일 생성시 초기 크기
- SYS_UNDO_FILE_MAX_SIZE
언두 테이블스페이스의 데이터 파일 최대 크기
- SYS_UNDO_TBS_NEXT_SIZE
언두 테이블스페이스의 데이터 파일 자동 확장 크기
- SYS_UNDO_TBS_EXTENT_SIZE
언두 테이블스페이스 한 익스텐트의 페이지 개수
- TRANSACTION_SEGMENT_COUNT
트랜잭션 세그먼트의 개수

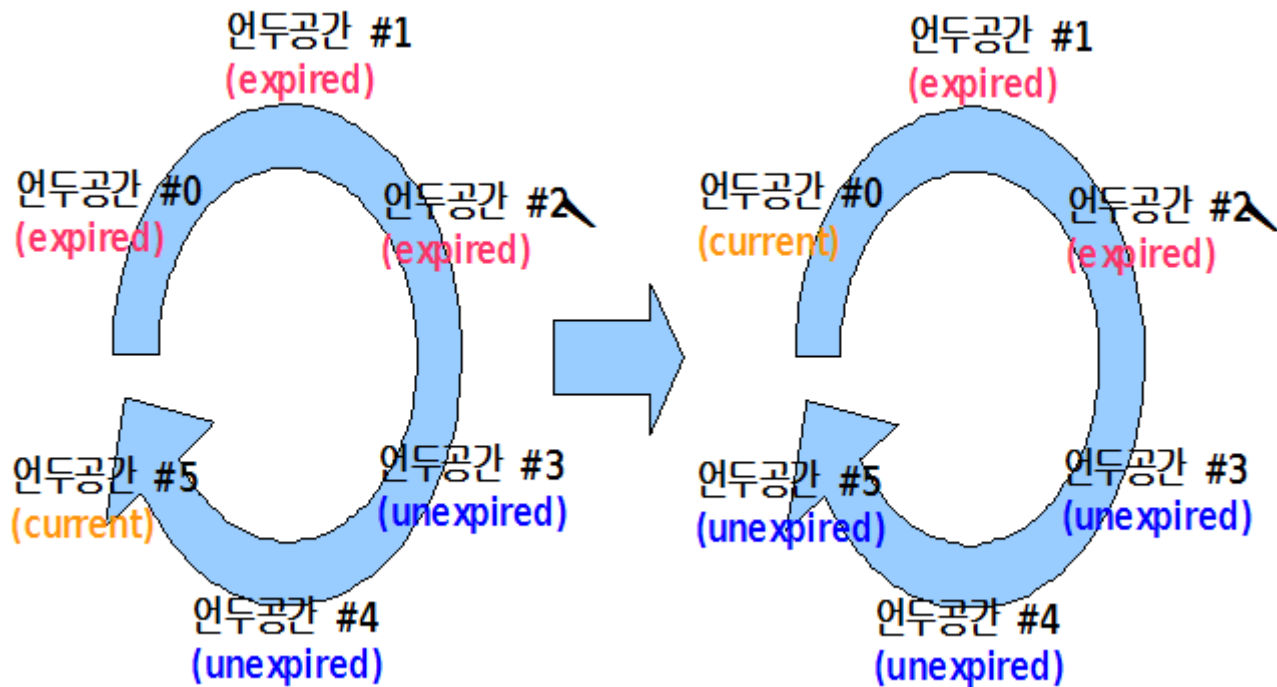
세그먼트의 공간 재사용

트랜잭션 커밋 후에 언두 데이터는 트랜잭션 롤백이나 복구를 목적으로 더 이상 필요하지 않다. 하지만 트랜잭션의 커밋 주기가 긴 Long-Term 트랜잭션은 읽기 일관성을 위해서 언두 데이터에 의존하고 있는 레코드의 이전 버전이 필요하다. 그렇지만 어느 정도 시간이 지나면 읽기 일관성을 위해서도 더 이상 언두 데이터는 필요하지 않게 된다.

따라서 Altibase 데이터베이스는 커밋된 언두 데이터라고 하여도 최소한의 기간 동안만 유지하고, 그 기간이 지나면 그 언두 데이터가 차지했던 공간을 다른

트랜잭션이 재사용할 수 있도록 하고 있다.

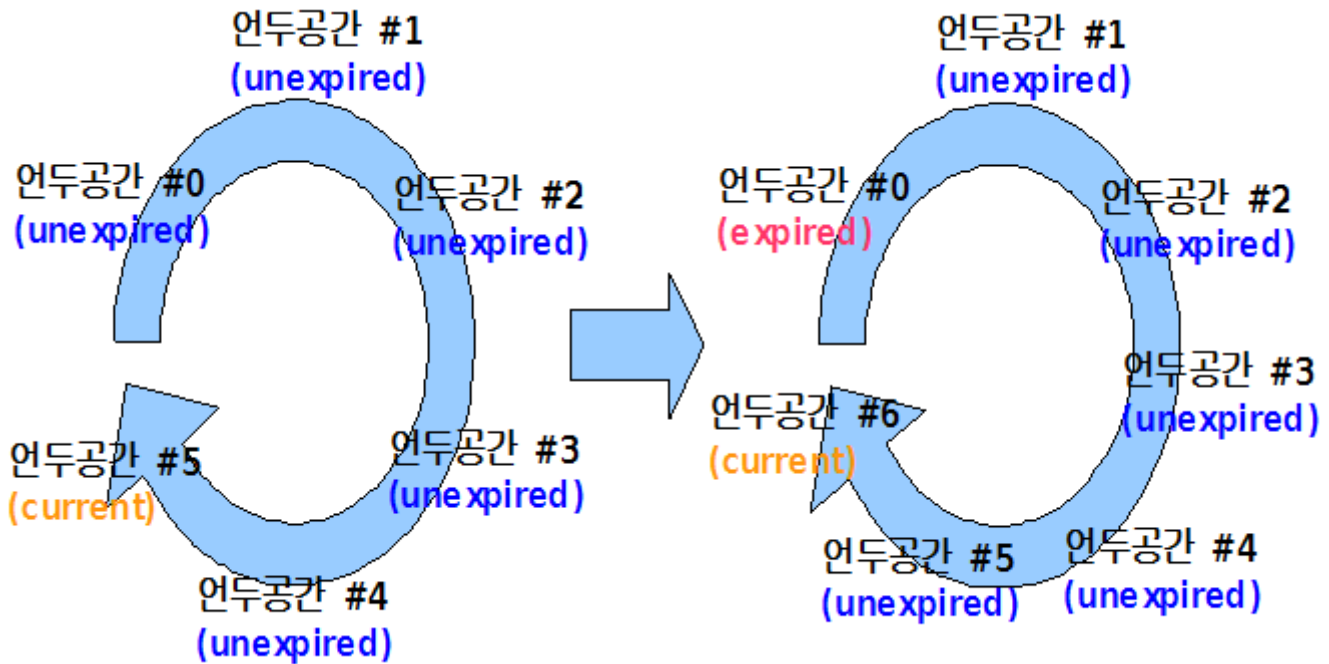
만약 커밋된 트랜잭션을 위한 언두 데이터를 가지고 있는 공간에 접근하는 온라인 트랜잭션들이 더 이상 존재하지 않는다면, 해당 언두 공간은 기간이 만료 (Expired)되었다고 한다. 반대로 그 언두 공간에 접근이 가능한 온라인 트랜잭션이 아직 존재한다면, 해당 언두 공간은 기간이 유효 (Unexpired)하다고 한다. 기간이 만료된 언두 공간은 다른 트랜잭션에 의해서 재사용 될수 있으나, 기간이 유효한 언두 공간은 재사용될 수 없다.



[그림 6-12] 언두 세그먼트의 재사용

위 그림은 언두 세그먼트의 순환 구조가 언두 공간의 재사용을 어떻게 허락하는지를 보여준다.

그림은 언두 공간 #0부터 시작해서 순서대로 사용되면서 현재(Current)의 언두 공간 #5을 사용하고 있는 것을 나타낸다. 그리고 다음 차례의 언두 공간 #0이 만료된 것을 확인하고, 언두 공간 #5를 모두 사용하면 언두 세그먼트를 더 이상 확장하지 않고 언두 공간 #0을 재사용한다.



[그림 6-13] 언두 세그먼트의 확장

그러나 언두 공간 #0이 유효한 상태라면 위의 그림처럼 언두 세그먼트는 익스텐트를 확장하여 언두 공간 #6을 추가하게 된다.

이와 같은 세그먼트 공간의 재사용성은 TSS 세그먼트에도 동일하게 적용된다.

언두 테이블스페이스 변경

언두 테이블스페이스는 ALTER TABLESPACE 구문을 사용하여 변경될 수 있다. 그러나 언두 테이블스페이스는 대부분이 시스템에 의해서 관리되므로, 다음과 같은 연산들에 대해서만 사용자가 수행할 수 있다.

- 데이터 파일 추가 및 제거
- 데이터 파일 크기 확장 및 축소
- 데이터 파일의 온라인 백업 시작 및 완료

언두 테이블스페이스에 용량 부족 또는 용량 부족과 관련된 에러가 발생하는 것을 방지하려면, 사용자는 데이터 파일들을 추가하거나 기존 데이터 파일의 크기를 확장해야 한다.

다음은 언두 테이블스페이스에 데이터 파일을 추가하는 예제이다.

```
ALTER TABLESPACE SYS_TBS_DISK_UNDO
ADD DATAFILE 'undo002.dbf' AUTOEXTEND ON NEXT 1M MAXSIZE 2G;
```


ALTER TABLESPACE ... DROP DATAFILE 구문으로 데이터 파일을 제거할 수도 있으며, ALTER TABLESPACE ... ALTER DATAFILE... 구문으로 파일의 크기를 확장하거나 축소할 수 있다.

그리고 ALTER TABLESPACE ... BEGIN BACKUP 구문으로 데이터 파일의 백업을 시작할 수 있으며, ALTER TABLESPACE ... END BACKUP 구문을 사용해서 백업 완료를 할 수 있다.

테이블스페이스 상태

테이블스페이스는 서비스 상태에 따라 온라인 (online), 오프라인 (offline), 또는 폐기 (discard) 상태로 있게 된다.

테이블스페이스 중에서 사용자가 생성한 디스크 테이블스페이스와 메모리 테이블스페이스의 상태는 온라인과 오프라인으로 변경할 수 있으나, 휘발성 테이블스페이스와 임시 테이블스페이스의 상태는 변경할 수 없다. 또한 테이블스페이스 내에 이중화가 걸려있는 테이블이 존재할 경우에도 변경 불가능하다.

상태를 변경하기 위해서는 ALTER TABLESPACE ... ONLINE 과 ALTER TABLESPACE ... OFFLINE 구문을 사용한다. 단, 테이블스페이스의 온라인/오프라인 상태 전이는 메타 (META) 단계와 서비스 (SERVICE) 구동 단계에서만 가능하다.

온라인 (Online)

테이블스페이스와 관련된 모든 자원이 할당되고 준비된 상태이며, 데이터베이스에서 테이블스페이스를 사용할 수 있게 설정된 상태이다. 테이블스페이스와 그 안에 존재하는 테이블과 인덱스에 대해 DML과 DDL을 수행할 수 있다. 만약 온라인 상태인 테이블스페이스와 테이블스페이스에 생성된 테이블 및 인덱스를 일시적으로 사용할 수 없게 하려면 ALTER TABLESPACE ... OFFLINE 구문을 실행하면 된다.

오프라인 (Offline)

테이블스페이스와 관련된 모든 자원이 해제된 상태이며, 데이터베이스에서 테이블스페이스를 일시적으로 사용할 수 없게 설정된 상태이다.

테이블스페이스에 존재하는 테이블과 인덱스에 대한 DML과 DDL을 수행할 수 없다. 그러나 테이블스페이스에 대한 DROP TABLESPACE와 ALTER TABLESPACE ONLINE DDL구문은 사용할 수 있다.

테이블스페이스와 그 안에 생성된 테이블과 인덱스를 다시 사용할 수 있는 온라인 상태로 전이하기 위해서는 ALTER TABLESPACE ... ONLINE 구문을 사용하면 된다.

메모리 테이블스페이스가 오프라인으로 되면 메모리 테이블스페이스의 객체는 메모리에 적재되지 않기 때문에, 메모리 한계 (메모리 부족) 상황이 발생했을 때 사용자는 메모리 테이블스페이스를 오프라인으로 변경해서 그 상황을 해소할 수 있다.

폐기 (Discard)

데이터의 일관성이 깨진 특정 테이블스페이스 때문에 정상적인 Altibase 구동이 불가능한 경우, 깨진 테이블스페이스를 제외한 나머지에 대해서만이라도 정상적으로 데이터베이스 운영을 할 수 있어야 한다. 이를 위해서 해당 테이블스페이스는 폐기시켜야 한다.

특정 테이블스페이스를 폐기 상태로 전이시키기 위해서는 CONTROL 구동 단계에서 ALTER TABLESPACE ... DISCARD 구문을 실행해야 한다.

그러나 한 번 폐기된 테이블스페이스는 제거 (DROP TABLESPACE)만 할 수 있기 때문에 ALTER TABLESPACE ... DISCARD 구문을 수행할 때에는 주의해야 한다.

테이블스페이스 관리

본 절에서는 Altibase에서 지원하는 테이블스페이스를 관리하는 방법에 대해 설명한다

생성 (CREATE)

테이블스페이스 생성은 SYS 사용자 또는 테이블스페이스 생성 권한을 가진 사용자만 할 수 있다. 테이블스페이스를 생성하려면 CREATE TABLESPACE ... SQL 구문을 사용하라. 사용자는 사용자 정의 데이터 테이블스페이스 (User-defined Data Tablespace)만 생성할 수 있다. 즉 시스템 테이블스페이스들은 사용자가 임의로 생성할 수 없다.

디스크 테이블스페이스는 디스크 데이터 테이블스페이스와 디스크 임시 테이블스페이스로 분류된다.

메모리 테이블스페이스는 메모리 데이터 테이블스페이스만 있고, 메모리 임시 테이블스페이스는 없다.

마찬가지로 휘발성 테이블스페이스는 휘발성 데이터 테이블스페이스만 있고, 휘발성 임시 테이블스페이스는 없다.

다음은 테이블스페이스를 생성하는데 사용되는 SQL 구문이다.

```
CREATE [DISK/MEMORY/VOLATILE] [DATA/TEMPORARY] TABLESPACE
```

(1) 테이블스페이스 이름

- (2) 디스크 데이터 파일 속성
- (3) 디스크 임시 파일 속성
- (4) 메모리 테이블스페이스 속성
- (5) 휘발성 테이블스페이스 속성;

테이블스페이스에 저장된 객체의 크기 및 접근 빈도수와 같은 특성을 고려해서 메모리, 디스크, 또는 휘발성 테이블스페이스의 생성 여부를 결정해야 한다.

테이블스페이스 생성시 지정할 수 있는 테이블스페이스 속성들은 디스크, 메모리, 또는 휘발성에 따라 다르다. 메모리 테이블스페이스는 여러 개의 데이터 파일로 관리되는 디스크 테이블스페이스와는 달리 객체들이 하나의 선형적인 메모리 공간에 저장된다. 따라서, 디스크 테이블스페이스 생성의 경우 각 데이터 파일에 속성이 적용되지만, 메모리 테이블스페이스의 경우 그 메모리 테이블스페이스 전체에 적용된다. 즉 메모리 테이블스페이스에 초기 크기, 확장될 크기 등이 적용되며, 디스크 테이블스페이스는 데이터 파일에 해당 속성들이 적용된다.

테이블스페이스 이름

테이블스페이스 이름은 유일해야 한다. 동일한 이름의 객체가 두개 이상 생성될 수 없다. 디스크 테이블스페이스의 경우에는 데이터 파일들의 이름을 지정할 수 있지만, 메모리 테이블스페이스의 경우에는 체크포인트 이미지 파일이 저장될 경로만을 지정할 수 있다. 체크포인트 이미지 파일의 이름은 테이블스페이스의 이름을 이용하여 자동으로 만들어진다.

디스크 데이터 파일 속성

데이터 파일 속성은 디스크 데이터 테이블스페이스에만 적용되며, 다음과 같은 구문을 갖는다.

```
DATAFILE [①데이터 파일절
AUTOEXTEND [②자동확장절
MAXSIZE [③최대크기절] ] ]
EXTENTSIZE [④익스텐트사이즈절]
```

각 데이터 파일은 다음과 같은 속성을 가질 수 있다.

데이터 파일절

```
DATAFILE {데이터 파일 경로 및 이름} SIZE integer [K/M/G] [REUSE]
```

데이터 파일 경로 및 이름을 지정한다. SIZE절 이하는 생략 가능하다. SIZE절은 데이터 파일이 생성될 때의 초기 데이터 파일의 크기를 명시하는데 사용된다. 각 데이터 파일에는 파일 헤더가 저장된다. SIZE는 파일 헤더 크기 (1 page)를 제외한 나머지 페이지들의 총 크기를 의미한다. 따라서, 지정한 초기 크기와 실제 데이터 파일의 크기가 정확히 일치하지는 않는다. 만약 운영 체제에서 지원하는 최대 파일 크기가 초기 크기보다 작을 경우 에러가 반환될 것이다.

자동확장절

```
AUTOEXTEND [{ON NEXT integer [K/M/G]}/{OFF}]
```

디스크 데이터 파일의 확장 여부를 지정한다. ON일 경우 시스템에 의해서 데이터 파일이 자동으로 확장된다. OFF일 경우 사용자가 명시적으로 파일을 확장해야 한다. 임시 데이터 파일의 확장 단위는 사용자가 NEXT절에 명시할 수 있다.

데이터 파일이 확장될 때 해당 데이터 파일이 속한 테이블스페이스에서 수행되는 모든 연산은 해당 데이터 파일이 확장이 끝날 때까지 대기한다.

최대크기절

```
MAXSIZE {{integer [K/M/G]}/{UNLIMITED}}
```

자동확장절의 부속절로써, 데이터 파일이 확장될수 있는 최대 크기를 의미한다. 초기 크기와 마찬가지로 만약 운영 체제에서 지원하는 최대 파일 크기가 데이터 파일의 최대 크기보다 작을 경우 운영 체제의 최대 파일 크기로 설정된다. UNLIMITED로 설정된 경우에는 데이터 파일이 디스크의 가능한 공간을 모두 사용할 때까지 사이즈가 늘어난다.

익스텐트 사이즈절

```
EXTENTSIZE {{integer [K/M/G]}/{UNLIMITED}}
```

테이블스페이스에 저장되는 세그먼트 (테이블 또는 인덱스)가 할당받는 단위인 익스텐트의 사이즈를 정의한다. 익스텐트 사이즈를 명시하지 않을 경우 기본값으로 512K (64 pages)를 갖는다.

디스크 임시 파일 속성

임시 파일 속성은 디스크 임시 테이블스페이스에만 적용되며, 다음과 같은 구문을 갖는다.

TEMPFILE {㉑임시 파일절}
AUTOEXTED [㉒자동확장절
MAXSIZE [㉓최대크기절]]
EXTENDSIZE [㉔익스텐트사이즈절]

각 임시 파일은 다음과 같은 속성을 가질 수 있다.

임시 파일절

TEMPFILE {데이터 파일 경로 및 이름} SIZE integer [K/M/G] [REUSE]

이 절은 임시 파일 경로 및 이름을 지정하며, SIZE절 이하는 생략 가능하다. SIZE절은 임시 파일이 생성될 때의 초기 크기를 명시하는데 사용된다. 각 임시 파일에는 파일 헤더가 저장된다. SIZE는 파일 헤더 크기 (1 page)를 제외한 나머지 페이지들의 총 크기를 의미한다. 따라서, 지정한 초기 크기와 실제 임시 파일의 크기가 정확히 일치하지는 않는다. 만약 운영 체제에서 지원하는 최대 파일 크기가 초기 크기보다 작을 경우 예러가 반환될 것이다.

자동확장절

AUTOEXTEND [{ON NEXT integer [K/M/G]}/{OFF}]

디스크 임시 파일의 확장 여부를 결정한다. ON일 경우 시스템에 의해서 임시 파일이 자동으로 확장된다. OFF일 경우 사용자가 명시적으로 파일을 확장해야 한다. 임시 파일의 확장 단위는 사용자가 NEXT절에 명시할 수 있다.

최대크기절

MAXSIZE {{integer [K/M/G]}/{UNLIMITED}}

자동확장절의 부속절로써, 임시 파일이 확장될수 있는 최대 크기를 의미한다. 초기 크기와 마찬가지로 만약 운영 체제에서 지원하는 최대 파일 크기가 임시 파일의 최대 크기보다 작을 경우 운영 체제의 최대 파일 크기로 설정된다. UNLIMITED로 설정된 경우에는 임시 파일이 디스크의 가능한 공간을 모두 사용할 때까지 크기가 늘어난다.

익스텐트 사이즈절

EXTENDSIZE integer [K/M/G]

임시 테이블스페이스에 저장되는 세그먼트 (테이블 또는 인덱스)가 할당받는 단위인 익스텐트의 사이즈를 정의한다. 익스텐트 사이즈를 명시하지 않을 경우 기본값으로 256K (32 pages)를 갖는다.

메모리 테이블스페이스 속성

메모리 테이블스페이스에 적용되는 속성은 디스크 테이블스페이스의 데이터 파일 속성과 유사하지만, 추가적으로 체크포인트 이미지 경로를 포함한다. 구문은 다음과 같다.

```
SIZE {①초기 크기절}
AUTOEXTED [②자동확장절
MAXSIZE [③최대크기절] ]
CHECKPOINT PATH [④체크포인트 이미지 경로절]
```

메모리 테이블스페이스는 다음과 같은 속성을 가질 수 있다.

초기 크기절

```
SIZE integer [K/M/G]
```

메모리 테이블스페이스 생성시 초기에 할당되어야 할 메모리 크기를 나타낸다. 이 값은 메모리 테이블스페이스의 기본 확장 단위의 배수여야 한다. (EXPAND_CHUNK_PAGE_COUNT 프로퍼티에 지정한 페이지 개수 * 메모리 테이블스페이스의 페이지 크기(32KB)²)

[²]예를 들어 EXPAND_CHUNK_PAGE_COUNT를 128로 지정하였다면, 메모리

테이블스페이스의 기본 확장 크기는 128 * 32K로 계산되어 4MB가 된다. 그러므로 SIZE로 지정할 수 있는 크기는 4MB의 배수이다.

이 크기는 KiloBytes (K), Megabytes (M), 또는 Gigabytes (G) 단위로 명시할 수 있다. 이 단위를 명시하지 않을 경우 기본 단위는 KiloBytes (K)이다.

자동확장절

```
AUTOEXTEND [{ON NEXT integer [K/M/G]}/{OFF}]
```

메모리 테이블스페이스의 자동 확장 여부를 결정한다. ON일 경우 시스템에 의해서 테이블스페이스가 자동으로 확장되지만, OFF일 경우 사용자가 명시적으로 크기를 확장해야 한다. 확장되는 크기는 사용자가 NEXT절에 명시할 수 있다.

확장되는 크기는 초기 크기와 마찬가지로 EXPAND_CHUNK_PAGE_COUNT 프로퍼티에 설정된 페이지 크기의 배수에 해당하는 크기로 지정하여야 한다.

자동 확장 크기가 너무 작으면 자동확장이 빈번하게 발생할 수 있다. Altibase는 자동확장을 수행할 때 시스템에 존재하는 모든 메모리 테이블스페이스의 현재 크기를

합산하여 MEM_MAX_DB_SIZE 프로퍼티에 지정한 크기보다 작은지 비교한다. 이러한 연산이 빈번하게 수행되면 시스템의 성능이 저하될 수 있다.

최대크기절

```
MAXSIZE {{integer [K/M/G]}/{UNLIMITED}}
```

자동확장절의 부속절로써, 메모리 테이블스페이스가 확장될수 있는 최대 크기를 의미한다. 초기 크기와 마찬가지로 운영 체제에서 제공되는 메모리 공간의 크기를 초과할 수 없다. UNLIMITED로 설정된 경우에는 시스템에 존재하는 모든 메모리 테이블스페이스의 크기를 합친 전체 크기가 MEM_MAX_DB_SIZE 프로퍼티에 지정한 크기를 벗어나지 않는 한도 내에서 테이블스페이스가 자동확장된다.

체크포인트 이미지 경로절

```
CHECKPOINT PATH ‘체크포인트 이미지 경로 리스트’  
SPLIT EACH integer [K/M/G]
```

체크포인트 이미지 경로는 메모리 테이블스페이스에만 적용된다. Altibase는 메모리 테이블스페이스의 고성능 트랜잭션 처리를 위하여 핑퐁(ping-pong) 체크포인트를 사용한다. 핑퐁 체크포인트를 위해서 두 별의 체크포인트 이미지가 디스크에 생성된다. 각 체크포인트 이미지는 다수의 파일에 분할되어 저장될 수 있다. 체크포인트 이미지가 분할되는 크기는 SPLIT EACH절에 명시할 수 있다. 분할된 파일은 디스크 입출력 비용을 분산하기 위하여 서로 다른 경로에 저장될 수 있으며, 사용자가 임의로 분할의 크기 및 체크포인트 이미지 파일들이 저장되는 경로들을 지정할 수 있다. 사용자는 체크포인트 이미지 경로를 추가하거나 변경할 수 있지만, 한번 지정된 분할의 크기를 변경할 수는 없다.

휘발성 테이블스페이스 속성

휘발성 테이블스페이스에 적용되는 속성은 체크포인트 이미지 경로를 제외하고는 메모리 테이블스페이스의 속성과 유사하다.

```
SIZE {①초기 크기절}  
AUTOEXTED [②자동확장절  
MAXSIZE [③최대크기절] ]
```

휘발성 테이블스페이스는 다음과 같은 속성을 가질 수 있다.

초기크기절

```
SIZE integer [K/M/G]
```

휘발성 테이블스페이스 생성시 초기에 할당되어야할 메모리 크기를 나타낸다. 이 값은 메모리 테이블스페이스의 기본 확장 단위의 배수여야 한다. (EXPAND_CHUNK_PAGE_COUNT 프로퍼티에 지정한 페이지 개수 * 메모리 테이블스페이스의 페이지 크기(32KB)³)

[³] 예를 들어 EXPAND_CHUNK_PAGE_COUNT를 128로 지정하였다면, 메모리

테이블스페이스의 기본 확장 크기는 128 * 32K로 계산되며, 값은 4MB가 된다. 그러므로 SIZE로 지정할 수 있는 크기는 4MB의 배수이다.

이 크기는 KiloBytes (K), Megabytes (M), 또는 Gigabytes (G) 단위로 명시할 수 있다. 이 단위를 명시하지 않을 경우 기본 단위는 KiloBytes (K)이다.

자동확장절

```
AUTOEXTEND [{ON NEXT integer [K/M/G]}/{OFF}]
```

휘발성 테이블스페이스의 자동 확장 여부를 결정한다. ON일 경우 시스템에 의해서 테이블스페이스가 자동으로 확장되지만, OFF일 경우 사용자가 명시적으로 크기를 확장해야 한다. 확장되는 크기는 사용자가 NEXT절에 명시할 수 있다.

확장되는 크기는 초기 크기와 마찬가지로 EXPAND_CHUNK_PAGE_COUNT 프로퍼티에 설정된 페이지 크기의 배수에 해당하는 크기로 지정하여야 한다.

자동 확장 크기가 너무 작으면 자동확장이 너무 빈번하게 발생할 수 있다. Altibase는 자동확장을 수행할 때 시스템에 존재하는 모든 휘발성 테이블스페이스의 현재 크기를 합산하여 VOLATILE_MAX_DB_SIZE 프로퍼티에 지정한 크기보다 작은지 비교한다. 이러한 연산을 빈번하게 수행하면 시스템의 성능이 저하될 수 있다.

최대크기절

```
MAXSIZE [{integer [K/M/G]}/{UNLIMITED}]
```

자동확장절의 부속절로써, 휘발성 테이블스페이스가 확장될수 있는 최대 크기를 의미한다. 초기 크기와 마찬가지로 운영 체제에서 제공되는 메모리 공간의 크기를 초과할 수 없다. UNLIMITED로 설정된 경우에는 시스템에 존재하는 모든 휘발성 테이블스페이스의 크기를 합친 전체 크기가 VOLATILE_MAX_DB_SIZE 프로퍼티에 지정한 크기를 벗어나지 않는 한도 내에서 테이블스페이스가 자동확장 된다.

예제

Ex.1) 세개의 데이터 파일을 가지는 디스크 데이터 테이블스페이스를 생성한다.


```
iSQL> CREATE DISK DATA TABLESPACE user_data DATAFILE
'/tmp/tbs1.user' SIZE 10M AUTOEXTEND ON NEXT 1M MAXSIZE 1G,
'/tmp/tbs2.user' SIZE 10M AUTOEXTEND ON NEXT 1M MAXSIZE 500M,
'/tmp/tbs3.user' SIZE 10M AUTOEXTEND ON NEXT 1M MAXSIZE 1G;
Create success.
```

Ex.2) 메모리 데이터 테이블스페이스를 생성한다.

```
iSQL> CREATE MEMORY DATA TABLESPACE user_data SIZE 12M
AUTOEXTEND ON NEXT 4M MAXSIZE 500M
CHECKPOINT PATH '/tmp/checkpoint_image_path1', '/tmp/checkpoint_image_path2' SPLIT EACH
12M;
Create success.
```

Ex.3) 휘발성 데이터 테이블스페이스를 생성한다.

```
iSQL> CREATE VOLATILE DATA TABLESPACE user_data SIZE 12M
AUTOEXTEND ON NEXT 4M MAXSIZE 500M;
Create success.
```

삭제 (DROP)

테이블스페이스 삭제는 SYS 사용자 또는 테이블스페이스 삭제 권한을 가진 사용자만이 할 수 있다. 테이블스페이스를 삭제하려면 DROP TABLESPACE ... SQL 구문을 사용하라. 시스템 테이블스페이스들은 사용자에게 의해서 삭제될 수 없다. 테이블스페이스의 삭제는 메모리나 디스크, 휘발성을 명시적으로 구분하지 않으며, 아래와 같은 구문을 갖는다.

```
DROP TABLESPACE {테이블스페이스 이름}
[{(1)객체 삭제절} [(2)데이터 파일 삭제절}
[(3)제약사항 삭제절]];
```

테이블스페이스의 이름을 지정하여 삭제를 수행하며 선택할 수 있는 옵션은 아래와 같다. 만약 아래 옵션들을 선택하지 않는다면 테이블스페이스의 스키마만이 로그앵커(loganchor)에서 삭제된다.

객체 삭제절

```
INCLUDING CONTENTS
```

테이블스페이스내 객체 (테이블 또는 인덱스)들과 객체의 내용을 삭제한다. 만약 테이블스페이스 내에 하나 이상의 객체가 존재한다면 반드시 해당 옵션을 선택해야 한다. 그렇지 않을 경우, 테이블스페이스 삭제 연산은 실패할 것이다.

데이터 파일 삭제절

INCLUDING CONTENTS AND DATAFILES

객체 삭제절을 명시하였을 경우, 객체의 레코드 및 인덱스 키가 삭제되지만 데이터 파일 자체가 삭제되는 것은 아니다. 따라서, 데이터 파일을 지우기 위해서는 데이터 파일 삭제절을 명시해야 한다.

데이터 파일 삭제절은 객체 삭제절의 부속절로써, 테이블스페이스가 가지고 있는 모든 데이터 파일을 물리적으로 삭제한다. 메모리 테이블스페이스의 경우에는 해당 메모리 테이블스페이스의 모든 체크포인트 이미지 파일이 물리적으로 삭제된다. 그러나 휘발성 테이블스페이스의 경우에 AND DATAFILES 구문을 사용하면 에러가 발생한다.

제약사항 삭제절

INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS

객체 삭제절의 부속절로써, 삭제하고자 하는 테이블스페이스 내의 객체들을 참조하는 제약사항 (constraints)이 다른 테이블스페이스에 존재하는 경우 테이블스페이스에 객체가 남아있다는 에러가 발생하며 실패하게 된다. 이럴 경우에 객체와 참조를 삭제하기 위해 CASCADE CONSTRAINTS 절이 사용될 수 있다.

변경 (ALTER)

테이블스페이스 변경은 SYS 사용자 또는 테이블스페이스 삭제 권한을 가진 사용자만이 할 수 있다. 테이블스페이스를 변경하려면 ALTER TABLESPACE ... SQL 구문을 사용하라. 이 구문은 기존의 테이블스페이스의 정의, 하나 이상의 데이터 파일 또는 임시 파일의 속성, 또는 메모리 테이블스페이스, 휘발성 테이블스페이스의 속성을 변경하는데 사용된다. 다음은 SQL 구문을 설명한다.

```
ALTER TABLESPACE {테이블스페이스 이름}
{ {(1)디스크 데이터 파일 변경절}/
  {(2)디스크 임시 파일 변경절}/
  {(3)메모리 테이블스페이스 변경절}/
  {(4)휘발성 테이블스페이스 변경절}/
  {(5)테이블스페이스 상태 변경절}};
```

디스크 데이터 파일 변경절

이 절은 디스크 시스템 테이블스페이스와 디스크 데이터 테이블스페이스에 사용할 수 있으며, 아래와 같은 옵션을 가진다.

```
ALTER TABLESPACE {테이블스페이스 이름}
{①데이터파일 추가절/
②데이터 파일 삭제절/
③데이터 파일 크기 변경절/
④데이터 파일 이름 변경절}
```

데이터파일 추가절

```
ADD {DATAFILE} {데이터 파일절}
      AUTOEXTEND [자동확장절
      MAXSIZE [최대크기절] ]
```

이 절은 디스크 테이블스페이스에서 데이터를 저장할 공간을 확장하려 할 때 사용된다. 가능한 옵션은 테이블스페이스 생성시에 데이터 파일에 적용되는 구문과 동일하다.

데이터 파일 삭제절

```
DROP {DATAFILE} {데이터 파일명}
```

이 절은 디스크 테이블스페이스에서 데이터를 저장할 공간을 축소하려 할때 사용된다. 데이터 파일 추가에 의한 확장은 자유롭게 수행할 수 있지만, 데이터 파일의 삭제는 해당 데이터 파일이 현재 사용중이지 않을 때, 즉 해당 데이터 파일까지 익스텐트가 확장되지 않았을 때만 가능하다.

데이터 파일 크기 변경절

```
ALTER {DATAFILE} {데이터 파일명}
      {{AUTOEXTEND [자동 확장절]} /
      {SIZE [크기 변경절]}}
```

이 절은 디스크 테이블스페이스에 속하는 각 데이터 파일들의 현재 크기, 최대 크기, 확장 단위 및 자동확장 여부 등을 변경하는 데 사용된다.

명시되는 현재 크기 및 최대 크기는 현재 사용중인 크기보다 커야 한다.

데이터 파일 이름 변경절

```
RENAME {DATAFILE} {기존 데이터 파일 경로및 이름}
      TO {새로운 데이터 파일 경로및 이름}
```

데이터 파일의 위치를 변경함으로써 테이블스페이스의 데이터가 저장된 파일 시스템을 변경하는 것이다. 이 절은 온라인이나 오프라인에 상관없이 어떤 구동 단계에서도

수행 가능하다. 하지만, 서비스 단계에서는 오프라인 상태인 테이블스페이스에 대해서만 수행할 수 있다.

디스크 임시 파일 변경절

이 절은 디스크 임시 테이블스페이스에만 사용할 수 있다. 아래와 같은 옵션들을 가진다.

```
ALTER TABLESPACE {테이블스페이스 이름}
    {①임시 파일 추가절/
     ② 임시 파일 삭제절/
     ③ 임시 파일 크기 변경절/
     ④ 임시 파일 이름 변경절}
```

임시 파일 추가절

```
ADD {TEMPFILE} {임시 파일절}
AUTOEXTEND [자동확장절
MAXSIZE [최대크기절]]
```

이 절은 디스크 임시 테이블스페이스에서 데이터를 저장할 공간을 확장하려 할 때 사용된다. 가능한 옵션은 테이블스페이스 생성시에 데이터 파일에 적용되는 구문과 동일하다.

임시 파일 삭제절

```
DROP {TEMPFILE} {임시 파일명}
```

디스크 임시 테이블스페이스에서 데이터를 저장할 공간을 축소하려 할때 사용한다. 데이터 파일 추가에 의한 확장은 자유롭게 수행할 수 있지만, 데이터 파일의 삭제는 해당 데이터 파일이 현재 사용중이지 않을 때, 즉 해당 데이터 파일까지 익스텐트가 확장되지 않았을 때만 가능하다.

임시 파일 크기 변경절

```
ALTER {TEMPFILE} {임시 파일명}
    {{AUTOEXTEND [자동 확장절]} /
     {SIZE [크기 변경절]}}
```

이 절은 디스크 임시 테이블스페이스에 속하는 각 임시 파일들의 현재 크기, 최대 크기, 확장 단위 및 자동확장 여부 등을 변경하는 데 사용된다.

명시되는 현재 크기 및 최대 크기는 현재 사용중인 크기보다 커야 한다.

임시 파일 이름 변경절

```
RENAME {TEMPFILE} {기존 임시 파일 경로및 이름}  
      TO {새로운 임시 파일 경로및 이름}
```

임시 파일의 위치를 변경함으로써 테이블스페이스의 데이터가 저장된 파일 시스템을 변경하는 것이다. 해당 기능은 온라인이나 오프라인에 상관없이 어떤 구동 단계에서도 수행 가능하다. 하지만, 서비스 단계에서는 오프라인 상태인 테이블스페이스에 대해서만 수행할 수 있다.

메모리 테이블스페이스 변경절

이는 메모리 시스템 테이블스페이스와 메모리 사용자 정의 테이블스페이스에만 사용될 수 있으며, 아래와 같은 옵션들을 가진다. 체크포인트 경로에 대한 추가, 삭제 및 변경은 어느 구동 단계에서도 수행 가능하다. 그러나 서비스 단계에서는 오프라인 테이블스페이스만 변경이 가능하다.

```
ALTER TABLESPACE {테이블스페이스 이름}  
  {① 체크포인트 경로 추가절/  
   ② 체크포인트 경로 삭제절/  
   ③ 체크포인트 경로 변경절/  
   ④ 테이블스페이스 크기 변경절}
```

체크포인트 경로 추가절

```
ADD CHECKPOINT PATH {디렉토리 경로}
```

체크포인트 이미지 경로를 추가적으로 설정한다.

체크포인트 경로 삭제절

```
DROP CHECKPOINT PATH {디렉토리 경로}
```

기존 체크포인트 이미지 경로를 삭제한다.

체크포인트 경로 변경절

```
RENAME CHECKPOINT PATH {기존 디렉토리 경로}  
      TO {새로운 디렉토리 경로}
```

기존 체크포인트 이미지 경로를 새로운 경로로 변경한다.

테이블스페이스 크기 변경절

ALTER

```
{{AUTOEXTEND [자동 확장절]} /  
{SIZE [크기 변경절]}}
```

메모리 테이블스페이스의 최대 크기, 확장 단위, 자동확장 여부 등을 변경한다.

휘발성 테이블스페이스 변경절

이 절은 휘발성 사용자 정의 테이블스페이스만 사용될 수 있으며, 아래와 같은 옵션을 가진다.

```
ALTER TABLESPACE {테이블스페이스 이름}  
    {① 테이블스페이스 크기 변경절}
```

테이블스페이스 크기 변경절

ALTER

```
{{AUTOEXTEND [자동 확장절]} /  
{SIZE [크기 변경절]}}
```

휘발성 테이블스페이스의 최대 크기, 확장 단위, 자동확장 여부 등을 변경한다.

테이블스페이스 상태 변경절

테이블스페이스의 상태는 온라인과 오프라인이 있으며, 다음과 같은 구문으로 상태를 설정할 수 있다.

```
ALTER TABLESPACE {테이블스페이스 이름}  
    {ONLINE/OFFLINE/DISCARD}
```

온라인 상태는 테이블스페이스에 속한 모든 객체에 사용자가 접근할 수 있는 일반적인 상태이다. 반면 오프라인 상태는 테이블스페이스 관련 DDL을 제외한 다른 연산에 의해 테이블스페이스 객체에 접근할 수 없는 상태이다. 이러한 오프라인 상태를 이용하여 한계 상황 극복이나 서비스 단계에서의 RENAME 연산 등을 할 수 있다. 단 시스템 테이블스페이스는 항상 온라인 상태로 유지되며, 오프라인으로 변경될 수 없다. 휘발성 테이블스페이스에 대해서는 이 구문을 사용할 수 없다.

디스카드 옵션은 Altibase에서 운용중인 여러 테이블스페이스 중 특정

테이블스페이스의 데이터에 오류가 발생⁴하여 Altibase가 기동되지 않는 문제가 발생할 경우에 사용될 수 있다. 해당 테이블스페이스를 디스카드 (DISCARD)함으로써 해당 테이블스페이스를 버리는 대신, 나머지 테이블스페이스로 Altibase를 기동할 수 있다. 한번 디스카드된 테이블스페이스는 제거 (DROP)외에 다른 연산은 사용이 불가하게 되므로, 사용에 신중을 기하여야 한다. 아울러, 테이블스페이스는 오직

컨트롤 (CONTROL) 단계에서만 디스카드 될 수 있다. 이 옵션은 디스크 테이블스페이스와 메모리 데이터 테이블스페이스에 사용될 수 있다.

[4] 예를 들어 DBA가 실수로 특정 메모리 테이블스페이스의 체크포인트 이미지

파일을 삭제했다고 가정하자. 이 경우 서버 기동시 해당 메모리 테이블스페이스를 로드할 수 없기 때문에 DBA는 매체 복구를 이용하여 삭제된 체크포인트 이미지를 재생성하는 방법을 먼저 생각해 볼 수 있다. 그러나, 아카이브 로깅을 사용하지 않고 있다면, 매체 복구가 불가능하기 때문에 이 방법은 사용할 수 없을 것이다. 이 때 만약 해당 테이블스페이스를 삭제해도 상관이 없다면, 디스카드 기능을 이용하여 해당 테이블스페이스를 제외하고 DB를 구동한 후 테이블스페이스를 제거하면 된다.

테이블스페이스 백업 및 복구

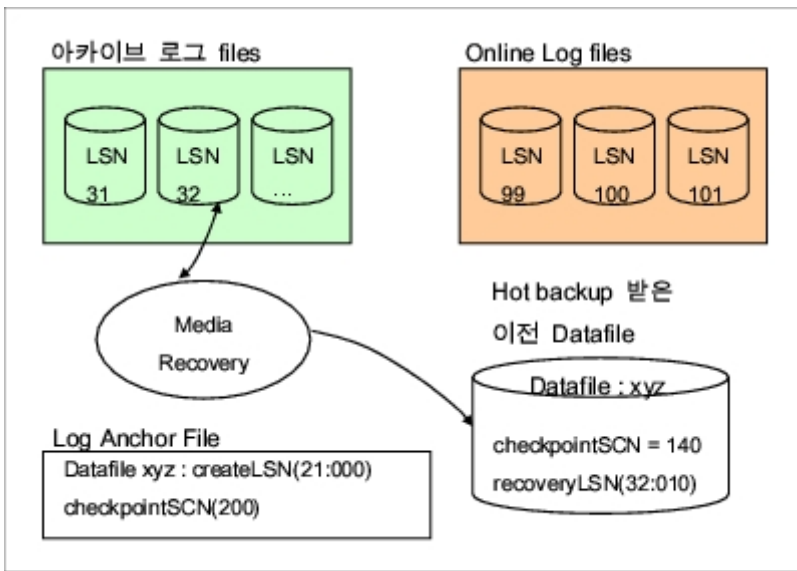
이번 절에서는 테이블스페이스의 온라인/오프라인 백업의 개념 및 특징을 간략히 설명한다. 백업 및 복구에 대한 자세한 설명은 이 매뉴얼의 해당 장과 *Getting Started Guide*을 참고한다.

테이블스페이스 온라인 백업 (HOT 백업)

테이블스페이스의 온라인 백업은 서비스 제공중인 테이블스페이스에 대한 백업을 하는 것이다. 온라인 백업은 트랜잭션 진행에는 영향을 주지 않기 때문에, 서비스 단계에서 이루어질 수 있다. 온라인 백업은 다음의 몇 가지 특징을 갖는다.

- 온라인 백업은 데이터베이스가 아카이브 로그 모드로 운영될 때만 가능하다.
- 아카이브 로그 모드에서는, 체크포인트와 로그 플러시 (Log Flush) 후에도 로그 파일을 별도의 스토리지에 백업하므로 대용량의 스토리지를 필수로 준비해야 한다.
- ALTER DATABASE BACKUP 구문을 이용해서 데이터베이스 운영 중에 온라인 백업을 할 수 있다.
- 장애로 인하여 데이터 파일이 손상되거나 지워지는 경우에도 데이터 파일을 현재 시점까지 매체 복구 (media recovery)가 가능하다.

[그림 6-14] 매체 복구 (Media Recovery)의 개념



- 디스크 테이블스페이스의 데이터 파일 xyz가 손상되었을 경우에 이전에 HOT 백업해 두었던 데이터 파일을 이용해서 복구가 가능하다. 메모리 테이블스페이스의 경우는 이전에 HOT 백업해 두었던 체크포인트 이미지 파일을 이용해서 복구가 가능하다.
- 백업해 둔 데이터 파일의 헤더에는 백업 시점에 최종 체크포인트된 SCN(140)과 recovery LSN(32:010)이 있으므로, 이를 기준으로 현재의 최종 체크포인트 SCN (200)까지 데이터 파일을 복원할 수 있다.
- 재시작 시점에 온라인 로그의 리두(Redo)와 언두(Undo) 로그를 이용해서 데이터 파일 또는 메모리 테이블스페이스의 가장 최근 상태 이미지로 복구된다.

테이블스페이스 오프라인 백업 (Cold 백업)

테이블스페이스의 오프라인 백업은 테이블스페이스의 서비스 진행을 중단하고, 백업하는 형태를 의미한다. 오프라인 백업 방식은 온라인 백업보다 빠르며, 회복에 걸리는 시간을 단축시킨다. 오프라인 백업은 다음과 같은 특징을 갖는다.

- 오프라인 백업은 데이터베이스가 노-아카이브 모드로 운영될 때 가능하다.
- 오프라인 백업이란 데이터베이스 서버를 정상 종료 시킨 후에 데이터 파일, 로그 파일, 로그 앵커(log anchor) 파일 등을 복사하는 방식이다.
- 장애로 인하여 데이터 파일이 손상되거나 지워지는 경우에는 최종 오프라인 백업된 시점까지만 복구가 가능하다.

오프라인 복구

복구는 백업 이미지를 바탕으로 데이터베이스를 일관적인 상태로 만드는 과정이다. 복구는 데이터베이스가 온라인 중에는 진행될 수 없으며, 반드시 오프라인으로 진행되어야 한다.

데이터베이스의 서비스를 중지한 상태에서 기존 데이터베이스를 오프라인 백업된 파일로 바꾼 후에 재시작함으로써 복구가 수행된다.

테이블스페이스 사용 예제

본 절에서는 메모리 테이블스페이스와 휘발성 테이블스페이스 사용 예제를 살펴본다.

메모리 테이블스페이스

메모리 테이블스페이스 생성 - 기본

메모리테이블스페이스를 생성하는 가장 간편한 방법은 아래와 같이 SIZE절에 초기 크기를 명시하면서 CREATE MEMORY TABLESPACE 구문을 이용하는 것이다.

```
isql> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 256M;  
Create success.
```

이 경우 기본적으로 자동확장 모드는 OFF가 되며, 테이블스페이스의 크기로 256M의 공간을 한번에 할당받게 된다. 테이블스페이스의 256M의 공간을 모두 사용하게 되어, Altibase 서버 내에서 해당 테이블스페이스에 공간을 추가로 할당⁵하려 할 때, 테이블스페이스 공간이 부족하다는 에러 메시지가 발생한다.

[⁵] 테이블스페이스에 테이블을 생성, 이미 생성되어 있는 테이블에 입력, 또는

테이블의 데이터를 수정하는 경우 테이블스페이스로부터 공간을 추가로 할당받는다.

또한 체크포인트 경로는 기본적으로 MEM_DB_DIR 프로퍼티에 지정한 하나 혹은 그 이상의 체크포인트 경로가 새로 생성된 테이블스페이스를 위한 체크포인트 경로로 사용된다.

altibase.properties 에 아래의 예처럼 두 개의 체크포인트 경로가 명시되어 있다고 가정하자. MEM_DB_DIR프로퍼티의 값이 Altibase 홈 디렉토리 아래의 dbs1과 dbs2, 두 개로 지정되어 있다.

```
# altibase.properties  
MEM_DB_DIR      = ?/dbs1  
MEM_DB_DIR      = ?/dbs2
```

이와 같은 경우에 다음의 쿼리를 사용해서 앞서 생성한 USER_MEM_TBS테이블스페이스를 위한 체크포인트 경로로 MEM_DB_DIR에 지정한 dbs1과 dbs2가 지정된 것을 확인할 수 있다.

```

iSQL> SELECT CHECKPOINT_PATH
FROM V$MEM_TABLESPACE_CHECKPOINT_PATHS
WHERE SPACE_ID =
    (SELECT SPACE_ID
    FROM V$MEM_TABLESPACES
    WHERE SPACE_NAME='USER_MEM_TBS');
CHECKPOINT_PATH
-----
/altibase_home/dbs1
/altibase_home/dbs2
2 rows selected.

```

우선, 체크포인트 경로 안의 파일들을 살펴보자. dbs1 디렉토리 내에 다음과 같이 6개의 파일이 존재하는 것을 볼 수 있다.

```

SYS_TBS_MEM_DATA-0-0
SYS_TBS_MEM_DATA-1-0
SYS_TBS_MEM_DIC-0-0
SYS_TBS_MEM_DIC-1-0
USER_MEM_TBS-0-0
USER_MEM_TBS-1-0

```

이 파일들은 모두 메모리 테이블스페이스의 체크포인트 이미지 파일이다. 파일 이름의 형식은 ‘테이블스페이스이름-{Ping Pong번호}-{파일번호}’이다. ‘Ping Pong 번호’는 0 또는 1로, 이는 두 개의 체크포인트 이미지 중 핑퐁 체크포인트⁶시에 사용된 하나를 가리킨다. 또한, 각 체크포인트 이미지들을 여러 개의 파일로 나누어 기록하는데, 파일 이름의 맨 마지막 ‘파일번호’가 바로 나누어진 체크포인트 이미지 파일의 번호이며, 0부터 시작하여 1씩 증가한다. 체크포인트 이미지 파일 하나의 크기는 CREATE TABLESPACE 구문의 SPLIT EACH절에 지정된다. 위의 CREATE MEMORY TABLESPACE 구문에서는 SPLIT EACH절을 사용하지 않기 때문에, 기본값으로 DEFAULT_MEM_DB_FILE_SIZE프로퍼티에 지정된 1GB 단위로 체크포인트 이미지 파일들이 분할될 것이다. 위 세 개의 테이블스페이스에 사용된 공간이 아직 1GB에 도달하지 않았기 때문에, 파일번호는 0까지만 존재하는 것을 볼 수 있다.

[⁶] Altibase는 메모리 상에 존재하는 테이블스페이스의 데이터의

영속성(Durability) 보장을 위해 디스크의 파일에 데이터를 저장한다. 이와 같이 테이블스페이스의 데이터가 저장되는 파일을 체크포인트 이미지라고 한다. Altibase가 사용하는 핑퐁 체크포인트 방식은 두 벌의 체크포인트 이미지를 두고, 하나씩 번갈아가면서 테이블스페이스의 데이터가 저장된다.

위에서 SYS_TBS_MEM_DIC은 메타 데이터를 지니는 시스템 딕셔너리 테이블스페이스이다. 이 테이블스페이스는 데이터베이스 생성시 자동으로 만들어진다.

SYS_TBS_MEM_DATA는 기본 시스템 데이터 테이블스페이스이다. 사용자가 테이블스페이스를 명시하지 않고 테이블을 생성할 때, 테이블의 데이터는 이 테이블스페이스에 저장된다.

마지막으로, USER_MEM_TBS가 바로 앞서 생성한 사용자 정의 데이터 테이블스페이스이다.

참고로, CREATE MEMORY TABLESPACE 구문에서 SIZE절에 지정하는 초기 크기는 확장 증가 크기의 배수여야 한다. 예를 들어 메모리 테이블스페이스가 확장될 때 증가할 페이지의 개수를 지정하는 EXPAND_CHUNK_PAGE_COUNT프로퍼티의 값이 128이면, 하나의 메모리 페이지는 32KB이므로, 메모리 테이블스페이스의 기본 확장 증가 크기는 4MB(128 * 32KB)가 된다.

만약 SIZE절에 지정한 크기가 확장 증가 크기로 나누어 떨어지지 않을 경우, 다음과 같이 에러가 발생한다.

```
iSQL> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 1M;  
[ERR-110EE : The initial size of the tablespace should be a multiple of expand chunk size ( EXPAND_CHUNK_PAGE_CO
```

메모리 테이블스페이스 생성 - 종합

메모리 테이블스페이스의 다양한 생성 방법에 대해 알아본다.

다음은 테이블스페이스의 초기 크기를 256M, 자동확장 모드를 ON으로 하고 한번 확장시마다 128M씩 확장하되, 최대 1G이상 확장되지 않도록 하는 예제이다.

```
iSQL> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 256M  
AUTOEXTEND ON NEXT 128M MAXSIZE 1G;  
Create success.
```

이 때, 테이블스페이스의 자동확장 증가 크기는 테이블스페이스의 초기 크기와 마찬가지로 (EXPAND_CHUNK_PAGE_COUNT 프로퍼티 * 한 페이지의 크기)의 배수로 설정되어야 한다. 자세한 내용은 위의 ‘메모리 테이블스페이스의 생성 - 기본’을 참고한다.

다음과 같이 MAXSIZE를 제한하지 않도록 테이블스페이스를 생성할 수 있다. MAXSIZE절을 지정하지 않는 경우에 UNLIMITED를 지정한 것과 같이 동작한다.

```
iSQL> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 256M  
AUTOEXTEND ON NEXT 128M MAXSIZE UNLIMITED;  
Create success.
```

이 경우 USER_MEM_TBS는 시스템에 존재하는 모든 메모리 테이블스페이스에 할당된 크기가 MEM_MAX_DB_SIZE를 벗어나지 않는 한도 내에서 확장된다.

다음과 같이 체크포인트 경로를 명시하여 메모리 테이블스페이스를 생성할 수도 있다.

```
iSQL> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 256M  
CHECKPOINT PATH 'dbs1', '/new_disk/dbs2';  
Create success.
```

위의 예제처럼, 상대경로 dbs1을 체크포인트 경로로 지정하는 것은 \$ALTIBASE_HOME/dbs1을 지정하는 것과 동일하다. 또한, CREATE TABLESPACE 구문에 지정한 체크포인트 경로를 실제로 파일 시스템에 생성하고, 거기에 쓰기와 실행 권한을 주는 작업은 테이블스페이스를 생성하기 전에 DBA가 수동으로 해야 한다.

다음과 같이 체크포인트 이미지 파일의 분할 크기를 결정할 수도 있다.

```
iSQL> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 256M  
SPLIT EACH 512M;  
Create success.
```

체크포인트 이미지 파일의 분할 크기 역시 테이블스페이스의 초기크기와 마찬가지로 (EXPAND_CHUNK_PAGE_COUNT 프로퍼티 * 한 페이지의 크기)의 배수에 해당하는 크기로 지정되어야 한다. 자세한 내용은 ‘메모리 테이블스페이스의 생성 -기본’편을 참고한다.

테이블스페이스를 오프라인(OFFLINE) 상태로 생성해두고, 나중에 해당 테이블스페이스를 사용하기 전에 온라인(ONLINE) 상태로 전이할 수도 있다. 메모리 테이블스페이스의 경우 생성된 크기만큼 시스템의 메모리를 차지하게 되므로, 생성 후 즉시 사용하지 않을 경우, 이와 같은 방법으로 시스템의 리소스를 최적으로 활용할 수 있다.

```
iSQL> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 256M  
OFFLINE;  
Create success.  
iSQL> ALTER TABLESPACE USER_MEM_TBS ONLINE;  
Alter success.
```

지금까지 살펴본 메모리 테이블스페이스 생성 옵션을 조합하여 테이블스페이스를 생성할 수도 있다.

```
iSQL> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 256M
AUTOEXTEND ON NEXT 128M MAXSIZE 1G
CHECKPOINT PATH 'dbs1', '/new_disk/dbs2'
SPLIT EACH 512M OFFLINE;
Create success.
```

메모리 테이블스페이스에 체크포인트 경로 추가

이 절에서는 메모리 테이블스페이스에 체크포인트 경로를 추가하는 절차에 대해 알아본다. 메모리 테이블스페이스를 위한 체크포인트 경로는 컨트롤(CONTROL) 구동 단계에서만 설정이 가능하다. 우선 Altibase 서버를 종료 시킨 후에 다음과 같이 컨트롤 단계까지 Altibase를 기동한다.

```
$ isql -u sys -p manager -sysdba
iSQL(sysdba)> startup process
iSQL(sysdba)> startup control
```

컨트롤 단계에서는 테이블스페이스 관련 성능 뷰인 V\$TABLESPACES를 조회할 수 있다. 메모리 테이블스페이스 특유의 속성들을 볼 수 있는 성능 뷰인 V\$MEM_TABLESPACES는 메타(META) 단계 이후에만 조회가 가능하다. 대신, 컨트롤 단계에서는 V\$TABLESPACES를 이용하여 메모리 테이블스페이스 조회가 가능하다.

앞서 생성한 USER_MEM_TBS 테이블스페이스를 위한 체크포인트 경로를 조회하기 위해서는 V\$MEM_TABLESPACE_CHECKPOINT_PATHS 성능 뷰를 이용하면 된다.

만약 테이블스페이스의 데이터가 빈번하게 변경되어 체크포인트시 디스크 I/O의 양이 증가했다면, 이는 다음과 같이 기존의 체크포인트 경로와 물리적으로 다른 디스크에 새로운 체크포인트 경로를 추가해서 경감시킬 수 있다.

USER_MEM_TBS에 /new_disk/dbs3 경로를 새로 추가해보자. 우선, 추가하고자 하는 체크포인트 경로와 디렉토리를 생성하고, Altibase 프로세스가 해당 디렉토리에 쓰기와 실행권한을 가지도록 권한을 설정해야 한다. Altibase 프로세스를 수행하는 운영체제의 사용자 계정이 altibase라고 가정한다.

```
$ su - root
$ mkdir /new_disk/dbs3
$ chown altibase /new_disk/dbs3
```

이제 다음과 같이 ADD CHECKPOINT PATH 구문을 이용하여 체크포인트 경로를 추가할 수 있다.

```
iSQL(sysdba)> ALTER TABLESPACE USER_MEM_TBS  
ADD CHECKPOINT PATH '/new_disk/dbs3';  
Alter success.
```

기존의 체크포인트 경로에 존재하는 체크포인트 이미지 파일들을 새로 추가된 체크포인트 경로로 이동시키는 것은 DBA의 책임이다. Altibase는 체크포인트 경로가 추가된 이후에 체크포인트가 발생할 경우, 새로 체크포인트 이미지 파일을 생성할 때, 추가된 체크포인트 경로에 체크포인트 이미지 파일을 생성한다.⁷

[7] 체크포인트시 테이블스페이스의 체크포인트 이미지 파일을 새로 생성할 경우

테이블스페이스에 속한 각각의 모든 체크포인트 경로를 하나씩 번갈아가면서 사용한다.

메모리 테이블스페이스의 체크포인트 경로 변경

이 절에서는 메모리 테이블스페이스의 체크포인트 경로를 변경하는 절차에 대해 알아본다. 메모리 테이블스페이스를 위한 체크포인트는 컨트롤(CONTROL) 구동 단계에서만 설정이 가능하다. 앞서 ‘메모리 테이블스페이스에 체크포인트 경로 추가’ 절에서 알아본 것과 같이 우선 Altibase 서버를 종료 시킨 후에 컨트롤 단계까지 Altibase를 기동한다.

본 예제는 기존의 체크포인트 경로인 Altibase 홈 디렉토리 아래의 dbs1을 새로 설치한 디스크인 /new_disk로 옮기는 절차를 보여준다.

체크포인트 경로를 변경하기 위해서는 기존의 체크포인트 경로를 절대경로로 정확히 입력해 주어야 한다. 컨트롤 단계에서 테이블스페이스의 체크포인트 경로를 보는 방법은 위의 ‘메모리 테이블스페이스에 체크포인트 경로 추가’절을 참고한다.

체크포인트 경로를 변경할 때에는 체크포인트 경로를 추가할때와 마찬가지로 다음과 같이 변경할 체크포인트 경로와 디렉토리를 DBA가 직접 생성하고, 그 디렉토리에 대한 쓰기 및 실행 권한을 Altibase 프로세스를 실행하는 OS 사용자 계정에 부여해야 한다. Altibase 프로세스를 실행하는 사용자 계정이 altibase라고 가정한다.

```
$ su - root  
$ mkdir /new_disk/dbs1  
$ chown altibase /new_disk/dbs1
```

이제, 다음과 같이 RENAME CHECKPOINT PATH를 이용하여 Altibase 홈의 dbs1이라는 체크포인트 경로를 새로 추가한 디스크인 /new_disk/dbs1으로 변경할 수 있다.

```
iSQL(sysdba)> ALTER TABLESPACE USER_MEM_TBS  
RENAME CHECKPOINT PATH '/opt/altibase_home/dbs1' TO '/new_disk/dbs1';  
Alter success.
```

마지막으로 기존의 Altibase 홈 디렉토리의 dbs1안의 USER_MEM_TBS 테이블스페이스의 체크포인트 이미지들을 모두 /new_disk/dbs1으로 옮겨주어야 한다.

```
$ mv $ALTIBASE_HOME/dbs1/USER_MEM_TBS* /new_disk/dbs1
```

메모리 테이블스페이스의 체크포인트 경로 제거

이 절에서는 메모리 테이블스페이스의 체크포인트 경로를 제거하는 절차에 대해 알아본다. 메모리 테이블스페이스를 위한 체크포인트는 컨트롤(CONTROL) 구동 단계에서만 설정이 가능하다. 앞서 ‘메모리 테이블스페이스에 체크포인트 경로 추가’절에서 알아본 것과 같이 우선 Altibase 서버를 종료 시킨 후 컨트롤 단계까지 Altibase를 기동한다.

본 예제는 기존의 체크포인트 경로인 Altibase 홈 디렉토리의 dbs2를 제거하는 절차를 보여준다.

체크포인트 경로를 변경하기 위해서는 기존의 체크포인트 경로를 절대경로로 정확히 입력해 주어야 한다. 컨트롤 단계에서 테이블스페이스에 속한 체크포인트 경로를 보는 방법은 위의 ‘메모리 테이블스페이스에 체크포인트 경로 추가’절을 참고한다

이제, 다음과 같이 DROP CHECKPOINT PATH 명령을 이용하여 Altibase 홈 디렉토리의 dbs2라는 체크포인트 경로를 제거할 수 있다.

```
iSQL(sysdba)> ALTER TABLESPACE USER_MEM_TBS  
DROP CHECKPOINT PATH '/opt/altibase_home/dbs2'  
Alter success.
```

마지막으로 기존의 Altibase 홈 디렉토리의 dbs2안의 USER_MEM_TBS 테이블스페이스를 위한 체크포인트 이미지들을 USER_MEM_TBS의 다른 체크포인트 경로중 하나로 옮겨주어야 한다.

```
$ mv $ALTIBASE_HOME/dbs2/USER_MEM_TBS* /new_disk/dbs1
```

메모리 테이블스페이스의 자동확장 설정 변경

이 절에서는 메모리 테이블스페이스의 자동확장 설정을 변경하는 절차에 대해 알아본다. 메모리 테이블스페이스 생성시 자동확장 구문을 명시하지 않으면, 기본으로 해당 테이블스페이스는 자동확장이 되지 않도록 설정된다.

```
iSQL> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 256M;  
Create success.
```

이 때 자동확장을 하도록 설정하는 가장 간단한 구문은 다음과 같다.

```
iSQL> ALTER TABLESPACE USER_MEM_TBS  
ALTER AUTOEXTEND ON;  
Alter success.
```

위 예에서 테이블스페이스는 메모리 테이블스페이스의 기본 확장단위인 EXPAND_CHUNK_PAGE_COUNT 프로퍼티에 설정한 페이지 개수에 해당하는 크기만큼씩 확장될 것이다.

또한 최대 크기로 UNLIMITED를 지정한 것과 같이 동작하여 시스템내의 모든 메모리 테이블스페이스의 현재 크기의 총합이 MEM_MAX_DB_SIZE 프로퍼티를 벗어나지 않는 범위에서 테이블스페이스는 확장될 것이다.

메모리 테이블스페이스는 디스크 테이블스페이스와 달리, 체크포인트 이미지 파일을 DBA가 직접 관리할 필요가 없다. 자동확장이 필요한 경우 Altibase가 체크포인트 이미지파일을 자동으로 생성하기 때문이다.

메모리 테이블스페이스의 확장단위를 지정하기 위해서는 다음과 같은 구문을 사용한다.

```
iSQL> ALTER TABLESPACE USER_MEM_TBS  
ALTER AUTOEXTEND ON NEXT 128M;  
Alter success.
```

메모리 테이블스페이스의 최대 크기를 지정하기 위해서는 다음과 같은 구문을 사용한다.

```
iSQL> ALTER TABLESPACE USER_MEM_TBS  
ALTER AUTOEXTEND ON MAXSIZE 1G;  
Alter success.
```

메모리 테이블스페이스의 확장 크기와 최대 크기를 함께 지정하기 위해서는 다음과 같은 구문을 사용한다.

```
iSQL> ALTER TABLESPACE USER_MEM_TBS  
ALTER AUTOEXTEND ON NEXT 128M MAXSIZE 1G;  
Alter success.
```


메모리 테이블스페이스의 자동확장 설정을 끄기 위해서는 다음과 같은 구문을 사용한다.

```
iSQL> ALTER TABLESPACE USER_MEM_TBS  
ALTER AUTOEXTEND OFF;  
Alter success.
```

메모리 테이블스페이스의 온라인(ONLINE) 및 오프라인(OFFLINE)으로의 전이

본 예제는 메모리 테이블스페이스를 온라인 상태에서 오프라인 상태로, 그리고 그 반대로 전이하는 방법을 보여준다.

Altibase 메모리 테이블스페이스의 모든 데이터는 메모리에 적재된다. 이로 인하여 메모리 테이블스페이스가 사용중인 공간만큼 시스템의 메모리가 할당된다. Altibase는 이러한 메모리 사용을 DBA가 손쉽게 제어할 수 있도록, 메모리 테이블스페이스에 메모리를 할당하고 메모리를 반납할 수 있는 기능을 제공한다.

물론, 메모리 테이블스페이스의 메모리가 반납된 상황에서는 해당 테이블스페이스 안에 생성된 모든 객체를 일시적으로 사용할 수 없게 된다. 메모리 테이블스페이스의 메모리를 반납하기 위해서는 테이블스페이스를 오프라인으로 전이하면 된다.

```
iSQL> ALTER TABLESPACE USER_MEM_TBS OFFLINE;  
Alter success.
```

추후 해당 메모리 테이블스페이스 안에 생성된 테이블을 사용하기 위해서는 다음과 같이 테이블스페이스를 온라인으로 전이한다.

```
iSQL> ALTER TABLESPACE USER_MEM_TBS ONLINE;  
Alter success.
```

휘발성 테이블스페이스

휘발성 테이블스페이스 생성

기본적으로 휘발성 테이블스페이스 생성, 변경, 삭제 구문은 메모리 테이블스페이스의 생성, 변경, 삭제 구문과 거의 동일하다. 다만 체크포인트 이미지 파일 관련 구문은 사용할 수 없다는 차이점이 있다.

다음 구문으로 256M 크기의 휘발성 테이블스페이스를 생성할 수 있다.

```
iSQL> CREATE VOLATILE DATA TABLESPACE USER_VOL_TBS  
SIZE 256M;  
Create success.
```

이 경우 테이블스페이스의 크기는 256MB로 고정되어 있어 자동확장은 불가능하다.
다음 구문은 자동 확장되는 테이블스페이스를 생성한다.

```
iSQL> CREATE VOLATILE DATA TABLESPACE USER_VOL_TBS  
SIZE 256M AUTOEXTEND ON;  
Create success.
```

이 경우 테이블스페이스의 초기 크기는 256MB이지만 자동 확장 되며,
VOLATILE_MAX_DB_SIZE 프로퍼티의 값이 허용하는 만큼 확장될 수 있다. 자동 확장
단위는 4MB이다. 자동 확장 단위를 8MB로, 최대 512MB까지만 확장할 수 있게 하려면
다음 구문으로 생성하면 된다.

```
iSQL> CREATE VOLATILE DATA TABLESPACE USER_VOL_TBS  
SIZE 256M AUTOEXTEND ON NEXT 8M MAXSIZE 512M;  
Create success.
```

휘발성 테이블스페이스 변경

휘발성 테이블스페이스에 대해 자동 확장 모드, 자동 확장 단위, 최대 크기 등을
변경할 수 있다. 다음 구문은 자동 확장이 안되는 휘발성 테이블스페이스를 자동 확장
모드로 변경한다.

```
iSQL> ALTER TABLESPACE USER_VOL_TBS ALTER AUTOEXTEND ON;  
Alter success.
```

다음 구문은 자동 확장 모드로 바꾸면서 자동 확장 단위를 8MB로, 최대 512MB까지
확장할 수 있도록 변경한다.

```
iSQL> ALTER TABLESPACE USER_VOL_TBS ALTER  
AUTOEXTEND ON NEXT 8M MAXSIZE 512M;  
Alter success.
```

다음 구문은 자동 확장 모드를 끈다. 이 구문은 자동 확장 모드가 켜진
테이블스페이스에 사용할 수 있다.

```
iSQL> ALTER TABLESPACE USER_VOL_TBS ALTER AUTOEXTEND OFF;  
Alter success.
```

테이블스페이스의 삭제(DROP) - 디스크, 메모리, 휘발성 공통

테이블스페이스 폐기(Discard) - 데이터가 손상된 테이블스페이스를 제거

이 절에서는 테이블스페이스를 폐기하는 절차에 대해 알아본다.

DBA가 실수로 디스크 테이블스페이스의 데이터 파일이나 메모리 테이블스페이스의 체크포인트 이미지 파일을 삭제한 경우, 혹은 매체 오류로 인하여 해당 파일의 내용이 유실된 경우에 Altibase 기동이 불가능해진다.

이런 경우에, 매체 복구를 통해서 해당 파일을 복원하는 방법을 먼저 생각해 볼 수 있다. 하지만 매체 복구는 아카이브 로깅이 수행되어 기존의 로그파일이 별도의 아카이브 공간에 모두 남아 있을 때에만 가능하다.

이와 같이 매체 복구가 불가능한 경우, 데이터 파일이나 체크포인트 이미지 파일이 유실된 특정 테이블스페이스를 ALTER TABLESPACE DISCARD 구문을 사용해서 폐기시켜 버리고 나머지 테이블스페이스들만으로 Altibase를 기동할 수 있다.

테이블스페이스가 ALTER TABLESPACE DISCARD 구문을 이용하여 한 번 폐기되면, 그 안에 생성된 객체에 대한 접근이 불가능해지고, 추후 그 테이블스페이스에 대해 할 수 있는 액션은 그것을 제거(DROP)하는 것뿐이다. 그러므로 이 구문 사용시 신중해야 한다.

다음 예제는 메모리 테이블스페이스인 USER_MEM_TBS를 생성한 후, 체크포인트 이미지를 삭제한 상태에서 해당 테이블스페이스만 제외한 나머지 테이블스페이스들로 Altibase를 기동하는 과정을 보여준다.

우선 다음과 같이 메모리 테이블스페이스를 생성한다.

```
iSQL> CREATE MEMORY TABLESPACE USER_MEM_TBS SIZE 256M;  
Create success.
```

이제 Altibase를 종료하고, 해당 테이블스페이스에 속한 체크포인트 이미지 파일을 지운다. 이제 Altibase를 재기동하면 다음과 같은 에러가 발생한다.

```
[SM-WARNING] CANNOT IDENTIFY DATAFILE  
[TBS:USER_MEM_TBS, PPID-0-FID-0] Datafile Not Found  
  
[SM-WARNING] CANNOT IDENTIFY DATAFILE  
[TBS:USER_MEM_TBS, PPID-1-FID-0] Datafile Not Found  
  
[FAILURE] The data file does not exist.  
Startup Failed....  
[ERR-91015 : Communication failure.]
```

Altibase는 이와 같이 일부 테이블스페이스의 데이터 파일이나 체크포인트 이미지 파일이 존재하지 않는 경우 에러를 발생시킨다.

이제 USER_MEM_TBS를 디스카드 할 차례이다.

디스카드 구문은 컨트롤(CONTROL) 단계에서만 수행이 가능하다. Altibase를 컨트롤 단계까지 기동한다.

```
$ isql -u sys -p manager -sysdba
iSQL(sysdba)> startup control
```

이제 체크포인트 이미지 파일이 유실된 테이블스페이스 USER_MEM_TBS를 디스카드한다.

```
iSQL(sysdba)> ALTER TABLESPACE USER_MEM_TBS DISCARD;
Alter success.
```

그리고 다음과 같이 STARTUP SERVICE명령을 수행하여 Altibase를 서비스 단계로 진입시킨다.

```
iSQL(sysdba)> startup service
Command execute success.
```

디스카드 명령은 해당 테이블스페이스를 버리겠다고 선언하는 것에 불과하다. 그렇기 때문에 해당 테이블스페이스 및 그 안의 객체는 DROP TABLESPACE INCLUDING CONTENTS 구문을 이용하여 직접 제거하여야 한다.

```
iSQL> DROP TABLESPACE USER_MEM_TBS
INCLUDING CONTENTS AND DATAFILES;
Drop success.
```

마찬가지로 디스크 테이블스페이스의 데이터 파일이 유실되거나, 매체 오류 때문에 데이터 파일의 내용중 일부가 깨진 경우에도 해당 테이블스페이스를 디스카드시키는 방법으로 Altibase를 가동할 수 있다.

테이블스페이스의 제거

본 사용 예에서는 테이블스페이스를 제거하는 방법을 보여준다.

테이블스페이스 안에 어떠한 객체도 생성되어 있지 않은 경우, 다음과 같이 간편하게 테이블스페이스를 제거할 수 있다. 단, 이 경우 디스크 테이블스페이스의 데이터 파일과 메모리 테이블스페이스의 체크포인트 이미지 파일은 파일 시스템에서 제거되지 않는다.

```
iSQL> DROP TABLESPACE MY_TBS;
Drop success.
```

만약, 테이블스페이스 안에 객체가 존재한다면 다음과 같이 INCLUDING CONTENTS절을 주어서 테이블스페이스 안의 모든 객체가 삭제되도록 한다. 이 경우에도 데이터 파일이나 체크포인트 이미지 파일이 파일 시스템에서 삭제되지는 않는다.

```
iSQL> DROP TABLESPACE MY_TBS  
INCLUDING CONTENTS;  
Drop success.
```

만약 삭제하려는 테이블스페이스의 테이블을 참조하는 참조 제약(Referential Constraint)을 제거하려면 다음과 같이 INCLUDING CONTENTS절과 함께 CASCADE CONSTRAINTS를 쓰면 된다. 이 경우에도 데이터 파일이나 체크포인트 이미지 파일이 파일 시스템에서 삭제되지 않는다.

```
iSQL> DROP TABLESPACE MY_TBS  
INCLUDING CONTENTS CASCADE CONSTRAINTS;  
Drop success.
```

디스크 테이블스페이스의 데이터 파일이나 메모리 테이블스페이스의 체크포인트 이미지 파일을 삭제하려면 다음과 같이 INCLUDING CONTENTS절 바로 뒤에 AND DATAFILES절을 주면 된다.

```
iSQL> DROP TABLESPACE MY_TBS  
INCLUDING CONTENTS AND DATAFILES;  
Drop success.
```

```
iSQL> DROP TABLESPACE MY_TBS  
INCLUDING CONTENTS AND DATAFILES  
CASCADE CONSTRAINTS;  
Drop success.
```

테이블스페이스 공간 관리

이 절에서는 Altibase의 테이블스페이스 공간을 관리하는 방법에 대하여 설명한다.

언두 테이블스페이스 크기 계산

언두 테이블스페이스(Undo Tablespace)는 언두 세그먼트를 저장하기 위해 사용된다. 언두 테이블스페이스가 부족할 경우 트랜잭션 성능에 영향을 줄 수 있으므로 DBA는 이를 적절한 크기로 관리해야 한다.

만약 업무 시스템에서 변경 트랜잭션, 특히 오랜 시간동안 구문이 실행되는 트랜잭션이 자주 발생하는 경우에는 언두 세그먼트가 계속 확장될 것이다. 이는 언두 테이블스페이스의 공간 부족을 초래할 수 있다.

사용자는 언두 테이블스페이스를 자동 확장 모드로 설정하거나, 대략적인 최대 크기를 예측하여 그 예측치를 최대 크기로 지정한 고정 크기 모드로 설정할 수 있다.

언두 테이블스페이스의 자동 확장 모드

사용자가 처음으로 애플리케이션을 수행하는 경우 얼마만큼의 언두 테이블스페이스 공간이 필요한지 알기가 쉽지 않다. 이런 경우에는 언두 테이블스페이스의 자동 확장 모드를 활성화하여 필요한 공간까지 자동으로 확장되도록 한다.

Altibase는 애플리케이션 개발 환경에서 언두 테이블스페이스의 용량을 계획하기 쉽도록 자동 확장 모드를 제공한다. 기본적으로 언두 테이블스페이스는 자동 확장 모드로 설정되며, ALTER TABLESPACE 구문으로 변경할 수 있다.

언두 테이블스페이스의 고정 크기 모드

사용자가 고정 크기의 언두 테이블스페이스를 사용하고 싶다면, 필요한 용량을 예측해야 한다. 이를 위해서 사용자는 애플리케이션이 수행되는 동안 사용되는 TSS 세그먼트 공간과 언두 세그먼트 공간의 사용 패턴을 수집하여 분석해야 한다.

필요한 언두 테이블스페이스 크기는 일반적으로 다음과 같은 계산식으로 대략적인 산출이 가능하다.

- 언두 테이블스페이스 크기 =
 $\text{Long-Term 트랜잭션 수행 시간(sec)} \times (\text{초당 할당되는 언두 페이지 개수} + \text{초당 할당되는 TSS 페이지 개수}) \times \text{페이지 크기(8KB)}$

예를 들어 Long-Term 트랜잭션의 수행 시간이 600초(10분)이고 초당 언두 페이지 1000개와 TSS 페이지 24개가 할당된다면, $10 \times 60 \times (1000 + 24) \times 8K = 4800MB$ 이므로 약 4.7G 정도가 필요하다.

이와 같이 예측하는 것이 어렵다면, 디스크 공간이 허락하는 한 충분히 넉넉한 크기를 언두 테이블스페이스에 할당하는 것도 방법이다.

언두 테이블스페이스의 확장

업무 시스템에서 변경 트랜잭션 (특히 Long-Term 트랜잭션, 즉 트랜잭션이 커밋되기까지 긴 시간이 소요되는 트랜잭션)이 자주 발생하는 경우에 언두 테이블스페이스의 공간 부족이 발생할 수 있다. 이러한 경우에 ALTER TABLESPACE 구문을 이용하여 언두 테이블스페이스에 적정한 크기의 데이터 파일을 추가하거나 파일의 크기를 적당히 늘려준다.

메모리 테이블의 크기 추정

데이터 크기 계산

메모리 테이블의 데이터 크기는 각 칼럼의 데이터 타입, 칼럼 정렬 (alignment)를 위한 패딩(padding) 등에 기반해서 예측이 가능하다. 수학 공식으로 표현하면 다음과 같다:

데이터 크기 = [(각 칼럼의 추정 크기의 합 + 각 칼럼을 위한 패딩 크기의 합) * 데이터 레코드 개수]

데이터 타입 별 추정 크기는 다음 표에서 보여준다.

(P = Precision, V = Value length)

자료형	예측 칼럼 크기
INTEGER	4
SMALLINT	2
BIGINT	8
DATE	8
DOUBLE	8
CHAR	2 + P
VARCHAR	22 + V
NCHAR	2+(P * 2)-UTF8 2+(P * 3)-UTF16
NVARCHAR	22+(V * 2)-UTF8 22+(V * 3)-UTF16
BIT	4 + (P/8)
VARBIT	22 + (P/8)
FLOAT	3 + (P+2)/2
NUMERIC	3 + (P+2)/2

위 도표에서 P(Precision)는 테이블 생성시 결정된 칼럼의 크기를 가리킨다. P보다 긴 데이터는 해당 데이터 타입의 칼럼에 입력될 수 없다. V(Value length)는 입력된 데이터의 실제 길이로, V는 P보다 클 수 없다.

CHAR, NCHAR, BIT 타입 같은 고정 길이 칼럼은 P 만큼의 공간을 항상 점유하므로, 칼럼의 길이는 데이터의 실제 길이에 상관없이 고정된다. 그러나 VARCHAR, NVARCHAR,

VARCHAR같은 가변 길이 칼럼은 점유하는 공간이 데이터 길이에 따라서 가변적이다.

디스크 테이블과 달리, 메모리 테이블은 데이터 접근 속도를 높이기 위한 패딩 공간을 포함한다. 이 공간의 크기는 데이터 타입과 칼럼의 위치에 따라서 가변적이다.

인덱스 크기 추정

메모리 인덱스는 테이블 데이터가 저장되는 테이블스페이스에 저장되지 않는다. 대신에 이는 메모리 공간에 독립적으로 저장된다. 데이터 저장 위치를 가리키는 포인터가 메모리 인덱스 노드의 각 버킷에 저장되기 때문에, 인덱스 크기는 데이터 타입에 상관없이 포인터 크기와 테이블에 현재 저장된 레코드의 개수에 기반하여 추정할 수 있다.

$$\text{인덱스 크기} = (\text{데이터 레코드의 개수}) * p$$

(p = 포인터 크기)

위 공식에서 p는 포인터 크기, 즉 한 포인터를 저장하는데 필요한 크기이다. 32-bit 시스템에서는 이 크기가 4바이트이고, 64-bit 시스템에서는 이 크기가 8바이트이다. 이 공식에서, 인덱스의 크기는 모든 리프 노드 (즉, B트리의 최하위의 노드)의 총 크기만큼이다. B트리에는 리프 노드에 더해서 인터널 노드 (즉, 리프 노드의 상위 노드)가 있는데, 이의 총 크기는 리프 노드 크기의 1/128 에 불과하므로 무시해도 된다. 인덱스 관리에 사용되는 추가 정보를 저장하는 크기도 리프 노드 크기의 1/16 정도로 무시할만큼 작다. 그러므로 모든 리프 노드의 총 크기에 기반해서 인덱스의 총 크기를 계산하면 된다.

그러나 이 공식은 리프 노드의 모든 버킷이 그 안에 키 값을 가지고 있는 상황만을 고려한 것이기 때문에, 이 공식을 이용해서 추정된 값은 인덱스의 실제 크기와 다를 수 있다. 즉, 노드 내에 빈 버킷이 많이 있다면 인덱스의 실제 크기는 추정된 크기보다 많이 클 수 있다. 이 경우 인덱스를 재구축해서 인덱스 크기를 줄일 수 있다.

예제 1

아래처럼 테이블이 생성된 경우 데이터 크기를 추정해 보자.

```
CREATE TABLE T1 ( C1 Integer, C2 char(1024), C3 varchar(1024) )
tablespace user_data01;
```

이 테이블의 칼럼 C1과 칼럼 C2는 고정 길이 칼럼이고 C3는 가변 길이 칼럼이다. 그러므로 한 레코드의 크기는 칼럼 C3에 따라 변할 것이다. 이를 고려하여 한 레코드의 크기를 계산하면, 아래처럼 T1테이블의 데이터 크기는 (한 레코드 크기 * 레코드 개수)와 같다.

[레코드 헤더] = 32 바이트
 [C1 칼럼] = 4 바이트
 [C2 칼럼] = 2+P 바이트 = 2+1024 바이트
 [C3 칼럼] = 22+V 바이트

- 칼럼 C3의 데이터 길이가 200바이트이면:

[총 길이] = 32 + (4) + (2+1024) + (22+200) + padding
 = (1284 + padding) 바이트

- 칼럼 C3의 데이터 길이가 500바이트이면:

[총 길이] = 32 + (4) + (2+1024) + (22+500) + padding
 = (1584 + padding) 바이트

예제 2

아래 구문으로 생성된 테이블 T1의 인덱스 크기를 계산해 보자. 테이블 T1에는 현재 500,000 레코드가 있고, 시스템은 64-bit이다.

```
CREATE TABLE T1 ( C1 Integer, C2 char(300), C3 varchar(500))
tablespace user_data01;
CREATE INDEX T1_IDX1 ON T1( C1, C2, C3 );
```

[index size] = 500,000 records * 8 = 3.814 Megabytes

예제 3

아래 구문으로 생성된 테이블의 데이터와 인덱스 크기를 계산해 보자. 테이블에는 현재 1,000,000 레코드가 있고, 시스템은 64-bit이다.

```
CREATE TABLE TEST001 (
C1 char(8) primary key,
C2 char(128), N1 integer,
IN_DATE date)
tablespace user_data01;
```

- 한 레코드의 크기와 총 데이터 크기

[총 길이] = 32[헤더] + (2+8) + (2+128) + (4) + (8) = 184 바이트
 [전체 데이터 크기] = [184] * 1,000,000 records
 = 175.47 MB

인덱스 크기

[전체 인덱스 크기] = 8 * 1,000,000 records = 7.629 MB

이 값은 데이터 크기와 리프 노드의 크기에 기반해서 계산되었기 때문에, 실제로는 페이지 헤더, 인덱스 노드, 그리고 프리 (free) 페이지 관리를 위한 메모리에 사용되는 공간이 추가로 있을 것이다.

디스크 테이블의 크기 추정

Altibase의 디스크 테이블 크기는 자료형과 데이터의 구성을 바탕으로 계산할 수 있으며 [테이블 로우의 총 길이 * 데이터 건 수] 의 값을 가진다. 다음 표는 자료형 별 길이를 보여준다.

(P = Precision, V = Value length)

자료형	추정되는 칼럼 크기		
	Null	250바이트 이하	251바이트 이상
Integer	1	5	X
SmallInt	1	3	X
BigInt	1	9	X
Date	1	9	X
Double	1	9	X
Char	1	1+P	3+P
Varchar	1	1+V	3+V
NChar	1	1+P	3+P
NVarchar	1	1+V	3+V
Bit	1	5+(P/8)	7+(P/8)
Varbit	1	5+(V/8)	7+(V/8)
Float	1	4+(V+2) / 2	6+(V+2) / 2
Numeric	1	4+(V+2) / 2	6+(V+2) / 2

위 도표에서 P(Precision)는 테이블 생성시 결정된 칼럼의 최대 크기이다. P 보다 큰 길이를 갖는 데이터는 그 타입의 칼럼에 입력되지 않는다. 또한 고정길이 칼럼인

CHAR, NCHAR, BIT 등은 항상 P 만큼의 공간을 점유하기 때문에, 데이터의 길이와 상관없이 칼럼의 길이는 일정하다.

V(Value length)는 실제로 삽입된 데이터의 실제 길이로, P보다 클 수 없다. 또한 가변 길이 칼럼인 VARCHAR, NVARCHAR, VARVIT등은 데이터의 크기에 따라 점유하는 공간의 크기가 달라진다. 따라서 데이터의 크기에 따라 칼럼의 크기가 변한다.

로우(row) 크기 추정

이 절에서는 테이블 스키마가 다음과 같을 경우, 로우 크기를 계산하는 방법에 대해 설명한다.

```
CREATE TABLE T1 ( C1 char(32), C2 char(1024), C3 varchar(512) )  
tablespace user_data02;
```

이 스키마에서 C1 칼럼과 C2 칼럼은 고정길이 칼럼이며, C3 칼럼은 가변길이 칼럼이다. 따라서 C3 칼럼의 크기에 따라 로우의 크기가 변한다. 또한 칼럼에 널(NULL)이 존재하는지 여부에 따라 로우의 크기가 변한다. 이를 고려하여 로우의 크기를 계산하면 아래와 같으며, 테이블 T1의 크기는 (한 로우의 총 길이 * 데이터 건 수)가 된다.

[로우 헤더] 34 바이트

[C1 칼럼] 1+P 바이트 = 1+32 바이트

[C2 칼럼] 3+P 바이트 = 3+1024 바이트

[C3 칼럼] 3+V 바이트

- C3 칼럼의 데이터 크기가 200 바이트인 경우

[총 길이] $34 + (1+32) + (3+1024) + (1+200) = 1295$ 바이트

- C3 칼럼의 데이터 크기가 500 바이트인 경우

[총 길이] $34 + (1+32) + (3+1024) + (3+500) = 1597$ 바이트

- C2 칼럼이 널이고, C3 칼럼의 데이터 크기가 300 바이트인 경우

[총 길이] $34 + (1+32) + (1) + (3+300) = 371$ 바이트

- C3 컬럼이 널인 경우

[총 길이] $34 + (1+32) + (3+1024) + (0) = 1094$ 바이트

마지막 컬럼이 널(null)이고 데이터가 없는 경우, 마지막 칼럼은 저장하지 않으므로 크기에 반영되지 않는다.

인덱스(index) 크기 추정

인덱스의 크기 역시 자료형과 데이터의 구성을 바탕으로 계산할 수 있다. 다음 표는 인덱스 크기 계산 시 사용할 자료형 별 길이를 보여준다.

(P = Precision, V = Value length)

자료형	인덱스 키의 크기		
	Null	250바이트 이하	251바이트 이상
Integer	4	4	X
SmallInt	2	2	X
BigInt	8	8	X
Date	8	8	X
Double	8	8	X
Char	1	1+P	3+P
Varchar	1	1+V	3+V
NChar	1	1+P	3+P
NVarchar	1	1+V	3+V
Bit	1	5+(P/8)	7+(P/8)
Varbit	1	5+(V/8)	7+(V/8)
Float	1	4+(V+2) / 2	6+(V+2) / 2
Numeric	1	4+(V+2) / 2	6+(V+2) / 2

위 도표에서 P(Precision)와 V(Value length)는 각각 테이블 생성시 결정된 칼럼의 최대 크기와 실제로 삽입된 데이터의 크기를 의미한다.

한 인덱스의 크기는 다음과 같다.

$$[10(\text{헤더 길이}) + (\text{키 칼럼 길이의 합})] * \text{데이터 건 수}$$

위의 계산 공식은 leaf node (B*Tree에서 최하위의 노드)의 대략적인 크기이다. 이외에 internal node(leaf node의 상위 노드)의 크기의 경우, 키 칼럼(key column)의 크기가 작을 경우에는 무시할 수 있을 정도로 그 크기가 작다.

하지만 키 칼럼의 크기가 2K이상일 경우에는 B*Tree의 깊이가 깊어지고, internal node의 크기가 전체 Leaf Node 크기의 50% 정도까지 될 수 있으므로 이런 경우에는 internal node의 크기를 계산에 포함해야 한다.

다음은 테이블 및 인덱스의 스키마가 다음과 같을 경우, 인덱스의 크기를 계산하는 방법을 보여준다.

```
CREATE TABLE T1 ( C1 Integer, C2 varchar(500)) tablespace user_data02;  
CREATE INDEX T1_IDX1 ON T1( C1, C2 );
```

C1 칼럼은 Integer 형이므로 항상 4 byte로 크기가 결정된다. C2는 가변길이 칼럼으로 데이터의 크기에 따라 길이가 가변적으로 변한다.

[키 헤더] 10 byte
[C1 칼럼] 4 byte
[C2 칼럼] 1+V byte

- C2 칼럼의 데이터 크기가 50 바이트인 경우

[총 길이] $10 + 4 + (1+50) = 65$ 바이트

- C2 칼럼의 데이터 크기가 500 바이트인 경우

[총 길이] $10 + 4 + (3+500) = 517$ 바이트

- C2 칼럼이 널인 경우

[총 길이] $10 + 4 + 1 = 15$ 바이트

테이블 크기 계산 예제

테이블 스키마가 다음과 같고 데이터 건 수가 100만 건일 경우에, 로우의 크기와 인덱스의 크기를 포함한 테이블의 크기를 다음과 같이 계산한다.

```
CREATE TABLE TEST001 (
C1 char(8) primary key,
C2 char(128), N1 integer,
IN_DATE date)
tablespace user_data02;
```

- 로우 크기 및 전체 데이터 크기

로우 크기: $34[\text{헤더}] + (1+8) + (1+130) + (1+4) + (1+8) = 188$ 바이트

전체 데이터 크기: $[188] * 100\text{만 건} = 179.29 \text{ M 바이트}$

- 인덱스 크기

한 로우의 인덱스 크기: $10[\text{헤더}] + (1+8)[C1] = 19$ 바이트

전체 인덱스 크기: $19 * 100\text{만건} = 18.12 \text{ M 바이트}$

- TEST001 테이블 전체가 차지하는 디스크 크기

$179.29 (\text{데이터 크기}) + 18.12 (\text{인덱스 크기}) = 197.41 \text{ M 바이트}$

이것은 데이터의 크기만 계산한 것으로, 실제로는 페이지 헤더, internal node, 세그먼트 관리 영역 등을 추가로 사용하며 그만큼 공간을 더 사용한다. 이러한 부분을 고려하면, 총 테이블스페이스는 약 240M 바이트를 사용하게 될 것이다.

테이블 저장 공간 계산

위에서 테이블의 크기 계산에 사용된 테이블 TEST001을 기준으로 테이블의 레코드와 인덱스를 모두 저장할 수 있는 테이블의 적정 사이즈를 계산한다. 적정한 테이블 크기를 계산하기 위해서 다음과 같은 사항을 고려해야 한다.

트랜잭션의 유형의 상대 빈도를 고려한다

특정 테이블에 대하여 갱신 (Update) 트랜잭션이 많이 발생할 경우에는 트랜잭션의 성능을 높이기 위해 PCTFREE를 크게 하고, PCTUSED를 작게 하여 변경 작업을 위해 필요한 빈 공간(free space)을 많이 확보하는 것이 좋다.

변경 트랜잭션이 적고 입력(Insert) 트랜잭션이 주로 발생하는 테이블의 경우에는 반대로 PCTFREE를 작게하고, PCTUSED를 크게 하여 불필요한 빈 공간(free space)을 최소화해야 한다.

- PCTFREE

기본값은 10으로, 디스크 테이블 생성시 0에서 99 사이의 값을 명시할 수 있다. 이는 테이블의 각 페이지에서 기존 레코드에 대한 변경 연산 등을 위하여 미리

예약한 여유 공간의 비율이다. 따라서 PCTFREE가 10이고 입력(Insert) 트랜잭션만 발생한다고 가정할 경우에 테이블의 전체 크기가 100M 라면 테이블의 레코드와 인덱스를 위해 사용될 수 있는 공간은 90M가 된다.

• PCTUSED

기본값은 40이며 디스크 테이블 생성시 0에서 99 사이의 값을 명시할 수 있다. 특정 페이지가 PCTFREE에서 명시한 비율에 도달한 후에 변경과 삭제로 인해서 빈 공간의 비율이 40% 미만(39%)으로 될 때까지 해당 페이지에는 삽입 연산이 일어나지 않는다. 따라서 변경이 많이 발생하는 테이블일 경우에는 테이블의 크기를 산정할 때 여유 공간을 많이 확보해야 한다.

상황	테이블 사이즈 계산
대부분이 Read Only 트랜잭션이거나 UPDATE 시에 레코드의 크기가 증가되지 않을 경우	PCTFREE를 5로 지정하고, PCTUSED를 90으로 지정한 경우 ① 최소 테이블 크기 계산: TEST001(전체사이즈=215.53M)테이블을 저장하기 위해서 필요한 최소 사이즈는 다음과 같은 공식으로 계산한다. 테이블 전체 사이즈 / [1- (PCTFREE / 100)] = $215.53 / 0.95 \approx 227\text{M}$ ② 가중치 계산: 최소 사이즈에 적절한 크기의 가중치를 추가한다. 가중치는 시스템 상황에 따라 달라질 수 있다. 다음은 가중치 계산의 한가지 예이다. 최소사이즈 * [1- (PCTUSED / 100)] * 2 = $227 * 0.1 * 2 \approx 45\text{M}$ ③ 따라서 테이블을 총 272M 정도의 사이즈로 생성한다.
UPDATE가 빈번하고, UPDATE 시에 레코드의 크기가 증가될 경우	PCTFREE를 20으로 지정하고, PCTUSED를 40으로 지정한 경우 ① 최소 테이블 크기 계산: TEST001(전체사이즈=213.63M)테이블을 저장하기 위해서 필요한 최소사이즈는 다음과 같은 공식으로 계산한다. 테이블전체사이즈 / [1- (PCTFREE / 100)] = $213.63 / 0.8 \approx 267\text{M}$ ② 가중치 계산: 최소 사이즈에 적절한 가중치를 추가한다. 다음은 가중치 계산의 한가지 예이다. 최소 사이즈 * [1- (PCTUSED / 100)] * 2 = $267 * 0.6 * 2 \approx 320\text{M}$ ③ 따라서 테이블을 총 587M 정도의 사이즈로 생성한다.
INSERT와 UPDATE 가 자주 발생하지만 UPDATE를 할 때 레코드의 크기가 증가되지 않을 경우	PCTFREE를 10으로 지정하고, PCTUSED를 60으로 지정한다.

[표 6-3]트랜잭션 유형의 상대 빈도에 따른 테이블 크기 계산

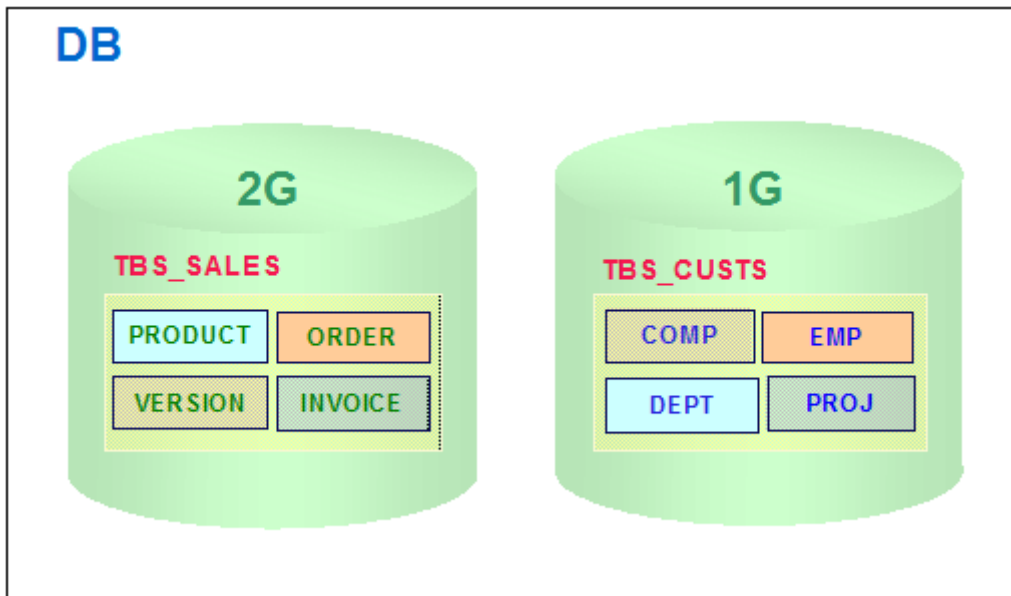
Note: 위의 표에서 설명한 테이블의 사이즈 계산은 절대적인 기준이 아니다. 시스템이 비정상 동작하여 데이터가 갑자기 증가하는 등의 장애 상황에 대한 고려가 필요하다.

적정한 백업 사이즈를 고려한다.

실제 업무에서 하나의 테이블스페이스에 하나의 테이블만 저장되는 경우는 드물다. 업무단위 또는 백업단위로 테이블을 묶어서 하나의 테이블스페이스에 생성하는 것이 더 효율적이다.

이 경우 테이블스페이스에 대한 백업 소요 시간 등을 고려하여 한 개 테이블스페이스의 적정사이즈를 설정해야 한다.

다음 그림은 데이터베이스 내에서 테이블스페이스를 업무단위 및 백업사이즈를 고려하여 적정사이즈로 분리하여 생성한 것을 보여준다.



[그림 6-15] 백업을 고려한 테이블스페이스

테이블스페이스 정보

Altibase는 테이블스페이스를 관리하기 위해서 테이블스페이스의 상태를 점검하거나 모니터링 하기 위한 성능 뷰와 메타 테이블을 제공한다.

SYSTEM_.SYS_TBS_USERS_

또한 다음의 성능 뷰를 통해 사용자들이 사용하는 데이터베이스의 크기, 사용량, 상태 등의 정보를 확인할 수 있다.

V\$TABLESPACES, V\$DATAFILES, V\$MEM_TABLESPACES