

Real Alternative DBMS
ALTIBASE



>>Contents

1. Introduction.....	1
1.1. HyBrid DBMS の必要性.....	2
1.2. ALTIBASE の特徴.....	4
1.3. ALTIBASE のアーキテクチャ.....	8
2. データベースの生成.....	12
2.1. データベースの作成.....	13
3. 起動と終了.....	20
3.1. ALTIBASE の起動.....	21
3.2. ALTIBASE の終了.....	23
4. デイクショナリー.....	24
4.1. メタテーブル.....	25
4.2. メタテーブルの種類.....	26
4.3. パフォーマンスビュー.....	27
4.4. パフォーマンスビューの種類.....	28
5. データベースオブジェクトと権限管理.....	30
5.1. テーブル.....	31
5.2. キューテーブル(Queue Table)管理.....	36
5.3. 制約条件(Constraints) 管理.....	38
5.4. インデックス管理.....	40
5.5. ビュー(View)管理.....	42
5.6. シーケンス(Sequence)管理.....	43
5.7. シノニム(Synonym)管理.....	45
5.8. ストアドプロシージャ、ストアドファンクション管理.....	46
5.9. トリガー管理.....	49
5.10. ユーザー(User)管理.....	51
5.11. 権限(Privilege) 管理.....	53
6. テーブルスペース管理.....	56
6.1. テーブルスペース(Tablespace)概要.....	57
6.2. テーブルスペースの生成.....	61
6.3. テーブルスペース の変更.....	67
6.4. テーブルスペース削除.....	73
6.5. テーブルスペース情報.....	74

7.	トランザクションとロック	76
7.1.	トランザクション(Transaction).....	77
7.2.	ロック(Lock)の概要	79
7.3.	トランザクション永続性(Durability)管理ポリシー	81
8.	バックアップ と復旧	84
8.1.	バックアップ(Backup)	85
8.2.	アルティベース復旧(Recovery)	87
8.3.	バックアップ及び復旧例	91
9.	ALTIBASE の監視	103
9.1.	監視項目	104
9.2.	モニターリングスクリプト(Monitoring Scripts)	105

1.	Introduction
	Hybrid DBMSの必要性
	ALTIBASEの特徴
	ALTIBASEの構造

1.1. Hybrid DBMS の必要性

- DRDBMS (Disk resident Relational Database Management System) の限界

DRDBMSは、データをディスク上に保存し、データを利用するときにバッファに読み込み、アプリケーションプログラムに提供する構造となっています。

また、標準SQLを使用したデータアクセスや、同時性制御とリカバリ機能によるデータ保護をDBMSが行ってくれるため、データを容易に共有できるとともに、アプリケーションプログラムの開発を容易に行えるという長所があります。また、データがディスクに保存されているため、大容量のデータを保存・管理することができます。

このような長所から、DRDBMSは多くの産業分野において使用されてきています。しかし、情報化が急速に進み情報処理の要求性能がアップするにつれ、その低い平均処理速度とディスクIO操作に依存した安定しない性能の問題により、高性能データ処理分野においては、利用できないケースが増えてきています。

- MMDBMS (Main Memory Database Management System) の限界

MMDBMSは、メモリ上に保存されたデータを読み込み、直接アプリケーションプログラムに提供する構造となっています。

リレーショナル構造を持つMMDBMSでは、RDBMSの長所である標準SQLによるデータアクセス、同時性制御、及び、リカバリ機能によるデータ保護など、DRDBMSと共通の長所を持っています。

さらに、ディスクにデータを保存するDRDBMSに比べ、MMDBMSはデータをメモリに保存するため、平均処理速度が速く均一な性能を保障します。従って、DRDBMSでは、実現が困難であった高速性能や均一性能を必要とする分野において、脚光を浴びています。

MMDBMSは、データの管理構造がDRDBMSと比較して単純化されているため高速な更新処理が可能となりますが、MMDBMSもまたDRDBMSと同様にデータ保護のためにログファイルをディスクに記録するため、メモリとディスク装置の物理的な違いのような、数百倍といった性能差はありません。

一般的に、DRDBMSに比べた場合、MMDBMSのほうが、更新処理性能で約10倍、検索処理性能で約3倍以上の性能を示しています。このような高速性能及び均一性能の長所にもかかわらず、メモリの物理的なサイズの限界から、膨大な量の情報処理を要求する分野においては、MMDBMSだけではその限界を克服できていないのが現状です。

- MMDBMS & DRDBMS 混在利用での問題点

このような、DRDBMSとMMDBMSの限界点を克服するために、一般的に利用されている構造は、高性能を必要とするデータはMMDBMSに、大容量が必要なデータはDRDBMSに保存しながら、データを差別化してMMDBMSとDRDBMSを混在利用する方式です。

この方式では、MMDBMSとDRDBMSで共通に利用するデータを互いに同期化しなければならないという問題や、MMDBMSとDRDBMSを同時に処理しなければならないことによるアプリケーションプログラムの複雑化、さらに、異なるアーキテクチャの製品を併用することによる障害復旧の複雑化というような問題を抱えています。

しかしながら、これまでは高性能処理と大容量処理を同時に行える構造を持つDBMSがなかったため、現実的な対応案として活用されているという現状があります。

- Hybrid DBMS の必要性

これまでのDRDBMS、MMDBMS、また、MMDBMSとDRDBMSを混在利用する方式でのそれぞれの長所を受け継ぎ、問題点を解決するために、Hybrid DBMSが登場することになりました。

Hybrid DBMSは、高性能を必要とするデータはメモリに、大容量を必要とするデータはディスクに保存することでデータを差別化して保存しながら、この二つの種類のデータを処理するDBMSは一つに統合されているという特徴を持っています。

高性能情報処理と大容量情報処理を一つのDBMS上で統合して処理する構造であるため、前述のDRDBMSとMMDBMSの混在利用方式でのデータの同期化や、障害対応処理の複雑化、アプリケーションの複雑化などの問題を解決できるようになりました。また、MMDBMS専用、DRDBMS専用、Hybrid DBMSなど、多様な構成も可能となります。

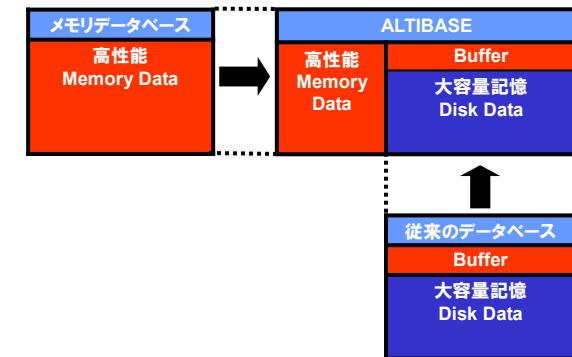


図1-1 ハイブリッドデータベースのアーキテクチャ

1.2. ALTIBASE の特徴

このセクションでは、高性能大容量ハイブリッドデータベースシステムであるALTIBASEが持つ特徴、構造、機能等について簡単に説明します。

● データモデル

ALTIBASEのデータモデルはリレーショナルモデル(**relational model**)を採用しています。リレーショナルモデルは三つの重要な要素を包含します。構造(**structure**)はデータベースに保存またはアクセスされるオブジェクト単位、即ち、テーブル、ビュー、インデックス等を称するもので、これらは、演算子により操作される単位となります。演算(**operation**)はデータベースのデータと構造をユーザーが操作できるように許容する行為(**action**)を定義したもので、無欠性規則を持ちます。無欠性規則(**integrity rule**)は、データと構造に許諾された演算を扱うための規則であり、データと構造を保護します。

● サーバープロセス構造

ALTIBASEサーバは、内部的にも多重スレッド構造を持っています。クライアント-サーバ構造下において、一つのクライアントは一つのサーバスレッドと一緒にセッション(**session**)を構成します。クライアント-サーバ構造は既存のRDBMSが提供する形態で、アプリケーションプログラムの作成が容易であると同時に、データベースの管理が便利であるといった長所があります。

● インターフェース

ALTIBASEは、既存のリアルタイムデータベースシステムとは異なり、汎用性を追求する一環として、業界標準のインターフェースをサポートしています。データベース間い合わせ言語として、SQL92標準に完全に準拠しています。プログラミングインターフェースとしてはODBC、JDBC、C/C++ Precompiler等をサポートしているため、既に作成されているデータベースアプリケーションプログラムの移行コストを最少化することができます。

● 多版型同時実行制御

ALTIBASEは多版型同時実行制御(以下、「MVCC: Multi-Version Concurrency Control」)を利用した同時性制御をサポートしています。MVCCは、一つのデータに対する複数のバージョンを維持し、読み取りと書き込み演算での衝突を無くすことで、最大の性能を発揮できるようにするものです。特に、既存の行レベルロック方式の短所であった、修正されたデータに対する読み取り演算が待機するといった問題や、既に読み取られたデータに対する修正演算が長期間待機するといった問題点を回避し、また、必要のない古いデータは即時回収することで、メモリの浪費

を防止します。MVCCは、多数のユーザーがアクセスする環境で最適な性能を発揮します。

● トランザクション

ALTIBASEはHybrid DBMSの構造に合わせ、最適な性能を発揮できるトランザクション構造と、それに関連する多様な機能を提供します。まず、データベースの中で同時に実行できるトランザクションの数を、プロパティを利用して調節でき、また、効率的なサーバ運用のために、別途サポートするautocommitモードを使用することができます。ALTIBASEシステムが提供するトランザクションの分離レベルは、**read committed**、**repeatable read**、**serializable**があり、ユーザーの要件に合わせ、適切な分離レベルを選択して使用することができます。

● ダブルライトバッファ(Double Write Buffer)

システムのページサイズとファイルシステムの物理的ページサイズが異なる場合、Disk I/Oの実行中に異常終了すると、ページが不完全な状態で残っている可能性があります。このような現象を防止するためにALTIBASEは、ページのディスクへの出力時にディスクの特定領域に存在するダブルライトバッファ領域に同じイメージを予め保存して置きます。ALTIBASEの再起動時にデータページが不完全な状態であった場合は、ダブルライトバッファ領域からページ情報を読み取って不完全なページを復旧します。

● ファジー&ピンポン チェックポイント

ALTIBASEは、最新のデータベース状態を安全にバックアップデータベースに反映するために、ファジー&ピンポン チェックポイントを実行します。メインメモリデータベース上でのファジーチェックポイント(**fuzzy checkpoint**)は、全ての変更されたデータページがバックアップデータベースに出力する際に、オンライントランザクション処理に影響を与えないようにページラッチを取得せずに実行され、チェックポイント処理中のトランザクションログ情報を一緒に記録します。また、チェックポイント処理中の異常終了によりチェックポイントファイルが不完全な状態となる場合に備えて、2世代のチェックポイントファイルを交互に使用して、常に完全な状態のチェックポイントファイルが維持されることを保証します。

● ストアドプロシージャ(Stored Procedure)

ALTIBASEは入力パラメタ、出力パラメタ、入出力パラメタを持ち、BODY内に定義された条件に従って複数のSQL文を一度で実行するストアドプロシージャを提供します。ストアドプロシージャの種類は、リターン値によりプロシージャと関数とに分けられます。

● デッドロック検知

デッドロックは、トランザクション間のリソース割り当てが自立的に解除できない、非正常なトランザクション停止状態です。このような場合、一般的にデッドロックを検知する別のスレッド、またはプロセスを置くことになりませんが、この方式では、一定の間隔でロックリソースの使用状況を確認して検出する構造となるため、一時的なサービス中断を招く恐れがあります。

ALTIBASEでは、ロックリソースの確保時にデッドロック状態となるかどうかを検出し、迅速な対応を取ることできるため、どのような状況においてもサービスを中断することなく、持続的かつ安定的なデータベース運用を保障します。

- テーブル圧縮

データベース運用時に、実際に、特定のメモリテーブルが、必要なメモリスペース以上を占めることがあります。主に、大量のデータがインサートされた後に変更や削除が行われる場合ですが、このような場合、該当テーブルから必要のないメモリをシステムに返還できれば、より効率的にメモリを使用することができます。そのため、ALTIBASEは、メモリテーブルに対しテーブル単位の圧縮機能を提供し、メモリリソースを効率的に使用できるようにします。

- データベース二重化

ALTIBASEは、システムの高い可用性(**high availability**)と無停止(**fault tolerance**)システムを実現するために、ログベースのデータベース二重化機能(**replication**)を提供します。ALTIBASEのレプリケーションでは、トランザクションログ情報を相手サーバーに伝達する方式により、負荷を増大させることなく効率的にデータベースの2重化を実現しています。サービス中のローカルシステムの二重化管理スレッドは、リモートシステムの二重化管理スレッドにリアルタイムで更新情報を伝達します。リモートシステムの二重化管理スレッドは、受信したログデータを分析しデータベースに反映します。こうすることで、サービス中のコンピュータシステムが中断された場合でも、システム復旧時間を必要とすることなく、直ちにサービスを続行できる体制を整えています。

2重化の機能を使用して負荷分散(**load balancing**)構成を実現することもできます。ALTIBASEのデータベース複製運用環境下で、サービスするトランザクションを二つ以上のグループに分け、それぞれのトランザクションが該当サーバーで実行されるようにし、各サーバーで変更されるデータベース内容を相手側サーバーに反映することで、複製されたデータベースの一貫性(**consistency**)を保障します。

- クライアント/サーバプロトコル

ユーザーは、アプリケーションシステムの構成に適したクライアント/サーバプロトコルを選択して使用することができます。ALTIBASEが提供する通信プロトコルにはTCP/IP、IPC、Unix Domain socketがあります。

TCP/IP(Transmission Control Protocol/Internet Protocol)プロトコルは、ネットワーク上でクライアント-サーバ間で使用される標準通信プロトコルです。IPC(Inter Process Communication)プロトコルでは、共有メモリ(**shared memory**)を利用してクライアントとサーバ間の通信を行うようにします。IPC通信は、通信パケットに対するマーシャリング(**marshaling**)を必要とせず、共有メモリを利用するため、他の通信プロトコルより高速に動作します。

- データベーススペース

ALTIBASEは、データベースの全てのデータを格納する、一つ以上のテーブルスペースで構成され、テーブルスペースは大きく、メモリテーブルスペースとディスクテーブルスペースに分けられます。メモリテーブルスペースは、その用途に合わせてSYSTEM用とUSER用に分けられ、ディスクテーブルスペースは、SYSTEM用、USER用、また、DATA、UNDO、TEMPORARYテーブルスペースに分けられます。

1.3. ALTIBASE のアーキテクチャ

ALTIBASEは、ストレージマネージャとクエリープロセッサ構成されたサーバ部分と、アプリケーションプログラムの作成のためのクライアントライブラリ、そして、これらの間の通信モジュールとで構成されています。また、これらの他に、SQL実行やデータローディングを行う複数のユティリティと、ディスクに保存されるバックアップデータベース及びログファイルがあります。

- サーバプロセス構造

ALTIBASEは、単一のプロセス内の複数のシステム/ユーザースレッドを生成して処理を行うシングルプロセス/マルチスレッドで構成されています。

Main Thread

全てのスレッドを生成/終了させ、生成したスレッドを管理します。

Service Daemon Thread

クライアントからの接続要求があった場合、Service Thread Poolで待機状態にあるService Threadと、要求してきたクライアントとを接続します。

Service Thread Pool

サーバが起動されると、定義ファイルで設定された数のサービススレッドを生成し、プール管理します。サービススレッドは、クエリーを処理するスレッド(Thread)です。

Checkpoint Thread

障害復旧時のログ適用処理量を削減するために、定期的、あるいは任意で、現在のデータベースやシステムの状況をデータファイルに記録します。

Session Management Thread

クライアントとサービススレッド間のセッションの状態、即ち、このセッションが切断されているかどうかを監視します。

Garbage Collection Thread

MVCCでは、一つのデータに対し必要のない古いデータが生成されることがあります。Garbage Collection Threadは、このような不要になった古いバージョンのレコードを回収し、再利用できるようにしてメモリの使用効率を最大化します。

Log Flush Thread

Log Flush Threadは、データベース内の全てのトランザクションが生成したログを管理し、ログバッファに出力されたログデータをログディスクに反映する機能を行います。ディスクに反映されたログは、データベースシステムの障害や災害発生時に安全に復旧できるように、リカバリ時に使用されます。

Buffer Flush Thread

バッファプールの全てのメモリが使用中である場合、特定のトランザクションが実行中にディスクI/Oを発生させ、応答性能が不安定になることがあります。Buffer Flush Threadは、定期的にバッファをチェックし、一定量以上の空きバッファを維持するよう、使用されていないページをディスクに出力して使用中バッファを解放します。

Archive Thread

メディアエラーに対する復旧をサポートするために、定期的にオンラインログファイルを、指定された場所(ARCHIVE_DIR)にコピーします。ALTIBASEがアーカイブモードで運用される時のみ動作します。

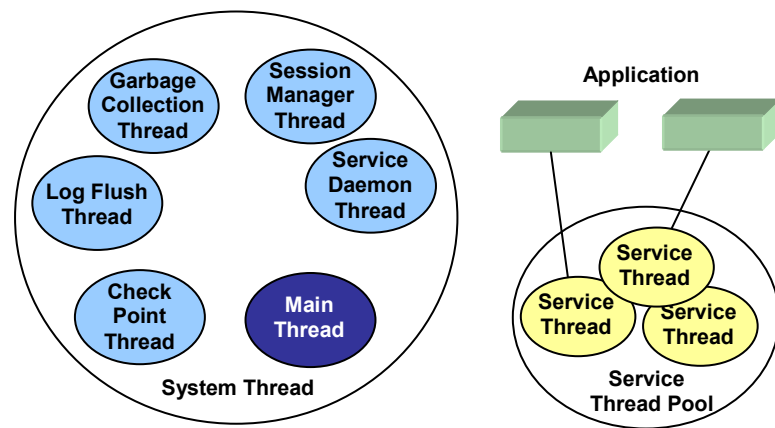


図1-2 ALTIBASEのサーバプロセス構造

- データベースの物理的構造

ALTIBASEデータベースは、ログアンカーファイル、ログファイル、データファイルとで構成されています。

ログアンカーファイル(loganchor file)

ログアンカーファイルは、データバックアップファイルと、ログとの関係を示す重要な情報が格納されています。これは、ログを基準に、ある特定の時点でのデータバックアップファイルが持っている情報の時間的な位置を示します。

ログファイル(logfile)

ログファイルは、トランザクションの回復性と持続性を維持するために使用されます。回復性は、トランザクションのロールバックにより、トランザクションの実行以前の状態へ復帰できるようにし、持続性は、

正常にコミットされたトランザクションが、全てのデータベース障害から本来の内容に復旧できるようにします。

ログファイルは、データベースの更新情報が保存された重要なファイルです。ログファイルは一般的に、データボリュームが損傷した場合に、バックアップしたデータファイルと一緒に、データベースを復旧するために使用されます。

データファイル(datafile)

データファイルのうち、systemファイルにはディスクテーブルを、SYS_TBS_MEMファイルにはメモリテーブルを、tempファイルにはクエリー実行の中間結果が保存されます。また、undoファイルには、MVCCで使用されるUndoレコードが保存されます。

ALTIBASEがメモリ又はディスクにデータを保存するとき、論理的には、テーブルスペースに保存され、物理的には、該当するテーブルスペースに関連付けられているデータファイル内に保存されません。

- データベースの論理的構造

ALTIBASEデータベースを構成するそれぞれのテーブルスペースは、1つ以上のデータファイルで構成されます。ただし、1つのデータファイルは一つのテーブルスペースにのみ関連付けられます。

次の図はテーブルスペースとデータファイルとの関係を説明します。

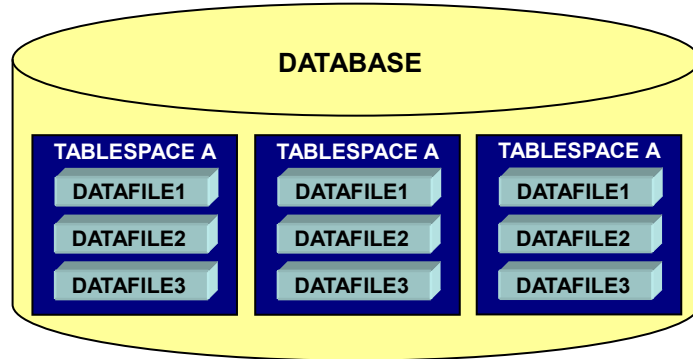


図 1-3 データベースの論理的構造

ALTIBASEはデータベース内の全てのデータを、論理的データベース領域であるテーブルスペースを割り当てます。データベース領域の割当単位には、ページ(page)、エクステント(extent)、及び、セグメント(segment)があります。

ページは最も小さい保存単位であり、ALTIBASEはページ内にデータを保存します。

論理的データベース領域の次の割り当て単位はエクステントです。1つ

のエクステントは、連続的なページ領域の集合であり、さらに一連のエクステントの集合をセグメントとして管理します。

- その他のファイル

ブートログファイル(ALTIBASE_boot.log)

ALTIBASEサーバの稼動状態を記録するログファイルです。このファイルに記録されている情報には、ALTIBASEの起動や終了時に取得されるシステム情報に関する情報があり、また、ALTIBASEの異常終了時には、エラーの発生状態を記録します。

プロパティファイル(ALTIBASE.properties)

ALTIBASEサーバの環境設定のためのファイルであり、ALTIBASEサーバの運用方式やチューニングに関する全ての構成要素を定義します。

メッセージファイル

ストレージマネージャ、クエリープロセッサ、ALTIBASEサーバメインモジュール、及び、関数実行やデータ型関連のエラーメッセージを出力するファイルです。

2. データベースの生成

データベースの生成

2.1. データベースの作成

ALTIBASEを使用するためには、先ず最初にデータベースを作成しなければなりません。

- テーブルスペースの種類

ALTIBASEで定義されるテーブルスペースは次の通りです。

メモリーテーブルスペース(Memory Tablespace)

ディクショナリーテーブルとメモリーテーブル、及び、これに関連するデータベースオブジェクトが保存されるテーブルスペースです。ユーザーが指定するメモリーテーブルスペースを生成することもできます。

データテーブルスペース(Data Tablespace)

ディスクテーブルとインデックスが保存されるテーブルスペースです。これは、さらに、システムデータテーブルスペースとユーザーデータテーブルスペースに区分されます。

Undoテーブルスペース(Undo Tablespace)

ディスクテーブルに存在するレコードに対するMVCC機能をを提供するために、変更前のイメージを一定期間の間保存しておくテーブルスペースです。

揮発性テーブルスペース(Volatile Tablespace)

揮発性テーブルスペースでは、更新情報をトランザクションログに出力しない機能を提供します。このテーブルスペースを使用することにより、回復要件のないデータを高速に処理することができます。

Tempテーブルスペース(Temporary Tablespace)

クエリーを処理する過程で発生する一時テーブルとインデックスを保存するテーブルスペースです。データテーブルスペース同様、システムTempテーブルスペースとユーザーTempテーブルスペースに分けられます。各テーブルスペースを構成するファイルは.dbfの拡張子を持ち、CREATE DATABASE時に\$ALTIBASE_HOME/dbs/ディレクトリに生成されます。

<参考> ALTIBASE運用中に、ユーザーが追加するファイルの拡張子の形式やファイルの保存場所には制限はありません。

- ログギングシステム

データベース内のデータは、いかなる状況下においても永続性を持たなければなりません。ALTIBASEは次の二つのファイルでログギングシステムを構成し、データベースの永続性を提供します。

ログファイル

トランザクション実行中の異常終了に備え、復旧のためのログレコードを記録するファイルです。

ログファイル名称: logfilenn(nnはログファイルのシーケンス番号)

ログアンカー(Log Anchor)ファイル

テーブルスペースについての情報やファイルの位置、チェックポイント関連情報等、サーバ運用に関連する重要なデータが保存されているファイルです。サーバを正常に起動するためにはこのファイルが有効でなければならず、このファイルが無効となっている場合は、サーバを起動することができません。

データベース作成時、ログファイルとログアンカーファイルは \$ALTIBASE_HOME/logs/ディレクトリに生成されます。

ALTIBASEは、3つのログアンカーファイルを作成して使用します。ファイル障害に備えるため、これらの3つのログアンカーファイルは、互いに異なるファイルシステムに置くことを推奨しています。ログアンカーファイルの位置に関連するプロパティはLOGANCHOR_DIRです。

- データベース生成の準備

データベースに接続せず、iSQLを管理者モードで実行します。

```
shell> isql -u sys -p manager -sysdba

shell > isql -sysdba
-----
Altibase Client Query utility.
Release Version 5.1.1.6
Copyright 2000, ALTIBASE Corporation or its subsidiaries.
All Rights Reserved.
-----
ISQL_CONNECTION = TCP, SERVER = 127.0.0.1, PORT_NO = 20300
iSQL(sysdba)>
```

管理者モードでログイン後、CREATE DATABASEコマンドを実行するために、サーバプロセスを起動します。サーバの起動は、次のような順序で行われます。

1. Pre-Processフェーズ
サーバを起動する以前の準備フェーズです。
2. Processフェーズ
create databaseを実行したり、propertyを照会したり変更できるフェーズです。
3. Controlフェーズ
databaseファイルのロード、リカバリを実行するフェーズです。
4. Metaフェーズ
Recoveryが完了し、メタデータの更新、オンラインログのリセットが可能なフェーズです。
5. Serviceフェーズ
ユーザーに対しサービスを提供する最終のフェーズです

データベースの生成はProcessフェーズで実行できます。管理者モードでのログイン直後はPre-Processフェーズのため、次のようなコマンドを実行してProcessフェーズへ移行します。

```
iSQL(sysdba)> startup process
Connecting to the DB server.. Connected.

TRANSITION TO PHASE : PROCESS
To be expired at yyyy-mm-dd.Command execute success.
iSQL(sysdba)>
```

Processフェーズへ移行したら、CREATE DATABASEコマンドを利用してデータベースを生成することができます。

- データベースの生成

Processフェーズでデータベースを生成するためのCREATE DATABASEコマンドは、次のように実行します。
基本オプションでデータベースを生成する例:

```
iSQL> create database mydb initsize=50M noarchive;
DB Info (Page Size = 32768)
(Page Count = 1600)
(File Size = 52428800)
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.
```

- データベースサーバの終了

データベースの生成が完了すると、一旦起動したサーバを終了しなければなりません。サーバの終了は、次の通り実行します。

```
iSQL> shutdown abort;
```

サーバを終了すると、iSQLは再度サーバに接続していないPre-Process状態になり、サーバプロセスも終了します。

shutdownコマンドのオプションとしては、abort以外にもimmediateやnormalがありますが、これらは、サーバがserviceフェーズの場合のみ実行可能です。

- データベース生成関連プロパティ(Property)

CREATE DATABASE構文を実行する時に、指定してない属性はALTIBASEプロパティファイルにより決定されます。関連するプロパティは次の通りです。

プロパティ名	説明	既定値
DB_NAME	生成するデータベースの名称	Mydb
LOG_DIR	ログファイルを保存するディレクトリ	?/logs
MEM_DB_DIR	データベースファイルを保存するディレクトリ	?/dbs
LOGANCHOR_DIR	ログアンカーファイルを保存するディレクトリ	?/logs
TRC_DIR	ALTIBASE 稼働中にサーバーが出力するメッセージを記録するファイルを保存するディレクトリ	?/trc
SHM_DB_KEY	データベースを共有メモリに保存する場合に使用する共有メモリキー	0(使用しない)
MEM_MAX_DB_SIZE	メモリデータベースの最大サイズ	4G
MEM_DB_FILE_SIZE	メモリテーブルスペースを構成する1つのデータファイルの最大サイズ	1G
LOG_FILE_SIZE	ログファイルのサイズ	10M
STARTUP_SHM_CHUNK_SIZE	ALTIBASE 起動時に生成する共有メモリの最大サイズ	1G
PERS_PAGE_CHUNK_COUNT	メモリテーブルスペースの割り当てるページ数	3200
TEMP_PAGE_CHUNK_COUNT	Temp テーブルスペースを追加割り当てする際に1度に割り当てるページ数	128
SYS_DATA_TBS_EXTENT_SIZE	システムテーブルスペースのエクステントサイズ	256K
SYS_DATA_TBS_INIT_SIZE	システムテーブルスペースの初期割り当てサイズ	100M
SYS_DATA_TBS_MAX_SIZE	システムテーブルスペースの最大サイズ	2G
SYS_DATA_TBS_NEXT_SIZE	システムテーブルスペースの追加割り当てサイズ	1M
SYS_TEMP_TBS_EXTENT_SIZE	一時テーブルスペースのエクステントサイズ	256K
SYS_TEMP_TBS_INIT_SIZE	一時テーブルスペースの初期割り当てサイズ	100M

プロパティ名	説明	既定値
SYS_TEMP_TBS_MAX_SIZE	一時テーブルスペースの最大サイズ	2G
SYS_TEMP_TBS_NEXT_SIZE	一時テーブルスペースの追加割り当てサイズ	1M
SYS_UNDO_TBS_EXTENT_SIZE	UNDOテーブルスペースのエクステントサイズ	128K
SYS_UNDO_TBS_INIT_SIZE	UNDOテーブルスペースの初期割り当てサイズ	100M
SYS_UNDO_TBS_MAX_SIZE	UNDOテーブルスペースの最大サイズ	2G
SYS_UNDO_TBS_NEXT_SIZE	UNDOテーブルスペースの追加割り当てサイズ	1M
USER_DATA_TBS_EXTENT_SIZE	ユーザーテーブルスペースのエクステントサイズ	256K
USER_DATA_TBS_INIT_SIZE	ユーザーテーブルスペースの初期割り当てサイズ	100M
USER_DATA_TBS_MAX_SIZE	ユーザーテーブルスペースの最大サイズ	2G
USER_DATA_TBS_NEXT_SIZE	ユーザーテーブルスペースの追加割り当てサイズ	1M
USER_TEMP_TBS_EXTENT_SIZE	ユーザー一時テーブルスペースのエクステントサイズ	256K
USER_TEMP_TBS_INIT_SIZE	ユーザー一時テーブルスペースの初期割り当てサイズ	100M
USER_TEMP_TBS_MAX_SIZE	ユーザー一時テーブルスペースの最大サイズ	2G
USER_TEMP_TBS_NEXT_SIZE	ユーザー一時テーブルスペースの追加割り当てサイズ	1M

ADD_EXTENT_NUM_FROM_TBS_TO_SEG	セグメント拡張時のエクステント数	1
ADD_EXTENT_NUM_FROM_SYSTEM_TO_TBS	テーブルスペース拡張時に追加するエクステント数	4
CHECKSUM_METHOD	ページの整合性チェック方法(0:ヘッダ/フッタバイトをチェック / 1:ページのチェックサム計算)	1

3. ALTIBASE 起動と終了

ALTIBASEの起動

ALTIBASEの終了

3.1. ALTIBASE の起動

ALTIBASEを実行するためには、データベース生成時と同様に、iSQLを-sysdbaオプションで実行します。

```
shell> isql -s 127.0.0.1 -u sys -p manager -sysdba
-----
Altibase Client Query utility.
Release Version 5.1.1.6
Copyright 2000, ALTIBASE Corporation or its subsidiaries.
All Rights Reserved.
-----
ISQL_CONNECTION = TCP, SERVER = 127.0.0.1, PORT_NO = 20300
iSQL(sysdba)>
```

ALTIBASE を起動する時、ステータスは PRE_PROCESS, PROCESS, CONTROL, META, SERVICEの順で移行し、SERVICE状態になって初めて、一般ユーザーからの接続を受け付けることができます。SERVICEステータスへ進行するためには、次のSTARTUP構文を使用します。

```
STARTUP [PROCESS | CONTROL | META | SERVICE];

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]
```

```

Collection: ..... [SUCCESS]
[SM] Rebuilding Indices [Total Count:61]
*****
..... [SUCCESS]
TRANSITION TO PHASE: SERVICE
No IPC Initialize: Disabled
--- STARTUP Process SUCCESS ---
Command execute success.

```

Startupの各フェーズで実行可能な作業は次の通りです。

開始フェーズ	実行可能な作業
PRE-PROCESS	<ul style="list-style-type: none"> PROCESS フェーズに移行できます。
PROCESS	<ul style="list-style-type: none"> CREATE DATABASE を実行できます。 一部のパフォーマンスビューを参照できます。 プロパティ値を変更できます。 CONTROL フェーズへ移行できます。
CONTROL	<ul style="list-style-type: none"> メディアリカバリを実行できます。 META フェーズへ移行できます。
META	<ul style="list-style-type: none"> META データを更新できます。 オンラインログをリセットできます。 SERVICE フェーズへ移行できます。
SERVICE	<ul style="list-style-type: none"> 一般ユーザーからの要求を受け付けることができます。 SHUTDOWN NORMAL/IMMEDIATE/ABORT を実行できます。

3.2. ALTIBASE の終了

現在稼働中のALTIBASEサーバを終了するためにはSHUTDOWN構文を使用します。

```

SHUTDOWN [NORMAL | IMMEDIATE | ABORT];
iSQL> shutdown normal;
Ok..Shutdown Proceeding....
TRANSITION TO PHASE: Shutdown ALTIBASE
. Writing Persistent Indices[Total Count:61] shutdown
normal success.
[ERR-00000: Connected to idle instance]
iSQL> shutdown abort;
ALTIBASE process killed..
[ERR-00000: Connected to idle instance]Collection: .....
[SUCCESS]

```

SHUTDOWN NORMAL と SHUTDOWN IMMEDIATE は、ALTIBASEがSERVICE状態にある時のみ実行可能で、SHUTDOWN ABORTは、どんな状態でも実行可能です。ALTIBASEのSHUTDOWNコマンドは、SYSTEMユーザーでのみ実行することができます。

NORMAL

サーバを正常に終了する方法であり、接続しているクライアントが全て終了されるまでサーバの終了を待機します。サーバ終了時には、クライアント/サーバ間の通信セッションの監視スレッドの終了、サービススレッドの終了、ストレージマネージャの終了処理を順に行い、すべての処理を安全に終了します。

IMMEDIATE

サーバを終了時に、現在接続中のセッションを強制終了させてから、実行中のトランザクションをロールバックしてALTIBASEサーバを終了する方法です。

ABORT

ALTIBASEサーバを、kill -9 システムコマンド (Windows環境の場合はTerminateProcess) を使用して強制的に終了する方法です。この方法で、ALTIBASEサーバを終了した場合、次のサーバ起動時に、データベース復旧プロセスが実行されます。

- 4. データディクショナリー
 - メタテーブル
 - メタテーブルの種類
 - パフォーマンスビュー
 - パフォーマンスビューの種類

4.1. メタテーブル

メタテーブルとは、データベースオブジェクトに関する全ての情報を保存するためのシステム定義テーブルです。このセクションでは、メタテーブルの種類と構造、及び、メタテーブルの照会や変更方法について説明します。

- 構造と機能

メタテーブルはシステムがデータベースオブジェクトを管理するために生成するシステム定義テーブルであり、ユーザーが生成する一般テーブルと同じ構造を持ちます。従って、メタテーブルが使用するデータ型やレコード格納形態は、一般テーブルと同じです。

ALTIBASEは起動時にデータベースオブジェクト情報をローディングし、**DDL**文の実行時に、データベースオブジェクト情報を照会、保存、変更するためにメタテーブルを使用します。

メタテーブルの所有者はシステムユーザー(**SYSTEM_**)であり、一般ユーザーはメタテーブルへアクセスが制限されています。

- 1.

4.2. メタテーブルの種類

メタテーブルの名前は **SYS_** で始まり、以下のテーブルが提供されています。

メタテーブル名称	説明
SYS_COLUMNS_	テーブルの列情報を格納します。
SYS_CONSTRAINTS_	テーブルの制約条件を格納します。
SYS_CONSTRAINT_COLUMNS_	制約条件が設定されている列情報を格納します。
SYS_DATABASE_	データベース情報を格納します。
SYS_GRANT_OBJECT_	オブジェクト権限情報を格納します。
SYS_GRANT_SYSTEM_	システム権限情報を格納します。
SYS_INDEX_COLUMNS_	インデックスの列情報を格納します。
SYS_INDICES_	インデックス情報を格納します。
SYS_PRIVILEGES_	権限情報を格納します。
SYS_PROCEDURES_	ストアードプロシージャ情報を格納します。
SYS_PROC_PARAS_	ストアードプロシージャのパラメタ情報を格納します。
SYS_PROC_PARSE_	ストアードプロシージャの定義情報を格納します。
SYS_PROC_RELATED_	ストアードプロシージャの使用オブジェクト情報を格納します。
SYS_REPLICATIONS_	レプリケーション情報を格納します。
SYS_REPL_HOSTS_	レプリケーションホスト情報を格納します。
SYS_REPL_ITEMS_	複製するテーブル情報を格納します。
SYS_TABLES_	テーブル情報を格納します。
SYS_TBS_USERS_	テーブルスペースのユーザー情報を格納します。
SYS_TRIGGERS_	トリガ情報を格納します。
SYS_TRIGGER_DML_TABLES_	トリガで参照するテーブル情報を格納します。
SYS_TRIGGER_STRINGS_	トリガの定義情報を格納します。
SYS_TRIGGER_UPDATE_COLUMNS_	トリガ更新イベントの列情報を格納します。
SYS_USERS_	ユーザー情報を格納します。
SYS_VIEWS_	ビュー情報を格納します。
SYS_VIEW_PARSE_	ビューの定義情報を格納します。
SYS_VIEW_RELATED_	ビューで使用するオブジェクト情報を格納します。

4.3. パフォーマンスビュー

パフォーマンスビューは、**ALTIBASE**システム内部の情報、即ち、システムメモリ、プロセス状態、セッション、バッファ等のメモリ構造を、ユーザが監視可能なテーブル形式で表示するための提供されるビュー構造です。

パフォーマンスビューは、**ALTIBASE**を使用するユーザーがテーブルに保存されているデータを検索するために**SQL**を使用するのと同様に、**ALTIBASE**運用時に使用されるメモリオブジェクト(例、セッション情報、ログ情報)に関する情報を、**SQL**を利用して検索する機能を提供します。

このセクションでは、**ALTIBASE**がサポートするパフォーマンスビューの構造と機能、照会方法、種類、及び、各ビューにおいて表示される情報について説明します。

● 構造と機能

ALTIBASE内部には、ユーザーが生成したデータベース関連テーブルだけではなく、**DBMS**の運用に必要なプロセス自体の情報を多数保有しています。

パフォーマンスビューは、**ALTIBASE**の運用過程で使用されるほとんどの内部メモリ構造体をビュー形式で提供します。全てのデータはリアルタイムに生成されるため、該当パフォーマンスビューに対する照会を行う時点のプロセス内部の最新情報を取得することができます。

また、パフォーマンスビューは常に読み取り専用の属性を持ちます。パフォーマンスビューに対して更新を行おうとした場合、**ALTIBASE**はエラーを出力し、該当トランザクションをロールバックします。

4.4. パフォーマンスビューの種類

パフォーマンスビューの名前はV\$で始まります。提供されるパフォーマンスビューは次の通りです。

パフォーマンスビュー名	説明
V\$ARCHIVE	アーカイブ/バックアップ関連情報
V\$BUFFPAGEINFO	バッファマネージャの統計情報
V\$BUFFPOOL_STAT	バッファプールの利用統計情報
V\$DATABASE	データベースの情報
V\$DATAFILES	データファイルの情報
V\$DISKGC	ディスクガーベージコレクター情報
V\$DISKTBL_INFO	ディスクテーブル情報
V\$MEMTBL_INFO	メモリーテーブル情報
V\$FLUSHINFO	バッファフラッシュ情報
V\$INDEX	インデックス情報
V\$INSTANCE	インスタンスのStartup情報
V\$LATCH	バッファプールのラッチとIO統計情報
V\$LFG	グループコミット統計情報
V\$LOCK	全テーブルのロック情報
V\$LOCK_WAIT	待ち状態のロック情報
V\$LOG	ログアンカー情報
V\$LOCK_STATEMENT	ロック取得中のステートメント情報
V\$MEMGC	メモリーガーベージコレクター情報
V\$MEMSTAT	メモリーの使用情報
V\$MUTEX	ALTIBASEプロセスのmutex統計情報
V\$PLANTEXT	SQLの実行計画情報
V\$PROCTEXT	スタアドプロシージャのテキスト情報
V\$PROPERTY	ALTIBASEに設定されているプロパティ情報
V\$REPEXEC	レプリケーションマネージャ情報
V\$REPGAP	レプリケーションのギャップに関連する情報
V\$REPRECEIVER	レプリケーションReceiver情報
V\$REPRECEIVER_TRANSTBL	レプリケーションSenderのトランザクションテーブル情報
V\$REPSYNC	SYNC中のテーブルの情報
V\$REPSENDER	レプリケーションSender情報
V\$SEQ	シーケンス関連の情報

パフォーマンスビュー名	説明
V\$SERVICE_THREAD	サービススレッドの関連情報
V\$SESSION	ALTIBASEに生成されたクライアントに対するセッション情報
V\$SESSIONMGR	セッション統計情報
V\$STATEMENT	生成されている全てのセッションのステートメント情報
V\$SQLTEXT	システムで実行されるSQLのテキスト情報
V\$TABLE	パフォーマンスビューの一覧
V\$TABLESPACES	テーブルスペース情報
V\$TRACELOG	トレースロギング情報
V\$TRANSACTION	トランザクションオブジェクト情報
V\$TRANSACTION_MGR	トランザクションマネージャ情報
V\$UNDO_BUFF_STAT	Undo table spaceのバッファプール統計情報
V\$VERSION	データベースバージョン関連情報
V\$CATALOG	ストレージマネージャのカatalog情報
V\$DISK_BTREE_HEADER	ディスクBTREEインデックスヘッダー情報
V\$MEM_BTREE_HEADER	メモリーBTREEインデックスヘッダー情報
V\$MEM_BTREE_NODEPOOL	メモリーBTREEインデックスのためのノードプール情報
V\$MEM_TABLESPACES	メモリーテーブルスペース情報
V\$MEM_TABLESPACE_CHECKPOINT_PATHS	メモリーテーブルスペースのチェックポイントファイル位置情報
V\$MEM_TABLESPACE_STATUS_DESC	メモリーテーブルスペースのステータス一覧
V\$STABLE_MEM_DATAFILES	メモリーデータベースのデータファイル情報
V\$DB_FREEPAGELISTS	メモリーデータベースのFree Page List情報
V\$SEGMENT	ディスクデータベースに割り当てられているセグメント情報
V\$UNDO_TBS	ディスクデータベースに割り当てられているUndo Tablespace情報
V\$DATATYPE	DATABASEのデータ型一覧
V\$SESSION	Session情報
V\$ST_ANGULAR_UNIT	Geometry Dataの角度単位情報
V\$ST_AREA_UNIT	Geometry Dataの面積単位情報
V\$ST_LINEAR_UNIT	Geometry Dataの距離単位情報
V\$SYSSTAT	ALTIBASEのシステム統計情報

5. データベースオブジェクトと権限管理

テーブル
キューテーブル
制約条件
インデックス
ビュー
シーケンス
シノニム
ストアドプロシージャとストアドファンクション
トリガー
ユーザー
権限

5.1. テーブル

テーブルは最も基本的なデータの保存単位で、列で構成されたレコードの集合です。

- メモリテーブルとディスクテーブル

テーブルは、その格納先(テーブルスペース)によりメモリテーブルとディスクテーブルに分けられます。データの格納先となるテーブルスペースはテーブル生成時に決定され、1つのテーブルがメモリテーブルスペースとディスクテーブルスペースの両方に属することはできません。ユーザーは、あらかじめデータのアクセス頻度等を考慮して、多くアクセスが行われるホットデータ(Hot Data)をメモリテーブルに、相対的にアクセス頻度の低いデータをディスクテーブルに分割して生成すると限られたシステムリソースでパフォーマンスを最大化できます。

- システムテーブルとユーザーテーブル

テーブルの種類には、システムが内部的に生成し、管理するシステムテーブルと、一般ユーザーにより生成・管理されるユーザーテーブルがあります。

システムテーブルはデータディクショナリにデータベースオブジェクト情報を保存するメタテーブルと、システムプロセス情報を保存するプロセステーブルに分けられ、プロセス情報はさらに、固定テーブルとパフォーマンスビューとに分けられます。

- メモリテーブル圧縮機能

メモリテーブルが大量データの挿入等により拡張された場合、このスペースをテーブルスペースやOSに返還するための機能を提供します。

ALTIBASEはDBMS再起動時、各メモリテーブルの空いているPAGEをOSに変換するCompaction機能を提供します。

- パーティションドテーブル: Version5 サポート機能

パーティショニングは、大容量ディスクテーブルの様々な性能や管理上の問題を解決するための、ディスクデータベースの一般的なソリューションです。

ALTIBASEはVersion4で、従来のメモリデータベースに統合されたディスクデータベース機能をサポートしましたが、パーティショニング機能がサポートされていなかったため、大容量のテーブル(一般的に、5000万~1億Record以上)の場合、性能や管理上の問題を解決するために、月別、あるいは、日別にテーブルを分割して生成する必要がありました。

そのため、分割して生成した各テーブルに対し、照会時には“UNION ALL”を使用し通じてアクセスし、変更時には各テーブルを明示的に指定しなければならず、運用及びアプリケーション開発が複雑化するという問題がありました。

NOTE：分割する各パーティションの物理的な保存場所はディスクテーブルスペースのみサポートされます。即ち、メモリテーブルのパーティショニングはサポートされません。

- テーブルの生成

テーブル生成時には列定義、制約条件、テーブルが保存されるテーブルスペース、テーブルに挿入できる最大レコード数、データページの充填率などを指定することができます。

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

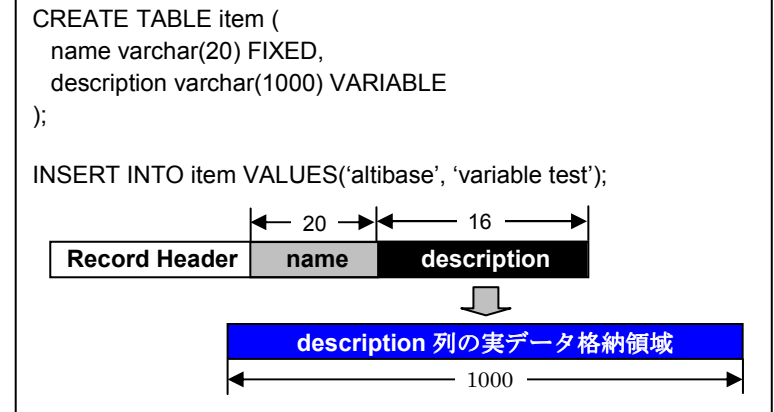
CREATE TABLE dept_c002
AS SELECT * FROM employee
WHERE dno = HSS_BYTESC002;
```

- 列定義時の注意事項

- **VARCHARタイプのFIXED/VARIABLE属性**

VARCHARデータ型の場合、FIXED/VARIABLE属性を指定することができます。この属性を指定しなかった場合、データベースは内部的に特定の長さ(メモリテーブル: 126 Bytes, ディスクテーブル: 14 Bytes)以下の場合、VARCHARであってもFIXEDとして扱います。FIXEDの場合は、データ型がVARCHARであるにもかかわらず、CHARデータ型と同様に格納領域を指定された長さ分確保し、VARIABLEの場合には、データの長さが格納領域として確保されます。VARCHARデータ型のデータ比較方式は、FIXED/VARIABLE属性に関係なくNON-BLANK PADDING比較方式に従います。

FIXED/VARIABLE属性が宣言された列がレコード内で保存される方式を図5-1に示します。



< 図5-1 > VARCHAR列の構造

itemテーブルの列nameはvarchar(20) FIXEDと宣言されているため、実際に挿入される値の'altibase'の長さが8Bytesであっても、レコード内で、20 Bytes分の格納領域が割り当てられます。

itemテーブルの列descriptionはvarchar(1000) VARIABLEと宣言されているため、実際に挿入される値の'variable test'の長さの13 Bytes分の格納領域がレコード内ではなく別領域に割り当てられます。Varchar VARIABLEと宣言された列は、実際のデータが格納されている位置情報(16Bytes)をレコード内に保持します。従って、図5-1の例でdescription列の値を保存するために実際に使用されるスペースは29 Bytes(位置情報16Bytes+データ長13Bytes))となります。

- 大容量データ型(LOB)のサポート：Version5 でサポート

ALTIBASE Version 4までは、全てのデータ型はページサイズを超えることができない制限がありました。そのため、メモリテーブルの場合で32Kbytes、ディスクテーブルの場合で8Kbytesのデータ長の制限があり、大容量が格納可能なデータ型をサポートする他のDBMSからALTIBASEへの移行時に、テーブル設計が変更が必要となるなど移行性に問題がありました。

この問題に対応するため、ALTIBASE5では、大容量データ型のBLOB/CLOBをサポートしています。

- テーブル定義の変更と削除

テーブル定義はALTER TABLE文、RENAME文を使用して変更することができます。

■ テーブル名の変更

```
RENAME book TO book_new;
```

■ 新規列の追加

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

■ 既存の列の削除

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

■ 列のデフォルト値の設定

```
ALTER TABLE book  
ALTER COLUMN (isbn set default 0);
```

■ 列名の変更

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

■ 制約条件の追加

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

■ 制約条件の削除

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

■ メモリテーブルの圧縮

```
ALTER TABLE book COMPACT;
```

■ 最大格納可能レコード数の変更

```
ALTER TABLE department MAXROWS 6;
```

■ インデックスの無効化

```
ALTER TABLE orders ALL INDEX DISABLE;
```

■ テーブルの削除

```
DROP TABLE employee;
```

● 全レコードの削除

テーブルのレコードはDELETE文を使用して削除することができますが、TRUNCATE TABLE文を利用して削除することもできます。DELETE文は、内部的にレコードを1件単位に削除するもので、TRUNCATE TABLE文は、内部的にDROP TABLE文を実行し、同じ定義のテーブルを再生成します。従って、TRUNCATE TABLE文を実行するとテーブルレベルのロックが取得され、その実行が完了した後に、ROLLBACK文を使用して削除されたデータを復旧することはできません。

```
TRUNCATE TABLE book;
```

● テーブルレコードの操作

テーブルのレコードは、DML文を使用して操作することができます。

■ データの挿入 (INSERT)

```
INSERT INTO employee(eno, ename, sex)  
VALUES(21, 'Altibase', 'M');
```

■ データの削除 (DELETE)

```
DELETE FROM orders WHERE eno = 21;
```

■ データの更新 (UPDATE)

```
UPDATE employee SET salary = salary * 1.07;
```

■ データの参照 (SELECT)

```
SELECT ename, salary FROM employee;
```

5.2. キューテーブル(Queue Table)管理

ALTIBASEは、メッセージキュー機能を用い、データベースとユーザープログラムの間のデータ通信をサポートします。この際、使用されるキューテーブルは、データベースオブジェクトの一つで、他のデータベーステーブルと同じようにDDLとDMLで制御することができます。

- キューの生成

CREATE QUEUE文を使用してキューを生成すると、データベースはキュー名に該当するテーブルを生成します。このテーブルを、キューテーブルと呼びます。キューテーブルは、以下のような構造を持っています。

列名	データ型	長さ	既定値	説明
MSGID	BIGINT	8	N/A	メッセージID
CORRID	INTEGER	4	0	メッセージ識別子
MESSAGE	VARCHAR	メッセージ長	N/A	メッセージ
ENQUEUE_TIME	DATE	8	N/A	ENQUEUE日時

ユーザーは、キューテーブルの列名やテーブル名を変更することはできません。MSGID列には自動的にPrimary Keyが生成されます。

MSGIDを一意に管理するため、データベースは内部的にQUEUE名NEXT_MSG_IDという名前のシーケンスを生成します。ユーザーは、SYSTEM_SYS_TABLES_ メタテーブルで、TABLE_TYPEが'W'であるものを検索すると、このシーケンスに対する情報を照会することができます。このシーケンスは、キューテーブルが削除されるまで維持される必要があるためDROP SEQUENCE文で削除することはできません。

キューテーブルは、SYSTEM_SYS_TABLES_ にTABLE_TYPEが'Q'のテーブルとして保存されます。

ユーザーは、必要に応じて、CREATE INDEX 文により、キューテーブルにインデックスを生成することができます。

```
CREATE QUEUE Q1(40) MAXROWS 10000;
```

* メッセージ長が最大40バイトで、レコード数が10,000のキューQ1を生成します。

- 変更及び削除

CREATE QUEUE 文で生成されたキューテーブルは、DROP QUEUE文で削除されるまで、ALTER TABLE文などを利用して構造を変更することはできません。

キューテーブルの削除：

```
DROP QUEUE Q1;
```

キューに登録されている全メッセージの削除：

```
TRUNCATE QUEUE Q1;
```

- キューテーブルのデータ操作

ユーザーは、ENQUEUE/DEQUEUE、DELETE、SELECTなどのDML文を使用してキューテーブルのデータを操作できます。

- キューへのメッセージの登録(ENQUEUE)

```
ENQUEUE INTO Q1(message)
VALUES('This is a message');
```

- キューからのメッセージの取り出し(DEQUEUE)

```
DEQUEUE MESSAGE, CORRID FROM Q1
WHERE CORRID = 'From Seoul';
```

- キューに登録されているメッセージの削除(DELETE)

```
DELETE FROM Q1;
```

- キューに登録されているメッセージの参照(SELECT)

```
SELECT message FROM Q1;
```

5.3. 制約条件(Constraints) 管理

制約条件は、テーブルのデータ挿入または変更時に制約事項を設定し、データの一貫性を維持する役割をします。

● 制約の種類

プライマリキー制約

プライマリキー制約は、ユニークキー制約及び、NOT NULL制約が含まれる制約条件です。1つ以上の列に対してプライマリキー制約を定義することができ、内部的にユニークキーインデックスが生成されます。プライマリキーに含まれている全ての列には、NULL値の挿入ができません。プライマリキー制約は1つのテーブルに1つだけ定義することができます。

ユニークキー制約

1つ以上の列に対して定義できる制約条件で、列または列の組合せに対して重複値の挿入を許可しない制約条件です。ユニークキー制約を定義すると、内部的にユニークキーインデックスが生成されます。

外部キー制約

参照整合性制約で、参照関係にあるテーブル間のデータ一貫性を維持させる制約条件です。

NOT NULL / NULL制約

列にNULL値が挿入できない制約条件で、列の単位で定義できます。NULL制約は、NULL値を許容する制約条件で、列に対してNOT NULL制約を設定しなかった場合は、NULL制約が定義されます。

TIMESTAMP制約

レコードの挿入または更新時のシステム時間値を設定する制約条件です。主に二重化の対象テーブルの列で定義して使用します。

● 制約の生成と削除

制約条件は、テーブル生成(CREATE TABLE文)、またはテーブル変更(ALTER TABLE文)時に定義することができます。

制約条件を定義する時、制約条件名はユーザーにより設定することができます。制約条件名を設定しなかった場合は、システムが自動的に付与します。制約条件の生成にインデックスの生成が必要な場合は、システムにより自動的に名前を付与したインデックスが生成されます。

```
CREATE TABLE inventory(  
  subscriptionid CHAR(10),  
  isbn CHAR(10),  
  storecode CHAR(4),  
  purchasedate DATE NOT NULL,  
  quantity INTEGER,  
  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);
```

制約条件の削除 :

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

5.4. インデックス管理

インデックスは、テーブル内のレコードへ、高速にアクセスのできるようサポートする構造体です。

- インデックスの種別

ALTIBASEは、BTREEとRTREEの2つのインデックスを提供します。RTREEは多次元インデックスで、空間クエリー時に使用されます。

B-treeインデックス

空間データ型(GEOMETRY)、及び、ラージオブジェクトデータ型(BLOB/CLOB)を除いた全てのデータ型にB-Treeのインデックスを生成することができます。B-Treeは、多くのDBMSで使用されているインデックス構造で、様々な派生形式があります。ALTIBASEは、B+Tree形式のインデックスを提供します。

B+Treeは、インデックスの最下位のレベルに存在するLeaf Nodeと、最上位のレベルに存在するRoot Node、またLeafとRootの間に存在するInternal Nodeで構成されます。キー値は、全てLeaf Nodeにのみ存在し、RootとInternal Nodeには左側子ノードと右側子ノードの間の中点の値であるSeparatorキーを持ちます。

R-treeインデックス型

空間データ型であるGEOMETRY列にはR-Treeインデックスを生成することができます。R-Treeは各空間オブジェクトを囲む最小四角形であるMBR(Minimum Bounding Rectangle)を使用して、一次的に条件フィルタリングを実行した後、この結果で残されたオブジェクトに対して正確なインデックス検索条件をチェックするRefinementを実行する方式で、対象オブジェクトを検索します。R-Treeの挿入、削除、ノード分割、ノード統合アルゴリズムはMBRを基準にすること以外は、B-Treeに類似しています。

- ユニークキーインデックス (Unique)

ユニークキーインデックス(Unique Index)

インデックス列に対して重複値を許容しないインデックスです。

重複キーインデックス(Non-unique Index)

インデックス列に対して重複値を許容するインデックスです。ユニークキーオプションを指定しない場合、このインデックスが生成されます。

ユニークキー(Unique Key)とプライマリキー(Primary Key)の相違点

ユニークキーとプライマリキーは、両方とも重複値を許容しない点は同じですが、NULL値を許容するかどうか異なります。プライマリキー

の場合、NULL値を許容しません。

- 複合キーインデックス (Composite Index)

単一キー インデックス(Non-composite Index)

インデックス対象列が1つであるインデックスです。

複合キー インデックス(Composite Index)

いくつかの列の組み合わせに対して、生成するインデックスです。

- インデックスの作成

インデックスは、テーブルの制約条件により、内部的に生成される場合と、CREATE INDEX文を使用して、ユーザーが明示的に生成する場合があります。

列ごとにソート順を指定

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

インデックスのタイプを指定

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

UNIQUEインデックスを作成

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

PERSISTENTインデックスを作成

```
CREATE INDEX TB1_IDX1 ON TB1(C1) SET PERSISTENT=ON;
```

- インデックスの変更と削除

インデックスを有効化するかどうか、インデックスを永続化するかどうかをALTER INDEX文を使用して変更できます。

```
ALTER INDEX EMP_IDX1 SET PERSISTENT = ON;
```

インデックスの削除は、DROP INDEX文を使用します。テーブルの制約条件により内部的に生成されたインデックスは、その制約条件を削除することで自動的に削除されます。

```
DROP INDEX emp_idx1;
```

5.5. ビュー(View)管理

ビューは、1つ以上のテーブル、またはビューを元にした実データを元にした論理的なテーブルです。

- ベーステーブル

ベーステーブルは、ビューがアクセスしてデータを読むとるテーブルであり、1つのビューに複数のベーステーブルが存在する場合があります。

ALTIbaseがサポートするビューは読み取り専用ビューで、変更可能ビュー、またはマテリアライズドビューはサポートされていません。

- ビューの作成

ビューはCREATE VIEW文を使用して作成します。

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

- ビューの変更と削除

既に存在するビューに対してビューの内容を変更する場合にはCREATE OR REPLACE VIEW文を使用します。

```
CREATE OR REPLACE VIEW emp_cus AS
SELECT DISTINCT o.eno, e.ename, c.cname
FROM employee e, customer c, orders o
WHERE e.eno = o.eno AND o.cno = c.cno;
```

ビューは、ベーステーブルを参照する論理的なテーブルのため、ベーステーブルでDDL文を実行してベーステーブルの定義が変更された場合、関連するビューが実行できない状態になることがあります。この場合、以下のようにALTER VIEW文を使用して、ビューが実行できるかどうかを検証することができます。

```
ALTER VIEW avg_sal COMPILE;
```

ビューの削除は、DROP VIEW文を使用します。

```
DROP VIEW avg_sal;
```

5.6. シーケンス(Sequence)管理

ALTIbaseでは、キージェネレータとしてシーケンスが提供されます、シーケンスはメモリを使用して発番されるため、非常に高速に動作します。

- シーケンスの用途

キージェネレータとして用いられるシーケンスは、主にDML文で任意の列に値を設定するために使用されます。シーケンスを使用する構文は、sequence名.nextval、sequence名.currvalがあります。Sequence名.nextvalは、シーケンスの次の値を使用し、sequence名.currvalはシーケンスの現在値を使用します。

シーケンス生成の直後は、sequence名.currvalを使用することはできません。Sequence名.currvalを使用するには、シーケンス生成後にsequence名.nextvalを1回以上実行しなければなりません。

シーケンスのnextvalを使用するたびに、指定された増減値(increment by)分だけシーケンス値が変化します。シーケンスの増減値が指定されていない場合、増減値は1となります。

- シーケンスの利用

シーケンスを使用してキーを生成し、レコードを挿入する例：

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

* sequence 生成時、初期値は1であるため、t1には1が挿入され、sequenceのNextvalは1増加した2になります。

- シーケンスの生成

CREATE SEQUENCE文を使用して、シーケンスを生成する際に使用できるオプションは以下の通りです。

START WITH:

シーケンスの開始値

INCREMENT BY:

シーケンスの増減値

MAXVALUE:

シーケンスの最大値

MINVALUE:

シーケンスの最小値

CYCLE:

シーケンス値が最大値/最小値に達した場合でも、引き続きシーケンス値を生成する場合に指定します。昇順シーケンスが最大値に達した場合には、最小値を生成し、降順シーケンスが最小値に達した場合に

は、最大値を生成します。

CACHE:

アクセスを高速化するためにメモリにキャッシュするシーケンス値の数を指定します。シーケンスキャッシュはシーケンスを最初に操作する時にメモリにキャッシュされ、以降シーケンス値が要求されるたびにキャッシュされたシーケンスから値が返されます。最後にキャッシュされたシーケンス値が使用されると、新しいシーケンス値を生成してメモリにキャッシュします。既定値は20です。

1から始め、1ずつ増加する基本的なシーケンスの生成

```
CREATE SEQUENCE seq1 ;
```

0から100の間で2ずつ増加し、循環するシーケンスの生成

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

- シーケンスの変更と削除

ALTER SEQUENCE文を用い、START WITH値を除いた全てのシーケンスオプションを変更できます。

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

DROP SEQUENCE文を使用してシーケンスを削除できます。

```
DROP SEQUENCE seq1;
```

5.7. シノニム(Synonym)管理

ALTIBASEは、テーブル、シーケンス、ビュー、ストアドプロシージャ及びストアドファンクションに対する別名(alias)を付与できるシノニムを提供します。

- シノニムの利用

以下のような場合に、データベースシノニムを使用します。

- ✓ オブジェクトの生成ユーザーとオブジェクト名を隠蔽したい場合。
- ✓ 使用するSQL文を単純化したい場合。
- ✓ ユーザーの変更によるプログラムの変更を最小化したい場合。

- シノニムの作成と削除

テーブルdeptの別名としてmy_dept シノニムを生成

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

シノニムの削除 :

```
DROP SYNONYM my_dept;
```

5.8. ストアドプロシージャ、ストアドファンクション管理

ストアドプロシージャは、SQL文と流れ制御文、割り当て文、エラー処理ルーチンなどを使用してプログラミングした業務ロジックを、データベースに保存して実行するデータベース オブジェクトです。

● 種類

ストアドプロシージャ

入力パラメタ、出力パラメタ、入出力パラメタを持ち、ボディー内に定義された条件により、複数のSQL文を実行するデータベースプロシージャです。戻り値を持たず、出力パラメタ、入出力パラメタによってクライアントへ値を返します。戻り値を持たないため、他のSQL文のexpression内で使用することはできません。

ストアドファンクション

ストアドプロシージャと同様な機能で、1つの戻り値を持つ関数です。ストアドプロシージャとは異なり、戻り値を持つため、他のSQL文のexpression内でシステム提供関数と同様に使用することができます。

Type Set

ストアドプロシージャのユーザー定義型を定義した集合です。主にストアドプロシージャ間で、パラメタまたは戻り値をユーザー定義型でやり取りする場合に使用します。

● ストアドプロシージャ保存プロシージャ関連文章

分類	ステートメント	説明
作成	CREATE [OR REPLACE] PROCEDURE 文	新しいストアドプロシージャを生成するか、既に生成されているストアドプロシージャの定義を変更します。
	CREATE [OR REPLACE] FUNCTION 文	新しいストアドファンクションを生成するか、既に生成されているストアドファンクションの定義を変更します。
	CREATE [OR REPLACE] TYPESET 文	タイプセットを生成または変更します。
変更	ALTER PROCEDURE 文	ストアドプロシージャ生成後、関連するオブジェクトの定義が変更された場合に、ストアドプロシージャを再コンパイルします。
	ALTER FUNCTION 文	ストアドファンクション生成後、関連するオブジェクトの定義が変更された場合に、ストアドファンクションを再コンパイルします。

分類	ステートメント	説明
削除	DROP PROCEDURE 文	生成されたストアドプロシージャを削除します。
	DROP FUNCTION 文	生成されたストアドファンクションを削除します。
	DROP TYPESET 文	生成されたタイプセットを削除します。
実行	EXECUTE 文	ストアドプロシージャまたはストアドファンクションを実行します。
	FUNCTION	SQL文内で built-in function と同じように使用できます。

● ストアドプロシージャの生成

```
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;
/
```

- ストアドプロシージャの変更と削除

既存のストアドプロシージャの名前は内容(パラメータ、または本体)を変更する場合は、CREATE OR REPLACE PROCEDURE文を使用し、ストアドプロシージャを再作成します。

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

ストアドプロシージャで使用しているテーブル、シーケンス、および、ストアドプロシージャが呼び出す他のストアドプロシージャまたは、ストアドファンクションが生成時に定義された状態と異なる場合、そのストアドプロシージャは実行できない無効な状態となります。

ALTER PROCEDURE文は無効なストアドプロシージャを再コンパイルして、有効な状態に変更します。

```
ALTER PROCEDURE proc1 COMPILE;
```

ストアドプロシージャを削除する場合、DROP PROCEDURE文を使用します。

```
DROP PROCEDURE proc1;
```

5.9. トリガー管理

トリガーとは、テーブルにデータが挿入、削除、または更新される時、システムにより起動・実行されるストアドプロシージャです。

- トリガー構成要素

- トリガーイベント(trigger event or statement)**

- トリガー実行を発生させるSQL文をトリガーイベントと言います。

- トリガー条件(trigger restriction)**

- トリガーイベントが発生したときに、トリガーを実行する条件で論理演算式を包む式です。

- トリガーアクション(trigger action)**

- トリガー条件がTRUEである場合に、トリガーが実行するストアドプロシージャです。

- トリガー イベント

- DELETE**

- 該当テーブルのデータを削除するDELETE文の実行時、トリガーを起動します。

- INSERT**

- 該当テーブルのデータを挿入するINSERT文の実行時、トリガーを起動します。

- UPDATE**

- 該当テーブルのデータを変更するUPDATE文の実行時、トリガーを起動します。UPDATEトリガーイベントにOF構文を使用する場合OF構文で指定された列が変更される場合のみトリガーを起動作します。

- ただし、データベースの無欠性のためにレプリケーションにより適用されるテーブルの変更は、トリガーイベントで処理できません。

- トリガーの生成

```
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;
/
```

- トリガーの変更と削除

生成したトリガーの起動を中止する場合や、無効な状態のトリガーを再コンパイルする場合、ALTER TRIGGER文を使用します。

```
ALTER TRIGGER del_trigger DISABLE;
```

トリガーを削除する場合DROP TRIGGER文を使用します。

```
DROP TRIGGER del_trigger;
```

5.10. ユーザー(User)管理

データベース生成直後の初期データベース内には、システム管理者であるSYSTEM_とSYSユーザーのみが存在します。このユーザーは、DBA(DB管理者)であるため、ユーザーを生成してスキーマオブジェクトを管理します。

- SYSTEM_とSYS ユーザー

データベースを生成すると、システムにより生成されるシステム管理者ユーザーであり、一般ユーザーと区別されます。

SYSTEM_ ユーザーは、メタテーブルの所有者でメタテーブルに対するDDL文とDML文の実行権限を持っています。

SYSユーザーはDBAとしてユーザーテーブルに対しての全ての権限を持ち、システムの全ての作業を実行できる権限を持っています。

また、このユーザーは削除することはできません。

- ユーザーの生成

CREATE USER文を使用してユーザーを生成します。ユーザー生成時にパスワードを指定します。また、そのユーザーが既定で使用するテーブルスペースを設定できます。

```
CREATE USER DLR IDENTIFIED BY DLR123  
DEFAULT TABLESPACE user_data  
TEMPORARY TABLESPACE temp_data  
ACCESS sys_tbs_memory ON;
```

- ユーザーの変更と削除

ALTER USER文を使用してユーザーのパスワードと該当ユーザーに関連付けられているテーブルスペースの設定を変更できます。

ユーザー暗証番号の変更

```
ALTER USER DLR IDENTIFIED BY DLR12345;
```

既定のデータテーブルスペースの変更

```
ALTER USER DLR DEFAULT TABLESPACE dlr1_data;
```

一時テーブルスペースの変更

```
ALTER USER DLR TEMPORARY TABLESPACE dlr1_tmp;
```

テーブルスペースアクセスへのアクセス許可設定の変更

```
ALTER USER DLR ACCESS dlr2_data ON;
```

ユーザー削除

```
DROP USER DLR CASCADE;
```

ユーザーを削除する場合、**DROP USER**文を使用します。該当ユーザーが所有している全てのオブジェクトを一括してまで一挙で削除したい場合、**cascade**オプションを使用します。**Cascade**オプションを使用しない場合、該当ユーザーのスキーマ内にオブジェクトが存在すると、**DROP USER**文の実行時にエラーが発生します。

5.11. 権限(Privilege) 管理

ユーザーがデータベースオブジェクト、またはデータにアクセスするためには適切な権限を設定する必要があります。

● 権限の種類

ALTIBASEは、ユーザーが権限(privilege)を管理できる機能はサポートしますが、権限のまとまりであるロール(role)はサポートしていません。

システムアクセス権限 (System Privilege)

システムアクセス権限は一般的にDBAが管理し、データベースの特定の作業を実行する権限や、全てのスキーマにあるオブジェクトを管理できる権限です。

ALTIBASEが提供する全体のシステムアクセス権限リスト

System privilege	Statement
DATABASE	ALTER SYSTEM
INDEXES	CREATE ANY INDEX
	ALTER ANY INDEX
	DROP ANY INDEX
PROCEDURES	CREATE PROCEDURE
	CREATE ANY PROCEDURE
	ALTER ANY PROCEDURE
	DROP ANY PROCEDURE
	EXECUTE ANY PROCEDURE
SEQUENCES	CREATE SEQUENCE
	CREATE ANY SEQUENCE
	ALTER ANY SEQUENCE
	DROP ANY SEQUENCE
	DROP ANY SEQUENCE
SESSIONS	CREATE SESSION
TABLES	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
	TABLESPACES
ALTER TABLESPACE	
DROP TABLESPACE	

System privilege	Statement
USERS	CREATE USER
	ALTER USER
	DROP USER
VIEWS	CREATE VIEW
	CREATE ANY VIEW
	DROP ANY VIEW
MISCELLANEOUS	GRANT ANY PRIVILEGES
TRIGGER	CREATE TRIGGER
	CREATE ANY TRIGGER
	ALTER ANY TRIGGER
	DROP ANY TRIGGER

オブジェクトアクセス権限 (Object Privilege)

オブジェクトアクセス権限はオブジェクトの所有者が管理し、オブジェクトにアクセスして、操作できる権限です。

ALTIBASEがサポートするオブジェクトアクセス権限リスト

Object privilege	Table	Sequence	PSM	View
ALTER	○	○		
DELETE	○			
EXECUTE			○	
INDEX	○			
INSERT	○			
REFERENCES	○			
SELECT	○	○		○
UPDATE	○			

- 権限の付与と取り消し

ユーザーには、任意の権限を付与できます。一般ユーザーの場合、CREATE USER文を実行してユーザーが生成されると、次の権限がシステムにより基本的に自動付与されます。

- ✓ CREATE SESSION
- ✓ CREATE TABLE
- ✓ CREATE SEQUENCE
- ✓ CREATE PROCEDURE
- ✓ CREATE VIEW

システムアクセス権限付与

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

オブジェクトアクセス権限付与

```
GRANT SELECT, DELETE ON sys.employee TO uare8;
```

システムアクセス権限解除

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

オブジェクトアクセス権限解除

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

- 6. テーブルスペース管理
 - テーブルスペース概要
 - テーブルスペースの生成
 - テーブルスペースの変更
 - テーブルスペースの削除
 - テーブルスペース情報の取得

6.1. テーブルスペース(Tablespace)概要

ALTIBASEデータベースの論理構造は、テーブルスペース、セグメント、エクステンツ、ページから構成されます。

テーブルスペースは、論理構造の最も大きい単位で、データベースは1つ以上のテーブルスペースで構成されます。

- テーブルスペースの種類

ALTIBASEがサポートするTABLESPACEの種類

ID	テーブルスペース 種別	保 存 先	テーブ ル ス ペ ース 名	生成される タイミング	説明
0	MEMORY TABLESPACE	M	SYS_TBS_ MEMORY	CREATE DATABASE	メモリを保存領域とするテーブルスペース。全てのデータベースオブジェクト(テーブル、インデックス、ビュー、シーケンス、ストアドプロシージャなど)が保存できます。また、メタテーブルも保存されます。
1	SYSTEM DATA TABLESPACE	D	SYS_TBS_ DATA	CREATE DATABASE	データベース作成時に生成されるテーブルスペース。データベースオブジェクトのうち、テーブルとインデックスのみ保存できます。
2	SYSTEM UNDO TABLESPACE	D	SYS_TBS_ UNDO	CREATE DATABASE	undo情報を保存するために使用される特別なテーブルスペース。ユーザーはundoテーブルスペース内にテーブルまたはインデックスなどを生成することはできません。データベース内に一つだけ存在します。ユーザーが明示的に作成したり、削除したりすることはできません。

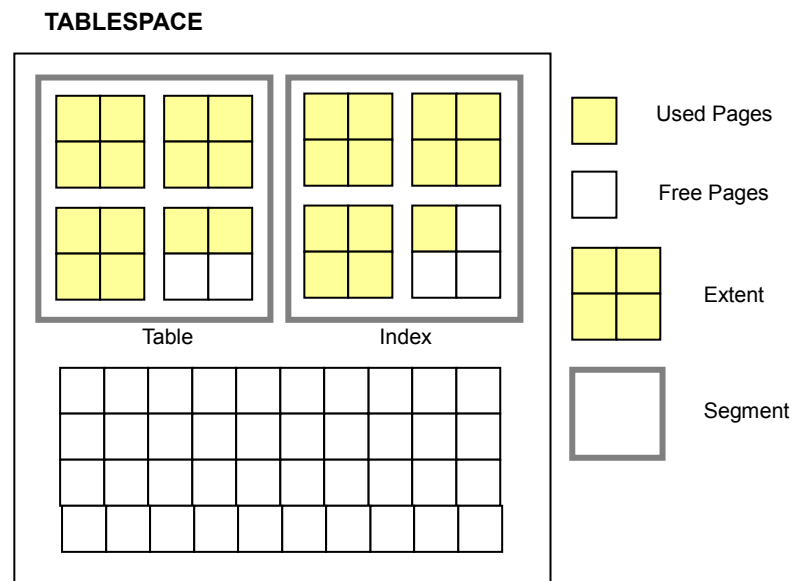
※ M: Memory / D: Disk

I D	テーブルスペース 種別	保存 先	テーブル スペース名	生成される タイミング	説明
3	SYSTEM TEMPORARY TABLESPACE	D	SYS_TBS_ TEMP	CREATE DATABASE	クエリ処理実行時に作業領域として一時的に使用される使用されるテーブルスペース。全てのユーザーのDISKオブジェクトのためのDEFAULT TEMP TABLESPACEとして使用されます。データベースオブジェクトのうち、テーブルとインデックスのみ保存できます。
4 以上	VOLATILE DATA TABLESPACE	M	ユーザー 指定	CREATE TBLESPACE	チェックポイントイメージ ファイルを持たず、ログ情報を出力しないメモリテーブルスペース。ALTIBASE終了により、このテーブルスペースのデータはすべて消失します。
4 以上	USER DATA TABLESPACE	D	ユーザー 指定	CREATE TABLESPACE	ユーザーのオブジェクトを保存するためのディスクテーブルスペース。データベース オブジェクトのうち、テーブルとインデックスのみ保存できます。
4 以上	USER DATA TABLESPACE	M	ユーザー 指定	CREATE TABLESPACE	ユーザーのオブジェクトを保存できるメモリテーブルスペース。データ オブジェクトのうち、テーブルとインデックスのみ保存できます。
4 以上	USER TEMPORARY TABLESPACE	D	ユーザー 指定	CREATE TEMPORARY TABLESPACE	ユーザーの一時テーブルスペースであり、各ユーザーごとに1つだけ指定できます。ユーザーが実行したクエリ処理中に発生する中間結果を処理する時、指定されたtemporary tablespaceが使用されます。

※ M: Memory / D: Disk

● テーブルスペースの論理的構造

テーブルスペースはデータベースの論理構造で最も上位の単位であり、以下のような下位の構造が含まれます。ページは、テーブルスペースを構成する最小の単位で、レコードが保存される最小の領域です。ALTIBASEでは、テーブルスペースの他に、ページを管理するエクステント(Extent)、テーブルスペースにオブジェクトを割り当てる単位となるセグメントという概念を使用します。



< 図6-1 > テーブルスペースの論理的構造

セグメント(Segment)

テーブルスペース内で、データベースオブジェクト(テーブルまたはインデックス)を割り当てる単位で、1つのテーブルまたはインデックスは論理的に1つのセグメントと言えます。

<表6-1> セグメントの種類

セグメント	説明
Tableセグメント	データベースの中にデータを保存する基本的な単位で、テーブルの全てのデータは一つのテーブルスペースに属します。
Indexセグメント	一つの特定のインデックスのレコードは一つのセグメント内に保存されます。インデックスは、キー

	を元にデータを検索するために使用します。
Undoセグメント	データを変更したトランザクションにより使用され、テーブルまたは インデックスを変更する前に、変更前の値をUndoセグメントに保存して 変更のキャンセルができるようにします。
TSS セグメント	ALTIBASEで内部的に管理される TSS(Transaction Status Slot)を管理するためのセグメントで、System Undo Tablespace内に割り当てられます。

* ページはOSが管理するページとは異なる概念で、物理的に32KBのサイズを持ちます。

エクステント(Extent)

データオブジェクトを保存するために必要なリソース、つまり、ページの拡張を割り当てる単位で、データ保存時に保存可能なFree Pageが不足した場合、テーブルスペースからエクステント単位で追加ページを割り当てます。

セグメントごとに、Free ExtentリストとFull Extentリストを管理し、Free Extentが不足した場合に、テーブルスペースにエクステントの追加を要求します。

1つのエクステントはデフォルトで8個のページ(256KB)で構成されます。

ページ(page)

テーブルとインデックスのレコードが保存される最小単位で、I/Oの最小単位です。

ページはページの基本情報とfree slot情報などを管理するためのヘッダを持ち、ヘッダを除いた残りのデータの領域にレコードが保存されます。

Physical header (type, extent RID, page ID, free size, free offset)
Logical header(free slot list)
record record

< 図 6-2 > ページ構造

6.2. テーブルスペースの生成

テーブルスペースの生成及び領域管理方法について説明します。

● テーブルスペース生成構文

```
CREATE TABLESPACE [①テーブルスペース名]
  DATAFILE [②データ ファイル句]
  AUTOEXTEND [③自動拡張句]
  MAXSIZE [④最大サイズ句] ] ]
  EXTENTSIZE [⑤エスメントサイズ句]
  [⑥ONLINE|OFFLINE]
```

① テーブルスペース名

テーブルスペース名を指定します。

② データファイル句

```
DATAFILE データファイル名 SIZE integer [K|M|G] [REUSE]
[自動拡張句]
```

データ ファイル名は絶対パス名を指定しなければなりません。この構文のSIZEパラメタ以下の構文は省略することができます。絶対パス名を指定せず、ファイル名のみを指定した場合、ALTIBASEPropertyに定義されているDefaultパス('\$ALTIBASE_HOME/dbs/')にデータファイルが生成されます。

[REUSE] が定義された場合には、既存のデータファイルを使用します。ただし、データファイル名で指定したパスにファイルが存在していなければなりません。

③ 自動拡張句

```
AUTOEXTEND [ON NEXT integer [K|M|G]
[最大サイズ句] | OFF]
```

AUTOEXTEND ONの場合

データファイルはNEXTパラメタで指定されたサイズ分だけ、[最大サイズ句]で指定されたサイズまで自動拡張されます。

AUTOEXTEND OFFの場合

データファイルは、最初に生成されたサイズから拡張されません。保存領域が不足した場合は、次のデータ ファイルに使用されます。使用可能なデータ ファイルが存在しない場合、トランザクションがキャンセルされます。

④ 最大サイズ句: 自動拡張句のオプション

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

自動拡張の属性が指定されたデータファイルが拡張できる最大サイズを指定します。UNLIMITEDが指定された場合にはデータファイルが指定された記憶保存装置が一杯になるまでサイズが拡張されます。

⑤ エクステントサイズ句

```
EXTENTSIZE integer [K|M|G].
```

テーブルスペースに保存されるセグメント(テーブル またはインデックス)を割り当てる単位であるエクステントサイズを定義します。エクステントサイズを指定しない場合、デフォルト値は256K(8pages)になります。

⑥ ONLINE/OFFLINE句

ONLINEを指定した場合、テーブルスペースが使用可能となります。OFFLINEが指定された場合は、ALTER TABLESPACE を使用してONLINEにステータスを変更してから使用します。

● テーブルスペースの生成例

デフォルト属性のデータ ファイルを1つ持つテーブルスペースの生成

```
iSQL> CREATE TABLESPACE user_data01
      DATAFILE 'user1/dbs/user01.dbf SIZE' 10M;
Create success.
```

user_data01 テーブルスペースは、10Mのデータファイル(/data01/dbs/user01.dbf)を1つ持ち、自動拡張されません。

自動拡張属性を持つエクステントサイズが再定義されたテーブルスペースの生成

```
iSQL> CREATE TABLESPACE user_data02
      DATAFILE 'user1/dbs/user01.dbf' SIZE 100M,
      'user1/dbs/user02.dbf' SIZE 100M
      AUTOEXTEND ON NEXT 1M MAXSIZE 1G
      EXTENTSIZE 512K;
Create success.
```

user_data02テーブルスペースは、2つのデータ ファイルを持ちます。/user1/dbs/user02.dbf ファイルは、自動拡張属性を持ちます(1M byte単位で拡張、最大サイズは1G byte)。したがって、user_data02 テーブルスペースは、2つのデータ ファイル合計で1.1Gまでのデータを保存できます。

また、EXTENTサイズは512Kで定義され、セグメントにPageが割り当てら

れる時、512K(16Pages)ずつ割り当てられます。

● テーブルスペースのサイズ見積もり

ディスク テーブルのサイズ 計算

レコードサイズの計算式

```
[ 32(header) + (columnの長さの和) + 16*可変長列数 ] * record
count
```

レコードヘッダ長(32バイト)にデータ長(Columnの長さの和)と可変長列ヘッダ長(16バイト×可変長列数)を加算します。

インデックスサイズ

```
[ 16(header) + (key column の長さの和) ] * record count
```

この計算式は、leaf node (B*Treeで最下位のノード)の概算サイズです。これ以外にinternal node (leaf nodeを除いた上位ノード)のサイズはkey列のサイズが小さい場合は、無視できるほどサイズが小さいが、key列のサイズが10K以上に大きい場合は、B*TreeのDepthが深くなり、全体

leaf Nodeサイズの50%ほどになることもあるため、該当される場合にはサイズ計算に包含する必要があります。

テーブルサイズの計算例

以下に100万件のレコードを持つテーブルのレコードのサイズとインデックスのサイズの計算例を示します。

```
CREATE TABLE TEST001 (
  C1 char(8) primary key,
  N1 double unique,
  C2 char(128),
  N2 integer,
  IN_DATE date) tablespace user_data02;
```

1. レコードサイズ

```
10(C1列)+ 130(C2列) + 8(N1列) + 4(N2列) + 8(IN_DATE列)
= 160 Bytes
[ 32(ヘッダ長) + 160(列の長さの和) ] * 100万件 = 183.11M
```

2. インデックスサイズ

```
[ 16(ヘッダ長) + 18(key列の長さの和) ] * 100万件 = 32.42M
```

3. TEST001 テーブル 全体サイズ

```
183.11(レコードサイズ) + 30.52(インデックスサイズ)
= 215.53M
```

- 一時(Temporary)テーブルスペース
 - 一時テーブルスペースは、クエリー処理過程中に各種の演算により発生する中間結果を保存するためのテーブルスペースで、データベース生成時に作られます。一時テーブルスペースにはシステムに1つだけ存在するSYSTEM TEMPORARY TABLESPACE(SYS_TBS_TEMP)と、ユーザー毎に割り当て可能なUSER TEMPORARY TABLESPACEの2種類があります。
- 一時テーブルスペースの生成構文
 - 一時テーブルスペースは、バックアップや復旧の対象ではなく、クエリー処理で使用する一時的なテーブルスペースです。”
 - ONLINE/OFFLINE句以外は、User Dataテーブルスペース生成構文と同じ構文を使用して割り当てることができます。

```
CREATE TEMPORARY TABLESPACE
  [①テーブルスペース名]
  DATAFILE [②データ ファイル句]
  AUTOEXTEND [③自動拡張句]
  MAXSIZE [④最大サイズ句] ]
  EXTENTSIZE [⑤エスメントサイズ句]
```

① テーブルスペース名

テーブルスペース名を設定します。

② データ ファイル句

```
DATAFILE データファイル名 SIZE integer [K|M|G] [REUSE]
[自動拡張句]
```

③ 自動拡張句: データファイル句のオプション

```
AUTOEXTEND [ON NEXT integer [K|M|G]
[最大サイズ句] | OFF]
```

④ エクステントサイズ句: エクステント割り当て単位を定義

```
EXTENTSIZE integer [K|M|G].
```

- 一時テーブルスペースの生成例

```
iSQL> CREATE TEMPORARY TABLESPACE user_temp01
TEMPFILE '/user1/dbs/usrtmp01.dbf' SIZE 10M;
Create success.
```

- Undo テーブルスペース

Undoテーブルスペースはデータベース生成時に作られるSYSTEM UNDO TABLESPACEが1つだけ存在し、ユーザーが追加で生成したり削除することはできません。

Undoテーブルスペースの拡張

業務システムで変更トランザクション(特に、Long-Termトランザクション: 実行時間の長いトランザクション)が多く発生する場合に、Undoテーブルスペース不足が発生することがあります。この場合、UndoテーブルスペースのALTER TABLESPACE構文を使用して、データファイルを追加するかファイルサイズを拡張する必要があります。

ディスクGC(Garbage Collector) Thread関連Propertyの調整

Commitが完了したUndoスペースをfree spaceに返還する作業をGarbage Collectingと言います。

短時間に多くのトランザクションが発生する場合、ディスクGCThreadによる使用済みのUndoスペースの回収処理が遅延し、テーブルスペース不足が発生することがあります。この場合、ALTIBASE ディスクGC Threadの実行間隔と実行時に処理するページの個数などの関連するPropertyを修正してディスクGCの性能を調整します。

6.3. テーブルスペース の変更

- テーブルスペースサイズの変更

テーブルスペースのサイズを変更する方法には、以下の4つの方法があります。

1. ALTER TABLESPACE ADD [DATAFILE|TEMPFILE] コマンドを使用してデータファイルを追加します。

【コマンド構文】

```
ALTER TABLESPACE [テーブルスペース名]
ADD DATAFILE|TEMPFILE [データ ファイル句
AUTOEXTEND [自動拡張句
MAXSIZE [最大サイズ句]]]
```

このコマンドは、テーブルスペースのONLINE/OFFLINEステータスに関わらず実行できます。

user_data02 テーブルスペースに初期サイズが100Mバイトで、500Mバイトまで拡張できるデータファイルを追加する例を以下に示します。

```
iSQL> ALTER TABLESPACE user_data02
ADD DATAFILE /user1/dbs/user03.dbf SIZE 100M
AUTOEXTEND ON NEXT 1M MAXSIZE 500M;
Alter success.
```

2. ALTER TABLESPACE DROP [DATAFILE|TEMPFILE] コマンドを使用して、データファイルを削除します。

【コマンド構文】

```
ALTER TABLESPACE [テーブルスペース名]
DROP DATAFILE|TEMPFILE データファイル名
```

このコマンドは、テーブルスペースのONLINE/OFFLINEステータスに関わらず実行できますが、未使用のデータファイルのみが対象となります。

user_data02 テーブルスペースから /user1/dbs/user03.dbf データファイルを削除します。

```
iSQL> ALTER TABLESPACE user_data02
DROP DATAFILE /user1/dbs/user03.dbf;
Alter success.
```

3. ALTER TABLESPACE ALTER [DATAFILE|TEMPFILE] AUTOEXTEND コマンドを使用して、データファイルの自動拡張モードを変更します。

【コマンド構文】

```
ALTER TABLESPACE [テーブルスペース名]
ALTER DATAFILE|TEMPFILE データ ファイル名
AUTOEXTEND [ON [自動拡張節] | OFF]
```

このコマンドは、テーブルスペースのONLINE/OFFLINEステータスに関わらず実行できます。

user_data02 テーブルスペースで/user1/dbs/user01.dbf データファイルの自動拡張モードをONに変更し、最大1GBまで拡張する例:

```
iSQL> ALTER TABLESPACE user_data02
ALTER DATAFILE /user1/dbs/user01.dbf
AUTOEXTEND ON MAXSIZE 1G;
Alter success.
```

user_data02 テーブルスペースで /user1/dbs/user02.dbf データファイルの自動拡張モードをOFFに変更する例:

```
iSQL> ALTER TABLESPACE user_data02
ALTER DATAFILE /user1/dbs/user02.dbf
AUTOEXTEND OFF;
Alter success.
```

4. ALTER TABLESPACE ALTER [DATAFILE|TEMPFILE] SIZEコマンドでを使用して、データファイルのサイズを変更します。

【コマンド構文】

```
ALTER TABLESPACE [テーブルスペース名]
ALTER DATAFILE|TEMPFILE データファイル名
SIZE integer [K|M|G]
```

このコマンドは、テーブルスペースのONLINE/OFFLINEステータスに関わらず実行できます。

user_data02 テーブルスペースで/user1/dbs/user01.dbf データファイルのサイズを200MBに拡張する例 :

```
iSQL> ALTER TABLESPACE user_data02
ALTER DATAFILE /user1/dbs/user01.dbf
SIZE 200M;
Alter success.
```

- テーブルスペースの属性の変更

- テーブルスペースとデータファイルの ONLINE/OFFLINE

テーブルスペース名を変更するか、テーブルスペースのオフラインバックアップまたは復旧を行う際には、テーブルスペースをOFFLINE状態に変更しなければなりません。

【コマンド構文】

```
-- テーブルスペース ONLINE/OFFLINE
ALTER TABLESPACE [テーブルスペース名] [ OFFLINE |
OFFLINE ]
-- データ ファイル ONLINE/OFFLINE
ALTER TABLESPACE [テーブルスペース名]
ALTER DATAFILE データファイル名[ OFFLINE | OFFLINE ]
```

テーブルスペースとデータファイルのOFFLINEはStartupのCONTROLフェーズで実行できます。

テーブルスペースをOFFLINE状態にする手順は以下の通りです。

1. ALTIBASEをShutdownします。
2. sysdbaモードをisqlに接続します。
3. startup processコマンドでPROCESSフェーズまで起動します。
4. alter database [db_name] control 構文でCONTROLフェーズに移行します。
5. ALTER TABLESPACE ~ OFFLINEコマンドでテーブルスペース、またはデータファイルをOFFLINEにします。

```
$ is -sysdba
iSQL> startup process
...
Command execute success.
iSQL> alter database mydb control;
...
Command execute success.
iSQL> ALTER TABLESPACE user_data02
ALTER DATAFILE /user1/dbs/user02.dbf OFFLINE;
Alter success.
iSQL> ALTER TABLESPACE user_data02 OFFLINE;
Alter success.
```

■ DATAFILE の位置の変更(名前変更)

テーブルスペースのデータファイルの位置を変更する場合は、ALTER TABLESPACEコマンドを使用してファイル名を変更します。

【コマンド構文】

```
ALTER TABLESPACE [テーブルスペース名]
RENAME [ DATAFILE | TEMPFILE ]
既存データファイル名 TO 変更データファイル名
```

データファイル名の変更は、テーブルスペースがOFFLINE状態の場合にだけ実行できます。

データファイルを移動する手順は以下の通りです。

1. Startup Controlフェーズで、ALTER TABLESPACE ... OFFLINEコマンドを実行しテーブルスペースをOFFLINEにします。
2. ALTER TABLESPACE ... RENAME DATAFILEコマンドでデータファイル名を変更します。
3. ONLINEさせます。

```
iSQL> startup process
...
Command execute success.
iSQL> alter database mypdb control;
...
Command execute success.
iSQL> ALTER TABLESPACE user_data02 OFFLINE;
Alter success.
iSQL> ALTER TABLESPACE user_data02
RENAME DATAFILE /user1/dbs/user02.dbf
TO /data01/dbs/user02.dbf;
Alter success.
iSQL> ALTER TABLESPACE user_data02 ONLINE;
Alter success.
```

● テーブルスペースバックアップ及び復旧

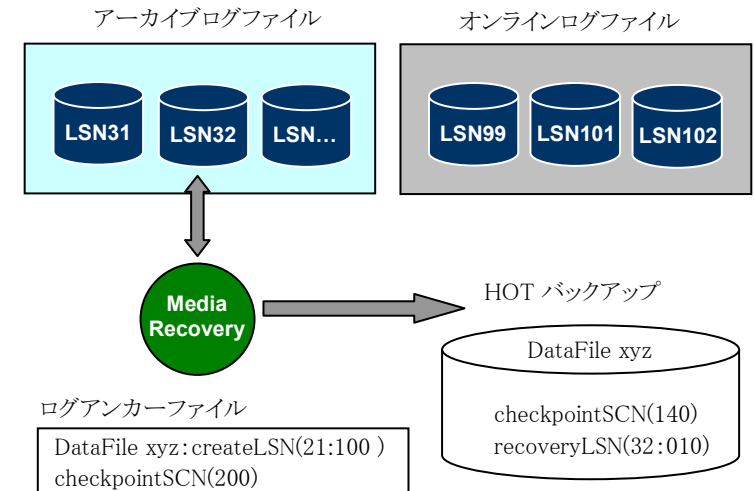
■ テーブルスペース オンラインバックアップ(HOT バックアップ)

オンラインバックアップの特徴

- ✓ データベースがアーカイブログモードで運用されている場合に取得できます。
- ✓ アーカイブログモードでは、チェックポイントとLog Flush後にログファイルをバックアップするため大容量の記憶装置が必要となります。
- ✓ Alter database backupコマンドを使用して、データベースの運用中にバックアップを実行できます。
- ✓ 障害により、データファイルが損傷、削除されてもデータファイルを障害の直前まで復旧(media recovery)できます。

データファイルの復旧の例

- ✓ Datafile xyzが損傷した場合、以前にHotバックアップしておいたデータファイルを使用して復旧できます。
- ✓ バックアップしておいたデータファイルのヘッダにバックアップ時点のチェックポイントされたSCN(140)=recovery LSN(32:010)があるため、これを基準に現在の最終チェックポイントSCN(200)までデータファイルを復旧できます。
- ✓ Startup時にオンラインログのRedo/Undoを行い、データファイルが最終状態のイメージで復旧されます。



< 図 6-3 > オンライン 復旧(Media Recovery)の概念

■ テーブルスペースオフラインバックアップ (Cold バックアップ)

オフラインバックアップの特徴

1. データベースがノンアーカイブモードで運用されている場合に取得できます。
2. オフラインバックアップでは、データベースサーバーを停止した後に、データファイル、ログファイル、log anchorファイルなどをコピーします。
3. 障害でデータファイルが損傷、削除された場合、最終オフラインバックアップされた時点までの復旧が可能です。

オフライン復旧

データベースを停止した後、オフラインバックアップからファイルを復旧 (コピー) した後、**startup** することで復旧が実行されます。

6.4. テーブルスペース削除

● テーブルスペース 削除

■ テーブルスペースオブジェクトのみを削除する方法

```
iSQL> DROP TABLESPACE user_data02;  
Drop success.
```

■ テーブルスペースに関連するオブジェクトも削除する方法

削除するテーブルスペースにテーブルまたはインデックスが存在する場合や、該当オブジェクトを参照する制約が存在する場合、“The tablespace has objects” エラーにより削除が失敗します。このような場合、全てのオブジェクトや参照をあらかじめ削除するか、**INCLUDING/CASCADE** オプションを指定してテーブルスペースを削除します。

```
iSQL> DROP TABLESPACE user_data02  
INCLUDING CONTENTS CASCADE CONSTRAINTS;  
Drop success.
```

■ テーブルスペースに関連するオブジェクトとデータファイルを削除する方法

```
iSQL> DROP TABLESPACE user_data02  
INCLUDING CONTENTS AND DATAFILES CASCADE CON  
STRAINTS;  
Drop success.
```

6.5 テーブルスペース情報

ALTIBASEは、テーブルスペースを管理するため、パフォーマンスビューを提供します。DB管理者は、パフォーマンスビューを参照して、テーブルスペースのサイズ、使用量、状態などを確認することができます。

テーブルスペースの使用情報

```
iSQL> select b.name, b.datafile_count,
a.maxsize || K tbs_max_size,
a.currsize || K tbs_curr_size,
a.usedsize || K tbs_used_size
from (
  select df.spaceid spaceid, maxsize, currsize, nvl(usedsize,n/a) usedsize
  from (select sum(maxsize) * 32 * 1024 maxsize ,
             sum(currsize) * 32 * 1024 currsize ,
             spaceid  from v$datafiles
             where state=1 group by spaceid) df left outer join
  (select tablespace_id spaceid,
             to_char(sum(disk_page_cnt) * 32 * 1024) usedsize
  from v$disktbl_info
  where disk_page_cnt > 0
  group by tablespace_id) uf on df.spaceid = uf.spaceid
  union all
  select 0 spaceid, max_db_size maxsize,
             mem_alloc_page_count * 32 * 1024 currsize,
             to_char((mem_alloc_page_count - mem_free_page_count)
             * 32 * 1024) usedsize
  from (select mem_alloc_page_count, mem_free_page_count
  from v$database),
  (select value1 max_db_size from v$property
  where name = MEM_MAX_DB_SIZE)
) a, v$tablespaces b
where a.spaceid = b.id and b.name = USER_DATA02
group by b.name, b.datafile_count, a.maxsize, a.currsize, a.usedsize;
```

```
NAME FILE_COUNT TBS_MAX_SIZE TBS_CURRSIZE TBS_USED_SIZE
```

```
-----
USER_DATA02 2 1178566656K 209715200K 20971520K
1 row selected.
```

SYSTEM._SYS_TBS_USERS_

- 各テーブルスペースを使用するユーザー情報の確認

```
iSQL> select b.user_name, a.user_id, c.name
  from system._sys_tbs_users_ a, system._sys_users_ b,
  v$tablespaces c
  where a.tbs_id = c.id and a.user_id = b.user_id;
```

```
USER_NAME USER_ID TBS_NAME
```

```
-----
USER01 22 USER_DATA02
```

```
1 rows selected.
```

7. トランザクションとロック

トランザクション

ロックの概要

トランザクション永続性管理ポリシー

7.1. トランザクション(Transaction)

● トランザクションの定義

トランザクション(Transaction)は、1つ以上のSQL文で構成された論理的な作業単位です。

銀行口座への振り込み処理を例にトランザクションを考えて見ます。A口座からB口座に100万の振り込みを行うには、以下の作業が必要となります。

- (1) A口座から100万を引きます。
- (2) B口座に100万を加えます。

この作業は、どちらか一方だけが実行された場合には、預金残高に矛盾が発生するため、1つの作業(トランザクション)として実行されなければなりません。トランザクションは、データベースを完全な状態を維持するため、以下のようなACID特性を満足させなければなりません。

- ✓ **Atomicity** : トランザクション内の全ての命令が反映されるか、もしくは全てが反映されないこと。(All or Nothing)
- ✓ **Consistency** : トランザクションで処理されるデータは、実行前と実行後でデータの整合性を持ち、一貫したデータを確保すること
- ✓ **Isolation** : 多数のトランザクションが同時に実行される時、1つのトランザクションが他のトランザクションの影響を受けないこと。
- ✓ **Durability** : 実行が完了したトランザクションは、どのような状況でもその内容が永続的に維持できること。

● トランザクションのコミット

トランザクションのコミットは、トランザクション内で実行した全てのSQL文の実行結果をデータベースに反映し、該当トランザクションを終了するコマンドです。

ALTIbaseでは、トランザクションがコミットされると以下のような作業を実行します。

- ✓ ログファイルにコミットログを記録します。
- ✓ トランザクションの実行により発生した解放可能なリソースの情報をGarbage Collectorに送ります。
- ✓ トランザクションの状態をコミット状態に変更します。
- ✓ トランザクション実行中に割り当てられたリソースを解放します。(ロック、一時領域、メモリなど)

- トランザクションのロールバック

トランザクションの実行途中にエラーが発生し、処理を継続できない場合には、データベースをトランザクション実行前の状態に戻さなければなりません。これを、トランザクションのロールバックと言います。トランザクションのロールバックは、トランザクション実行中に記録した各ログに対する補償(compensation) 演算を実行することで可能になります。

ALTIBASEでは、トランザクションがロールバックされると以下のような作業を実行します。

- ✓ ログレコードを記録順序の逆に読み取り、補償演算を実行します。
- ✓ トランザクションのロールバックログを記録します。
- ✓ 挿入などの演算で、割り当てられたリソースをGarbage Collectorに返還します。
- ✓ トランザクションの状態をロールバック状態に変更します。
- ✓ トランザクション実行中に割り当てられたリソースを返還します。(ロック、一時領域、メモリなど)

7.2. ロック(Lock)の概要

- ロックモード

ロックは、データベース内に存在する特定のオブジェクトに対するアクセス権を設定します。ロックには、その使用対象により、テーブルレベルロックとレコードレベルロックに分かれます。

テーブルレベルロックモード(Table Level Lock Modes)

ロックモード	説明	機能
S	Shared Lock	レコードロックを確保せずに、テーブルの全てのレコードが 検索できます。また、レコード検索のみを実行する他のトランザクションと同時に実行できます。
X	Exclusive Lock	レコードロックを確保せずに、テーブルの全てのレコードを検索 及び変更できます。そのテーブルに対して、ただ、1つのトランザクションのみが参照、及び更新を実行することができます。
IS	Intensive Shared Lock	レコードロックを確保した後にレコードに対する検索ができます。この点を除き、S モードと同じです。
IX	Intensive Exclusive Lock	レコード ロックを確保した後にレコードに対する検索及び修正ができます。お互いに異なるレコードを 修正するトランザクションは、同時に多数存在できます。
SIX	Intensive Exclusive Shared Lock	レコードロックを確保した後に レコードに対する検索及び修正ができます。レコードに対する修正は、ただ、一つのトランザクションのみが実行できます。

ロック互換性(Lock Compatibility)

ロック互換性とは、既に他のトランザクションが、該当オブジェクトに対してロックを獲得している場合に、別のトランザクションがそのオブジェクトに特定モードのロックを要求したときに、その要求が受け取られるかどうかを決定するために使用されるロックモード 間の互換性のことです。以下の表にALTIBASEのロック互換性を示します。

Requested MODE	Granted Mode					
	NONE	IS	IX	SIX	S	X
IS	○	○	○	○	○	—
IX	○	○	○	—	—	—
SIX	○	○	—	—	—	—
S	○	○	—	—	○	—
X	—	—	—	—	—	—

レコードレベルロックモード(Record Level Lock Modes)

ロックモード	説明	機能
S	Shred Lock	レコードに対して、検索を実行できます。
X	Exclusive Lock	レコードに対して、修正を実行できます。

DML演算の中で、挿入、削除、更新構文は、レコードに対するXロックを取得し検索演算はS ロックを取得します。

一般的にレコードに対するSロックとXロックはお互いに衝突するため互換性はありません。しかし、ALTIBASEでは、多版型同時実行制御を使用しているため、XロックによりSロックの取得が待たされることはありません。従って、更新中であるレコードに対する検索と、検索中であるレコードに対する更新の両方が許可されます。

7.3. トランザクション永続性(Durability) 管理ポリシー

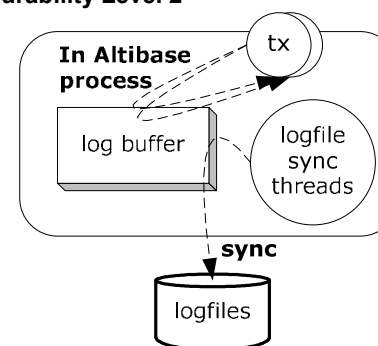
トランザクションの永続性は、トランザクション操作の完了通知をユーザーが受けた時点で、その操作を永続的として結果が失われないことを保障する属性です。

データベース管理システムは、トランザクションの永続性を提供するため、ログ(データベースオブジェクトの更新作業に対する記録)を管理します。コミットされたトランザクションにより更新された内容がディスクに反映される前にシステム障害が発生すると、システムの再起動時にログを読み取り、変更された内容を復旧します。

トランザクションの永続性は、トランザクションの処理性能と密接な関連がある重要な要素です。

- Durability Level の範囲
ALTIBASEのトランザクションdurabilityレベルは、2~5のレベルで分けられます。
- Durability Level の設定方法
altibase.properties ファイルの [TRANSACTION_DURABILITY_LEVEL] プロパティを設定すると、サーバー起動時に該当プロパティに設定されたレベルでALTIBASEサーバーが起動されます。Durability Levelは、運用中に変更することはできません。
- Durability Level の説明

Durability Level 2



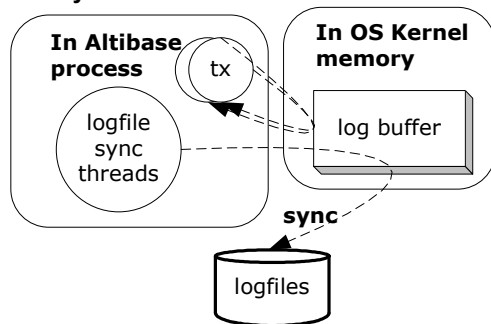
< 図 7-1 > ALTIBASE Durability Level 2

ALTIBASEプロセスのメモリ上のログバッファにログを記録し、sync threadがログバッファにあるログを非同期にログファイルに出力する方式です。

この方式では、トランザクションのコミットログをディスクに反映することを

保障しないため、システム障害やALTIBASEサーバープロセスの異常終了が発生した場合には、トランザクションがコミットした更新内容を保障できないことがあります。

Durability Level 3

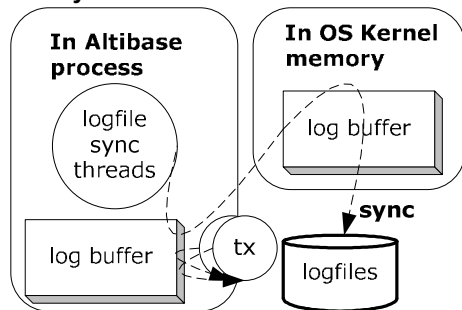


< 図7-2 > ALTIBASE Durability Level 3

durability level 2とは異なり、OSのKernel領域のログバッファにログを出力(非同期I/O)する方式です。

この方式では、ALTIBASEサーバープロセスが異常終了した場合には、トランザクションdurabilityを完全にサポートします。

Durability Level 4

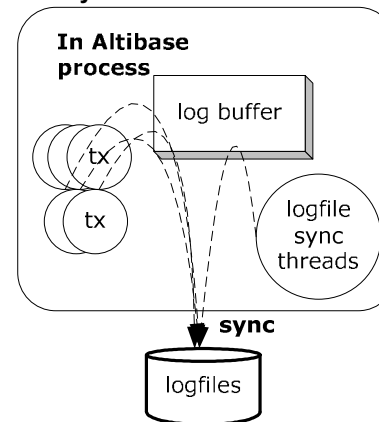


< 図 7-3 > ALTIBASE Durability Level 4

ALTIBASEプロセスのメモリ上のログバッファにログを記録し、sync threadがログバッファにあるログを OS Kernelにあるログバッファに出力(非同期I/O)する方式です。トランザクションがコミットを実行してもdurabilityを保障しません。なぜなら、トランザクションは、コミット時に出力されたログがログファイルまで反映されることを保障しないからです。

この方式は、durability level 2と3の中間の安定性をサポートしつつ、durability level 2の性能を発揮します。

Durability Level 5



< 図 7-4 > ALTIBASE Durability Level 5

ログをプロセス領域のログバッファに記録し、sync threadがログバッファにあるログをログファイルに直接渡すか、各トランザクションが直接コミットログをログファイルまで反映する方式です。

この方式では、トランザクションのコミット実行時にコミットログを包む全てのログが ログファイルに反映されることを保障するため、どのようなシステム障害が発生した場合でも、完全なトランザクションdurabilityを保障します。しかし、全てのコミットされたログがログファイルに反映されるのを待つことが必要のため、トランザクションの処理性能は、durability level 2、3、4より劣ります。

8. バックアップと復旧

バックアップ(Backup)

復旧(Recovery)

バックアップと復旧の実行例

8.1. バックアップ(Backup)

● アルティベース バックアップ

ALTIBASEのバックアップは、以下の2つに分けられます。

- ✓ 論理的バックアップ
- ✓ 物理的バックアップ

論理的バックアップは、**aexport**もしくは**iLoader**を使用してテーブル、及びインデックスの生成スクリプトとテーブルのデータを出力したファイルを作成します。

物理的バックアップは、データベースを構成するデータファイルとログアンカーファイルのコピーを実行します。

物理的バックアップは、データファイルの**snapshot**をコピーする際にサービスの中止によりオンラインバックアップとオフラインバックアップに分かれます。

バックアップ種類	バックアップ方法	バックアップ対象オブジェクト	Restore方法	サービス可否
iLoader利用	iLoaderのOUTオプション利用	指定したテーブル	iLoaderのINオプション利用	○
オンラインバックアップ(DB全体)	SQL文利用 alter database backup database to backup_dir;	システム全体のテーブルスペースのデータファイル、ログアンカーファイル (*1)	1> OSのコピーコマンドでファイルコピー. 2> alter database recover database;	○
オンラインバックアップ(特定テーブルスペース)	SQL文利用 alter database backup tablespace テーブルスペース名 to backup_dir;	指定したテーブルスペースの全てのデータファイル	1> OSのコピーコマンドでファイルコピー. 2> alter database recover database;	○
オフラインバックアップ	1>データベース shutdown 2> UNIXコマンド cp 利用	データベースの全てのデータファイル、そのアンカーファイル、ログファイル	OSのコピーコマンドでファイルコピー	○

(*1) アーカイブログ モード運用時のみ可能

● データベースモード

データベース運用時にデータファイルに反映したオンラインログファイルを管理する方法により、データベースモードはアーカイブログ(archivelog)モードとノアーカイブログ(noarchivelog)モードに分かれます。

アーカイブログモードでは、データファイルに反映した後のログファイルをアーカイブログディレクトリにコピーします。

アーカイブログディレクトリはARCHIVE_DIRプロパティで指定されます。

データベースモード	長所	短所
アーカイブログモード (archivelog)	メディア復旧(media recovery)ができます。つまり、データファイルの消失や損傷が発生しても障害発生時点まで復旧できます。	アーカイブログファイルを持つためのディスク空間が必要です。 DBAの役割が大きくなります。アーカイブログを他の記憶装置に退避することやファイルの整理などの運用が必要になります。アーカイブログ ディスク領域が不足すると障害が発生します。
ノアーカイブログモード (noarchivelog)	アーカイブログファイルの管理が不要なため、運用が容易です。	データファイルの損傷が発生しても、DBAは、オフラインバックアップを利用した復旧しかできません。つまり、復旧がオフラインバックアップされた時点までだけ可能で、バックアップされた 時点からデータファイルの損傷が発生した時点までのデータは消失します。

データベースモードは、create database構文で決定され、ALTIBASE開始時のcontrol フェーズで変更できます。

以下は、アーカイブログモードでデータベースを生成する例です。

```
create database mydb INITSIZE=100M archivelog;
```

controlフェーズでデータベースモードをアーカイブログモードで変更する例は以下の通りです。

```
shell> isql -silent -u sys -p manager -sysdba
[ERR-00000: Connected to idle instance]
iSQL> startup control;
Trying Connect to Altibase.. Connected with Altibase.
TRANSITION TO PHASE: PROCESS
TRANSITION TO PHASE: CONTROL
Command execute success.
iSQL> alter database archivelog;
```

8.2. アルティベース復旧(Recovery)

● 復旧概念

ALTIBASEは、以下の二つの復旧方法を提供します。

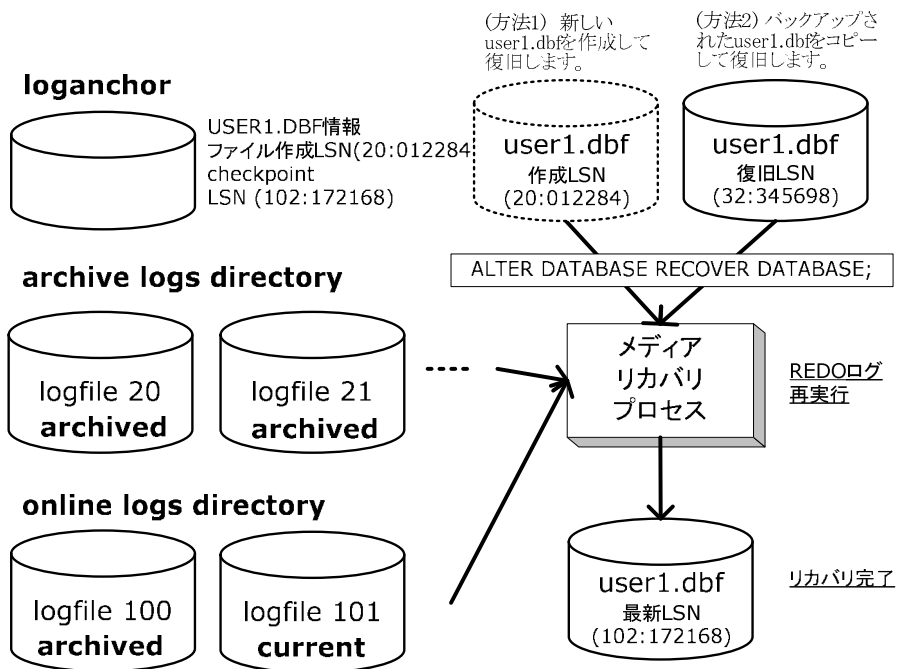
✓ 再起動時の自動復旧 (Restart Recovery)

✓ メディア復旧 (Media Recovery)

再起動時の自動復旧は、ALTIBASEプロセスが異常終了したあとの再起動時のstartupフェーズで自動的に実行される復旧です。

メディア復旧では、特定のデータファイルの消失や損傷が発生した場合に、バックアップされた過去のデータファイルのsnapshotを使用して、バックアップデータファイルの復旧スタートLSNから現在のLSNまでトランザクションを再実行して障害発生直前の状態まで復旧します。

データファイルのメディア復旧の必要性についての判断は、ログアンカーファイル上の該当データファイルのバージョンとデータファイルのバージョンとを比較して決定されます。



< 図 8-1 > アルティベース復旧過程

ALTIBASEのメディア復旧は、controlフェーズでのみ実行できます。つまり、オフライン メディア復旧(offline media recovery)のみサポートします。

- 完全復旧と不完全復旧

ALTIBASEのメディア復旧には、完全復旧と不完全復旧があります。完全復旧では、オンラインログとアーカイブログの消失なしに最新の状態でデータファイルを復元します。

不完全復旧では、アーカイブログまたは、オンラインログが消失している場合に、バックアップされたloganchorとバックアップされた全体データベースを使用して、特定の時点までデータベースを復旧します。

alter database recover database;

不完全復旧は、以下の3つに分けられます。過去の特定時点にデータベース全体を戻す場合。

2007年3月29日にもらった全体データベースバックアップ ファイルを、cpを用い、復旧した後、2007年4月4日のバージョンでデータベース全体が戻されます。

```
alter database recover database
until time '2007-04-04:17:55:00';
```

特定のオンラインログ ファイルが損傷し、最新の時点までredoができない場合、損傷したオンラインログファイルの直前までredoを実行します。

```
alter database recover database until cancel;
```

バックアップされたデータベースとアーカイブログファイルとオンラインログファイルが全て存在し、最新の状態まで復旧が可能な場合、最後のオンラインログファイルまでredoを実行します。

```
alter database recover database until cancel;
```

特に、この場合は、メディア復旧を行うことなく、サーバーを再起動するだけで復旧できます。

Controlフェーズで不完全復旧をした場合、meta フェーズに移行するに、必ず以下の構文を使用します。

```
alter database db-name meta resetlogs;
```

データベースが過去の特定の時点に復元されたため、再起動時の自動復旧(Restart Recover)を実行しないようオンラインログを初期化(resetlogs)します。

metaフェーズでresetlogsを実行してからサービスを開始した場合、オフラインバックアップ、または、オンラインバックアップでデータベース全体バックアップする必要があります。

- メディア復旧時の注意事項

完全復旧の場合は、最新のログアンカーファイルを使用して、メディア復旧を実行します。また、不完全復旧の場合は、バックアップされたログアンカーファイルを利用します。

データベース管理者が誤ってdrop tablespaceやalter tablespaceなどのコマンドでテーブルスペースやデータファイルを削除した場合、最新のログアンカーファイルにテーブルスペース情報が記録されていないためバックアップされたログアンカーファイルを使用します。

メモリテーブルスペースデータファイルに対しては、復旧時にstableなメモリデータファイルを用い、unstableなメモリデータファイルを作らなければなりません。

ALTIBASEは、メモリテーブルスペースに対して、ピンポンチェックポイ

ント(ping-pong checkpoint)を使用するため、ディスク上でメモリーテーブルスペースに対してデータファイルを2つ持ちます。同じイメージを記録している1ペア(pair)のデータファイルは、MEM_DB_DIRプロパティ(Property)に設定された位置に保存されます。2つのデータファイルが両方とも存在することで、ALTIBASEを正常に運用することができます。いずれかの1つの時点では、メモリーテーブルスペースは、ただ一つペアのデータファイルのみ使用されます。

バックアップ時にバックアップ時間を短縮するため、メモリーテーブルスペースの最新のチェックポイントファイルだけをバックアップすることができます。単一のチェックポイントファイルから復旧を行う場合、バックアップされたデータファイルを使用して、もうひとつのデータファイルを作成しなければなりません。

e.g.) バックアップされたメモリーテーブルスペースのデータファイルが3つ(mydb-1-0, mydb-1-1, mydb-1-2)ある場合、メモリーテーブルスペースのデータファイルのコピーは以下のように実行します。

```
shell>cp mydb-1-0 $ALTIBASE_HOME/dbs/mydb-0-0;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs/mydb-0-1;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs/mydb-0-2;
```

- バックアップ時の注意事項

オンラインバックアップとチェックポイントは同時に実行できません。

チェックポイント時にメモリー上のデータベースの内容がディスクに反映されるオンラインバックアップは、バックアップする直前の時点までのデータのみバックアップされることを保障するため、チェックポイントとオンラインバックアップはお互いに排他的に実行されます。

チェックポイントの実行時にオンラインバックアップ要求が実行されると、チェックポイント完了後にオンラインバックアップを開始します。

オンラインバックアップ実行中に、チェックポイント要求が発生すると、チェックポイントは、オンラインバックアップの完了後に実行されます。

ALTIBASEは、ハイブリッドデータベースの特性で、データベースバックアップ時にメモリーテーブルスペース、ディスクテーブルスペースの順番にバックアップを実行します。

メモリーテーブルスペースをバックアップ中には、チェックポイントは実行されませんが、メモリーテーブルスペースのバックアップが完了し、ディスクテーブルスペースのバックアップ中には、メモリーテーブルスペースに対するチェックポイントを実行することができます。

レプリケーション構成情報もバックアップされます。

バックアップされるALTIBASEでレプリケーションを使用している場合、

レプリケーションの構成情報もバックアップされるため、バックアップされたデータベースを異なるシステムに復旧する場合、そのマシンのnetwork ipアドレスなどが異なり、復旧後のレプリケーションで問題が発生することがあります。

従って、レプリケーションを使用している場合には、(1)二重化を中止または削除した後、バックアップを実行する。(2)二重化を使用する二つのシステム内でデータベースを復旧する場合、必ず一つのシステムのALTIBASEを停止した状態で、もう一方のシステムのALTIBASEを復旧してレプリケーションを削除することで、異なるシステムが二重化されることを防ぎます。

データファイルの追加、削除、名称変更のようなテーブルスペースの構成を変更した場合、メモリーテーブルスペースバックアップやデータベース全体をバックアップする必要があります。

ログアンカー ファイルには、データベースのテーブルスペース情報が含まれるため、テーブルスペースの構造が変更されるたびにメモリーテーブルスペースと共にバックアップされなければなりません。

8.3. バックアップ及び復旧例

- iLoaderを用いたテーブルバックアップ及び復旧

任意のテーブルでのみ問題が発生した場合や、何らかの理由により該当テーブルのみバックアップする場合に使用します。

バックアップ前の注意事項

バックアップする前に必ずバックアップするテーブルのスキーマに関するFORMファイル(FORM file)を生成する必要があります。FORMファイルとは、テーブルに関する基本情報(型、名、データ型)を記録したファイルです。

```
ex) テーブルt1のform file生成
(t1.fmtと言うファイル名で生成されます。)
lloader > formout -T t1 -f t1.fmt
```

バックアップ方法

iLoaderのオプションの中でoutオプションを使用します。

ユーザーが設定した名前でもテーブルに対するバックアップファイルが生成されます。

```
ex) テーブルt1のバックアップ
(利用する form ファイルは、t1.fmt 生成するバックアップファイルは、t1.dat )
```

復旧(restore)方法

iLoaderのオプションの中でinオプションを使用します。

復旧時にテーブルにレコードが存在する場合、そのレコードを維持するか、置き換えるかを選択できます。特に設定しなかった場合は、存在するレコードの内容は維持されます。

```
ex) テーブルt1の復旧
iloader > in -T t1 -d t1.dat -f t1.fmt
```

● Offline バックアップ及び復旧

オフラインバックアップ及び復旧は、主に、非アーカイブログモードの場合に使用します。

バックアップ前の注意事項

バックアップする前にALTIBASEと関連する全てのサービスを停止します。

サービス実行中にオフラインバックアップを実行すると、ログファイルの内容が変更されると同時にバックアップが実行されて、バックアップが正確に実行できないことがあります。従って、オフラインバックアップ時には、必ずALTIBASEを停止してから実行してください。

バックアップ方法

テーブルスペースのデータファイル(ログファイル、ログアンカーファイル)をUNIXコマンド `cp`を用い、バックアップします。

注意事項

- ✓ メモリデータファイルだけでなく、ディスク関連のテーブルスペースのデータファイルもバックアップしなければなりません。
- ✓ メモリテーブルスペースデータファイルの保存位置は、ALTIBASEプロパティファイルのMEM_DB_DIRで設定されています。
- ✓ メモリテーブルスペースのデータファイルをバックアップするためにはMEM_DB_DIRディレクトリを全てコピーする必要があります。
- ✓ ログアンカーファイルの位置は、LOGANCHOR_DIRプロパティ(Property)で設定されています。
- ✓ ログアンカーファイルをバックアップするためにはLOGANCHOR_DIRディレクトリのファイルをコピーしてから、ディスクのテーブルスペースのデータファイルをコピーします。

```
ex) プロパティファイルの
MEM_DB_DIR=$ALTIBASE_HOME/dbs0
MEM_DB_DIR=$ALTIBASE_HOME/dbs1
LOGANCHOR_DIR=$ALTIBASE_HOME/logs
ディスクテーブルスペースは system tablespace,undo
tablespace,
```

temp tablespaceのみある場合です。

バックアップファイルが保存される位置 : /home/backup

```
$cp -r $ALTIBASE_HOME/dbs0 /home/backup
```

```
$cp -r $ALTIBASE_HOME/dbs1 /home/backup
```

```
$cp -r $ALTIBASE_HOME/logs /home/backup
```

```
$cp -r $ALTIBASE_HOME/dbs/system*.dbf /home/backup
```

復旧方法

アルティベース設定ファイルは、バックアップされる当時の設定ファイルをそのまま利用します。

共有メモリに残っているデータベースを削除します。

バックアップ時に取得したファイルを、OSのコピーコマンドを使用して復旧します。

```
ex) 上でバックアップされたデータベースを用い、復旧
$cp -r /home/backup/dbs0 ALTIBASE_HOME/dbs0
$cp -r /home/backup/dbs1 $ALTIBASE_HOME/dbs1
$cp -r /home/backup/logs $ALTIBASE_HOME/logs
$cp -r /home/backup/system*.dbf $ALTIBASE_HOME/dbs
$cp -r /home/backup/undo.dbf $ALTIBASE_HOME/dbs
$cp -r /home/backup/temp.dbf $ALTIBASE_HOME/dbs
```

● メディア復旧例 1

シナリオ

アーカイブログモードでデータベースを運用し、バックアップされてないデータファイル /home1/dbs/abc.dbfが消失した。

復旧方法

(1) Controlフェーズで、空のデータファイルabc.dbfを生成します。

```
shell> isql -silent -u sys -p manager -sysdba
[ERR-00000: Connected to idle instance]
iSQL> startup control;
Trying Connect to Altibase.. Connected with Altibase.
TRANSITION TO PHASE: PROCESS

TRANSITION TO PHASE: CONTROL
Command execute success.
iSQL> alter database create datafile '/home1/dbs/abc.dbf';
```

- (2) メディア復旧を実行します。

```
iSQL> alter database recover database;
```

● メディア復旧例 2

シナリオ

アーカイブログモードでデータベースを運用し、三日前のテーブルスペースusersのデータファイルをバックアップした。

本日の午前中にusersテーブルスペースのデータファイルを全て失った。

バックアップ方法

三日前に実行したバックアップは以下の通りです。

```
iSQL>alter database backup tablespace users to '/backup1';
shell> ls /backup1
user001.dbf user002.dbf user003.dbf
```

復旧方法

- (1) バックアップディレクトリに保存していたバックアップ データファイルを users テーブルスペースのファイルがあった \$ALTIBASE_HOME/dbs/にコピーします。

```
shell> cp /backup1/*.dbf $ALTIBASE_HOME/dbs;
```

- (2) 以下のようにメディア復旧を実行します。

```
shell> isql -silent -u sys -p manager -sysdba
[ERR-00000: Connected to idle instance]
iSQL> startup control;
Trying Connect to Altibase.. Connected with Altibase.
TRANSITION TO PHASE: PROCESS
TRANSITION TO PHASE: CONTROL
Command execute success.
```

- (3) メディア復旧を実行します。

```
iSQL> alter database recover database;
```

● メディア復旧例 3

シナリオ

アーカイブログモードでデータベースを運用し、七日前にテーブルスペースtaskのデータファイルをバックアップした。

本日の午後にtaskテーブルスペースのデータファイルがある/disk1ファイルシステムが損傷したが/disk2のファイルシステムは無事だった。

バックアップ方法

七日前に実行したバックアップは以下の通りです。

```
iSQL>alter database backup tablespace task to /backup1;
shell> ls /backup1
task001.dbf task002.dbf
```

復旧方法

- (1) バックアップ ディレクトリに保存しておいたバックアップデータファイルをtaskテーブルスペースのファイルを /disk2ファイルシステムにコピーします。

```
shell> cp /backup1/*.dbf /disk2/dbs;
```

- (2) 以下のようにメディア復旧を実行します。

```
shell> isql -silent -s 127.0.0.1 -u sys -p manager -sysdba
[ERR-00000: Connected to idle instance]
iSQL> startup control;
Trying Connect to Altibase.. Connected with Altibase.
TRANSITION TO PHASE: PROCESS
TRANSITION TO PHASE: CONTROL
Command execute success.
```

- (3) taskテーブルスペースデータファイルの位置を変更します。

```
iSQL> alter database rename datafile
'/disk1/dbs/task001.dbf' to
'/disk2/dbs/task001.dbf';

iSQL> alter database rename datafile '/disk1/dbs/task002.dbf'
to '/disk2/dbs/task002.dbf';
```

- (4) メディア復旧を実行します。

```
iSQL> alter database recover database;
```

● メディア復旧例 4

シナリオ

アーカイブログモードでデータベースを運用し、DBAの運用ミスによりテーブルsummaryを削除した。削除された時間は2005年4月6日22時30分であり、10分前にデータベースを戻したい。

バックアップ方法

不完全メディア復旧をするためには、全体データベースバックアップが必要です。

```
iSQL>alter database backup database to '/backup1';
```

復旧方法

- (1) バックアップされたデータファイルを元の位置にコピーします。

```
shell> cp /backup1/*.dbf $ALTIBASE_HOME/dbs;
```

バックアップされたloganchorファイルをコピーします。

```
shell> cp /backup1/loganchor* $ALTIBASE_HOME/logs
```

- (2) メモリテーブルスペースは、ping-pongチェックポイントを使用しており、メモリテーブルバックアップ時にstableなデータベースファイルのみコピーされています。バックアップされたstableなバージョンのデータファイルを使用して、unstableなバージョンを作り、コピーします。

e.g.) バックアップされたメモリテーブルスペースのデータファイルは、以下の通りです。

mydb-1-0, mydb-1-1, mydb-1-2

コピーは、以下のように実行します。

```
shell>cp mydb-1-0 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-0 $ALTIBASE_HOME/dbs/mydb-0-0;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs/mydb-0-1;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs/mydb-0-2;
```

- (3) Controlフェーズで、メディア復旧に必要なアーカイブログファイルを検索します。以下で、検索された番号のログファイルがonline logディレクトリに存在していなければなりません。

```
iSQL> SELECT last_deleted_logfile FROM v$LFG;
LAST_DELETED_LOGFILE
```

32

- (4) 適切なアーカイブされたログファイル32番からonline logディレクトリに全てが存在するようにコピーします。
- (5) 不完全メディア復旧を以下のように実行します。

```
shell> isql -silent -u sys -p manager -sysdba
[ERR-00000: Connected to idle instance]
iSQL> startup control;
Trying Connect to Altibase.. Connected with Altibase.
TRANSITION TO PHASE: PROCESS
TRANSITION TO PHASE: CONTROL
Command execute success.
iSQL> alter database recover database until time
2005-04- 06:22:20:00;
```

- (6) 不完全メディア復旧を実行したため、metaフェーズに移行する前にresetlogsをします。

```
iSQL> alter database db_name meta resetlogs;
```

- (7) resetlogsを実行したため、データベース全体をバックアップします。

```
iSQL> alter database backup database to '/backup1';
```

● メディア復旧例 5

シナリオ

アーカイブログモードでデータベースを運用し、オンラインログファイルlog499~ 520があります。その中で、中間logである510番ログファイルが損傷したため、log 509番までデータベースを復旧したい。

バックアップ方法

- (1) バックアップされたデータファイルを元の位置にコピーします。

```
shell> cp /backup1/*.dbf $ALTIBASE_HOME /dbs;
```

- (2) バックアップされたloganchorファイルをコピーします。

```
shell> cp /backup1/loganchor* $ALTIBASE_HOME/logs
```

- (3) メモリテーブルスペースは、ping-pongチェックポイントを使用してお

り、メモリテーブルバックアップ時に**stable**なデータベースファイルのみコピーされました。バックアップされた**stable**なバージョンのデータファイルを用い、**unstable**なバージョンをつくりコピーします。

e.g.) バックアップされたメモリテーブルスペースのデータファイルは、以下の通りです。:

mydb-1-0, mydb-1-1, mydb-1-2
コピーは、以下のように実行されなければなりません。

```
shell>cp mydb-1-0 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-0 $ALTIBASE_HOME/dbs/mydb-0-0;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs/mydb-0-1;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs/mydb-0-2;
```

- (4) Controlフェーズでメディア復旧に必要なアーカイブログファイルを検索します。以下で 検索された番号のログファイルからはonline logディレクトリに存在していなければなりません。

```
iSQL> SELECT last_deleted_logfile FROM v$LFG;
LAST_DELETED_LOGFILE
```

489

- (5) 適切なアーカイブされたログファイル489番からonline log ディレクトリに全てが存在するようにコピーします。

- (6) 以下のように不完全メディア復旧を以下のように実行します。

```
shell> isql -silent -u sys -p manager -sysdba
[ERR-00000: Connected to idle instance]
iSQL> startup control;
Trying Connect to Altibase.. Connected with Altibase.
TRANSITION TO PHASE: PROCESS
TRANSITION TO PHASE: CONTROL
Command execute success.
```

- (7) Log499~509までのみredoを実行し、510番以後のlogに対してはredoを実行しません。

```
iSQL> alter database recover database until cancel;
```

- (8) 不完全メディア復旧を実行したため、metaフェーズに移行する前にresetlogsをします。

```
iSQL> alter database db_name meta resetlogs;
```

- (9) reset logを実行したため、データベース全体をバックアップします。

```
iSQL> alter database backup database to '/backup1';
```

● メディア復旧例 6

シナリオ

アーカイブログモードでデータベースを運用しつつ、メモリテーブルスペースのデータファイルが損傷された。

メモリテーブルスペースは、システムカタログなど重要システムデータが保存されている空間であるため、以前にもらった全体データベースバックアップを用い、復旧します。

バックアップ方法

メモリテーブルスペースのデータファイルの損傷を復旧するためには、全体データベースバックアップが必要です。

```
iSQL>alter database backup database to '/backup1';
```

復旧方法

- (1) バックアップされたデータファイルを元の位置にコピーします。

```
shell> cp /backup1/*.dbf $ALTIBASE_HOME/dbs;
```

- (2) バックアップされたloganchorファイルコピー本をcopyします。

```
Shell> cp /backup1/loganchor* $ALTIBASE_HOME/logs
```

- (3) メモリテーブルスペースは、ping pongチェックポイント技法を使用していて、メモリテーブルバックアップ時に**stable**なデータベースファイルのみコピーされました。ここで**Stable**な データベースファイルは、最後のチェックポイントを通じ、メモリ上で変更されたページを反映したひとペアのデータファイルの中で一つのことです。バックアップされた**stable**なバージョンのデータファイルを用い、**unstable**なバージョンを作り、コピーします。

e.g.) バックアップされたメモリテーブルスペースのデータファイルは、以下の通りです。:

mydb-1-0, mydb-1-1, mydb-1-2
コピーは、以下のように実行されなければなりません。

```
shell>cp mydb-1-0 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-0 $ALTIBASE_HOME/dbs/mydb-0-0;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs/mydb-0-1;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs/mydb-0-2;
```

- (4) control startup段階で、メディア復旧に必要なアーカイブログファイルを検索します。以下で検索された番号のログファイルからはonline logディレクトリに存在する必要があります。

```
iSQL> SELECT last_deleted_logfile FROM v$LFG;
LAST_DELETED_LOGFILE
-----
489
```

- (5) 適切なアーカイブされたログファイル489番からonline logディレクトリに全てが存在するようにコピーします。
- (6) メディア復旧を以下のように実行します。

```
shell> isql -silent -u sys -p manager -sysdba
[ERR-00000: Connected to idle instance]
iSQL> startup control;
Trying Connect to Altibase.. Connected with Altibase.
TRANSITION TO PHASE: PROCESS
TRANSITION TO PHASE: CONTROL
Command execute success.
iSQL> alter database recover database;
```

● メディア復旧例 7 シナリオ

アーカイブログモードでデータベースを運用しつつ、DBA間違いでテーブルスペースtaskを dropした。時間は、2005年4月6日22時30分だった。すぐ10分前にデータベースを戻したい。

バックアップ方法

不完全メディア復旧をするためには、全体データベースバックアップが必要です。

```
iSQL>Alter database backup database to '/backup1';
```

復旧方法

- (1) バックアップされたデータベース データファイルを元の位置にコピーします。

```
shell> cp /backup1/*.*.dbf /disk1/dbs;
```

- (2) メモリテーブルスペースは、ping pongチェックポイント技法を使用していて、メモリテーブルバックアップ時にstableなデータベースファイルのみコピーされました。ここで、Stableな データベース ファイルは最後のチェックポイントを通じ、メモリ上で変更されたページを反映した一つペアのデータファイルの中の一つです。バックアップされたstableなバージョンの データファイルを用い、unstableなバージョンを作り、コピーします。

e.g.) バックアップされたメモリテーブルスペースのデータファイルは、以下の通りです。:

mydb-1-0, mydb-1-1, mydb-1-2

コピーは、以下のように実行されなければなりません。

```
shell>cp mydb-1-0 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-0 $ALTIBASE_HOME/dbs/mydb-0-0;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-1 $ALTIBASE_HOME/dbs/mydb-0-1;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs;
shell>cp mydb-1-2 $ALTIBASE_HOME/dbs/mydb-0-2;
```

- (3) バックアップログアンカーファイルもコピーします。(現在ログアンカーにはdropされたテーブルスペース情報がないからです。)

```
shell> cp /backup1/loganchor* $ALTIBASE_HOME/logs;
```

- (4) control startup段階でメディア復旧に必要なアーカイブログファイルを検索します。以下で 検索された番号のログファイルからはonline logディレクトリに存在する必要があります。

```
iSQL> SELECT last_deleted_logfile FROM v$LFG;
LAST_DELETED_LOGFILE
-----
489
```

適切なアーカイブされたログファイル489番からonline logディレクトリに全てが存在するようにコピーします。

- (5) 不完全メディア復旧を以下のように実行します。

```
shell> isql -silent -u sys -p manager -sysdba
[ERR-00000: Connected to idle instance]
iSQL> startup control;
Trying Connect to Altibase.. Connected with Altibase.
TRANSITION TO PHASE: PROCESS
TRANSITION TO PHASE: CONTROL
Command execute success.
iSQL> alter database recover database until time
'2005-04-06:22:20:00';
```

- (6) 不完全メディア復旧を実行したため、`meta startup`段階に行く前に `resetlogs`をします。

```
iSQL> alter database db_name meta resetlogs;
```

- (7) `resetlogs`を実行したため、データベース全体バックアップをさせます。

```
iSQL> alter database backup database to '/backup1';
```

9. ALTIBASE の監視

監視項目

監視スクリプト(Monitoring Scripts)

9.1. 監視項目

セッション/Statementモニターリング

アルティベース運用中、連結されているセッションに対する情報を確認します。一つのセッションは多数のStatement1を割り当ててもらって使用可能で、セッション別にセッションの属性が異なることも可能です。

データベース(テーブル/インデックス)情報モニターリング

生成されたテーブルに対して情報を確認します。全体データベースに対する情報とテーブルスペース各テーブル及びインデックス情報が確認できます。

メモリ使用量モニターリング

アルティベースサーバーが運用しつつ、使用するメモリ領域の情報を確認します。メモリテーブルスペースのデータ(バージョン)保存の領域及びインデックス保存の領域、質疑処理のための臨時領域、セッション情報の保存空間、メモリバッファフルなどのメモリ使用量の情報が確認できます。

状態モニターリング

二重化状態の情報を確認します。二重化関連スレッド(Thread)(Sender/Receiver)状態と二重化 データ伝送状態が確認できます。

9.2. モニターリングスクリプト(Monitoring Scripts)

● ADM_SESSION

V\$SESSION は、現在アクセス中である全てのユーザーセッションの情報を照会。

実行例)

```
iSQL> select * from adm_session;
ID UNAME TASK_STATE IDLE_START_TIME
-----
PID COMM_NAME STMT_COUNT AUTOCOMMIT
----- 13 MON EXECUTING -
23234 TCP 127.0.0.1:35847 1 Y
14 SYS WAITING 08/16 17:12:04
23275 UNIX 0 N
```

VIEW)

```
iSQL> create or replace view adm_session
as
select
id,
db_username uname,
task_state,
decode(idle_start_time, 0, '-', to_char(to_date('19700101', 'yyyymmdd') +
idle_start_time / (1*24*60*60), 'mm/dd hh:mi:ss')) idle_start_time,
client_pid as pid,
replace2(replace2(comm_name, 'socket-', null), '-server', null) comm_name,
opened_stmt_count stmt_count,
decode(autocommit_flag, 1, 'Y', 'N') commitmode
from v$session ; iSQL> select * from adm_session;
```

重要列)

ID	セッションID
UNAME	DBアクセスユーザー名、“ADM_..” で名づけられた全てのビューで同じ意味。
COMM_NAME	ClientセッションがアクセスされたIP
PID	ClientプロセスのPID、ただしJDBCアクセスしている場合は、0

● ADM_TABLES

各テーブルが、メモリテーブルであるか、ディスクテーブルであるか、どちらのテーブルスペースに位置するのか、割り当てられたサイズはどのくらいかを照会。

実行例)

```
iSQL> select * from adm_tables order by size desc limit 5;
UNAME TBS_NAME TNAME SIZE IS_MEMORY
-----
QA2 SYS_TBS_MEM_DATA T_NATC_STATIC 508.562 Y
QA SYS_TBS_DISK_DATA WARNING_LIST 155 N
QA SYS_TBS_MEM_DATA NATC_STATIC 150.718 Y
QA2 SYS_TBS_MEM_DATA T_TEST_PROPERTY 112.468 Y
QA SYS_TBS_MEM_DATA A4_ATC_HISTORY 102.687 Y
```

VIEW)

```
iSQL> create or replace view adm_tables
as
select a.user_name uname, b.name tbs_name, c.table_name tname,
-- c.table_id,
decode(e.table_oid, NULL, d.fixed_alloc_mem + d.var_alloc_mem,
b.extent_page_count * b.page_size * e.extent_total_count
) / (1024*1024) size,
decode(e.segment_type, NULL, 'Y', 'N') is_memory
-- replication_count repl_cnt
from system_sys_users_a,
v$tablespaces b,
system_sys_tables_c
left outer join v$memtbl_info d on c.table_oid = d.table_oid
left outer join v$segment e on c.table_oid = e.table_oid
and e.segment_type = 'TABLE'
where a.user_name <> 'SYSTEM_'
and c.table_type = 'T'
and a.user_id = c.user_id
and b.id = c.tbs_id;
```

重要列)

TNAME	テーブル名
TBS_NAME	テーブルスペース名
SIZE	テーブルが割り当てたサイズ (MB)、インデックスサイズは除く

IS_MEMORY	メモリテーブルであるか/ディスクテーブルであるか、V\$SEGMENTはディスク資源に対する情報であることを利用
<> 'SYSTEM_'	DBメタテーブルの所有ユーザーであるSYSTEM_ は除くための条件

● ADM_CONSTRAINTS

参照キー(Foreign Key, FK)、ユニークキー(Unique Key, UK)、Primaryキー(Primary Key, PK)などを照会

実行例)

```
iSQL> select const_type as t, const_name, index_name, r_table, r_index
from adm_constraints where uname = 'MIS' and tname = 'T_AUTH_MEMBER';
T CONST_NAME INDEX_NAME R_TABLE R_INDEX
-----
F T_AUTH_MEMBER_FK1 T_ORG_EMP T_ORG_EMP_PK
P T_AUTH_MEMBER_PK
U __SYS_CON_PRIMARY_ID_2301 __SYS_IDX_ID_135
```

実行例 - PKがないテーブル照会)

```
iSQL> select uname, tname from adm_tables
where (uname, tname) not in ( select uname, tname from adm_constraints
where const_type = 'P' );
```

VIEW)

```
iSQL> create or replace view adm_constraints
as
select a.user_name uname, b.table_name tname,
decode( c.constraint_type, 0, 'F', 2, 'U', 3, 'P', '?' ) const_type,
c.constraint_name const_name,
decode( d.index_name, c.constraint_name, NULL, index_name ) index_name,
( select table_name from system_sys_tables_ where table_id =
c.referenced_table_id ) r_table,
( select index_name from system_sys_indices_
where index_id = c.referenced_index_id ) r_index
from system_sys_users_ a,
system_sys_tables_ b,
system_sys_constraints_ c
left outer join system_sys_indices_ d on c.index_id = d.index_id
where c.table_id = b.table_id
and a.user_name <> 'SYSTEM_'
and c.user_id = a.user_id
-- 1 : NOT NULL
and c.constraint_type <> 1
order by table_name, const_type ;
```

重要列)

CONST_TYPE	Constraint種類, F : FK, P : PK, U : UK
CONST_NAME	Constraint名
INDEX_NAME	CONST_NAMEと同じではない場合のみ出力
R_TABLE	FKが参照するテーブル名
R_INDEX	FKが参照するインデックス名

● ADM_SEGMENTS

SEGMENTはDBMS内のOBJECTの中、物理的な保存空間を持つオブジェクトを意味します。各 SEGMENTの割り当て量、テーブルスペース、(インデックスSEGMENTである場合) 所属テーブルなどを照会、現在バージョンのV\$SEGMENT性能ビューは、ディスクテーブルの情報のみ存在するため、メモリーテーブルの情報は照会できません。

実行例)

```
iSQL> select const_type as t, const_name, index_name, r_table, r_index
from adm_constraints where uname = 'MIS' and tname = 'T_AUTH_MEMBER';
T CONST_NAME INDEX_NAME R_TABLE R_INDEX
-----
F T_AUTH_MEMBER_FK1 T_ORG_EMP T_ORG_EMP_PK
P T_AUTH_MEMBER_PK
U __SYS_CON_PRIMARY_ID_2301 __SYS_IDX_ID_135
```

実行例 - ディスク テーブルとインデックスのテーブルスペースが分離されていないテーブル 照会)

```
iSQL> select a.tbs_name, a.tname, b.segment_name index_name
from adm_tables a, adm_segments b
where a.uname = b.uname
and a.tname = b.r_segment
and a.tbs_name = b.tbs_name
and b.segment_type = 'INDEX' ;
TBS_NAME TNAME INDEX_NAME
-----
SYS_TBS_DISK_DATA D1 __SYS_IDX_ID_442
SYS_TBS_DISK_DATA WARNING_LIST __SYS_IDX_ID_395
```

VIEW)

```
iSQL> create or replace view adm_segments
as
select a.user_name uname, d.table_name segment_name, c.segment_type,
b.name tbs_name, b.extent_page_count * b.page_size * c.extent_total_count /
(1024*1024) alloc_size, c.extent_total_count extent_count,
c.extent_full_count extent_full," r_segment
from system_sys_users_a, v$tablespaces b, v$segment c,
system_sys_tables_d
where c.table_oid = d.table_oid and d.user_id = a.user_id
and c.space_id = b.id and c.segment_type = 'TABLE'
union all
select a.user_name uname, d.index_name segment_name, c.segment_type,
b.name tbs_name, b.extent_page_count * b.page_size * c.extent_total_count /
(1024*1024) alloc_size, c.extent_total_count extent_count,
c.extent_full_count extent_full,
(select table_name from system_sys_tables_ where table_id = d.table_id
) r_segment
from system_sys_users_a, v$tablespaces b, v$segment c,
(
select aa.user_id, aa.index_name, bb.index_seg_desc
from system_sys_indices_aa, v$index bb
where aa.index_id = bb.index_id
) d
where a.user_id = d.user_id and c.segment_desc = d.index_seg_desc
and c.space_id = b.id and c.segment_type = 'INDEX' ;
```

重要列)

SEGMENT_NAME	物理的な保存空間を使用するSEGMENT名
SEGMENT_TYPE	2種類 : TABLE、INDEX
EXTENT_COUNT	総割り当てられたEXTENT数
EXTENT_FULL	空いている空間がないEXTENT数
R_SEGMENT	参照SEGMENT、インデックスが属しているテーブル名

● ADM_TRANSACTION

V\$TRANSACTIONは、実行中である全てのトランザクション(以下TX)の情報を照会。

実行例)

```
iSQL> select tx_id, firstlog, time, status from adm_transaction;
TX_ID FIRSTLOG UPDATE_TIME STATUS
-----
42176 313 4 BLOCKED
83104 313 2500 BEGIN
232768 READ_TX 0 BEGIN
```

VIEW)

```
iSQL> create or replace view adm_transaction
as
select a.id tx_id, decode(a.first_undo_next_lsn_fileno,
-1, 'READ_TX', a.first_undo_next_lsn_fileno) firstlog,
decode(a.first_update_time, 0, 0, d.base_time - a.first_update_time)
update_time, decode(c.autocommit_flag, 1, 'Y', 'N') autocommit,
decode (a.status, 0, 'BEGIN', 1, 'PRECOMMIT', 2, 'COMMIT_IN_MEMORY',
3, 'COMMIT', 4, 'ABORT', 5, 'BLOCKED', 6, 'END', '?') status,
decode(a.update_status, 0, 'READ', 1, 'UPDATING', '?') update_status,
--a.memory_view_scn mvscn,
--a.disk_view_scn dvscn,
b.session_id ss_id, b.id stmt_id, c.comm_name,c.client_pid, b.query
from v$transaction a, v$statement b, v$session c, v$sessionmgr d
where a.status != 6 and a.id = b.tx_id and b.session_id = c.id
--and a.first_update_time <> 0
order by a.first_update_time desc ;
```

重要列)

TX_ID	トランザクションID
FIRSTLOG	TXが参照する最も少ないログファイル番号
UPDATE_TIME	変更TX以後経過時間(秒)
STATUS	TXの状態
UPDATE_STATUS	変更TX
SS_ID	セッションID
STMT_ID	Statement ID
QUERY	Statement ID

● ADM_MEMTBL : メモリテーブル詳細情報(容量など)

実行例)

```
iSQL> select tname, alloc_size, used_size, efficiency from adm_memtbl
where alloc_size > 100 order by alloc_size desc;
TNAME ALLOC_SIZE USED_SIZE EFFICIENCY
-----
T_NATC_STATIC 508.563 506.202 99.54
NATC_STATIC 150.719 150.307 99.73
T_TEST_PROPERTY 112.469 111.979 99.56
A4_ATC_HISTORY 102.688 102.226 99.55
```

VIEW)

```
iSQL> create or replace view adm_memtbl
as
select a.user_name uname, b.table_name tname, d.name tbs_name,
--c.table_oid,
c.mem_slot_size fixed_row_size,
round( (c.fixed_alloc_mem + c.var_alloc_mem) / (1024*1024) , 3 ) alloc_size,
round( (c.fixed_used_mem + c.var_used_mem) / (1024*1024) , 3 ) used_size,
round( (c.fixed_used_mem + c.var_used_mem)/(c.fixed_alloc_mem + c.var_al-
loc_mem) * 100 , 2 ) efficiency
from system_sys_users_a, system_sys_tables_b, v$memtbl_info c,
v$tablespaces d
where a.user_name <> 'SYSTEM_' and b.table_type = 'T'
and a.user_id = b.user_id and b.table_oid = c.table_oid and b.tbs_id = d.id ;
```

重要列)

FIXED_ROW_SIZE	可変長さ列を除いた一つのレコードの長さ(Byte), 可変長さ列の例)メモリテーブルのVARCHAR(127)異常列及び BLOB
ALLOC_SIZE	テーブルが割り当てたサイズ (MB)
USED_SIZE	ALLOC_SIZEで空いている空間を除いたサイズ(MB)
EFFICIENCY	USED_SIZE * 100 / ALLOC_SIZE

- ADM_INDEX : インデックス基本情報

実行例)

```
iSQL> select index_name, tname, tbs_name, is_unique from adm_index
where tbs_name not like '%MEM%' and tname like 'T_%';
INDEX_NAME TNAME TBS_NAME IS_UNIQUE
-----
__SYS_IDX_ID_128 TO_MA_MASTER_ORG TBS_ALTIMIS_IDX Y
T_AM_HOST_PK T_AM_HOST TBS_ALTIMIS_IDX Y
```

VIEW)

```
iSQL> create or replace view adm_index
as
select a.user_name uname, c.index_name index_name,
--c.index_id,
b.table_name tname, nvl(d.name, 'SYS_TBS_MEMORY') tbs_name,
decode(c.is_unique, 'T', 'Y', 'N') is_unique
--, c.column_cnt "#COLUMN"
from system_sys_users_a, system_sys_tables_b, system_sys_indices_c
left outer join v$tablespaces d on c.tbs_id = d.id
where a.user_name <> 'SYSTEM_' and b.table_type = 'T'
and c.table_id = b.table_id
and c.user_id = a.user_id
order by b.table_name, c.index_name ;
```

重要列)

INDEX_NAME	インデックス名
IS_UNIQUE	ユニークインデックス
TNAME	インデックスが属しているテーブル名
TBS_NAME	インデックスが属しているテーブルスペース名

- ADM_INDEX_COLUMN : インデックス構成列の情報

実行例)

```
iSQL> select
decode(position,0,index_name,NULL) index_name,
column_name, descend
, lpad(' ', 2*(position))||column_name||' ||descend||', column_name
from adm_index_column where tname = 'T_SVN_HISTORY';
INDEX_NAME COLUMN_NAME DESCEND COLUMN_NAME
-----
T_SVN_HISTORY_PK PK ASC PK ASC,
T_SVN_HISTORY_UNIQUE REPOSITORY ASC REPOSITORY ASC,
REVISION ASC REVISION ASC,
ACTION ASC ACTION ASC,
BRANCHNAME ASC BRANCHNAME ASC,
DIRNAME ASC DIRNAME ASC,
FILENAME ASC FILENAME ASC,
```

VIEW)

```
iSQL> create or replace view adm_index_column
as
select d.user_name uname, c.table_name tname, b.index_name,
e.column_name, a.index_col_order position,
decode(a.sort_order, 'A', 'ASC', 'D', 'DESC') descend
from system_sys_index_columns_ a, system_sys_indices_ b,
system_sys_tables_ c, system_sys_users_ d, system_sys_columns_ e
where d.user_name <> 'SYSTEM_' and c.table_type = 'T'
and a.index_id = b.index_id and a.table_id = c.table_id
and a.user_id = d.user_id and a.column_id = e.column_id
order by uname, tname, index_name, position ;
```

重要列)

COLUMN_NAME	インデックス列名
POSITION	結合インデックスで列の順番
DESCEND	整列順番

● ADM_OBJ_PRIVS , ADM_SYS_PRIVS

ADM_OBJ_PRIVSは、OBJECTに対する権限、ADM_SYS_PRIVSは、非オブジェクト性権限関係を照会

実行例)

```
iSQL> select grantor, priv_name, object_name from adm_obj_privs where
grantee in 'MIS';
GRANTOR PRIV_NAME OBJECT_NAME
-----
ORG REFERENCES T_ORG_EMP
QA2 SELECT T_NATC_STATIC
QA2 SELECT V_RESULT_BASELINE
iSQL> select * from adm_sys_privs;
GRANTEE GRANTOR PRIV_NAME
-----
MIS SYS CREATE SESSION
MIS SYS CREATE TABLE
```

VIEW)

-- オブジェクト権限

```
iSQL> create or replace view adm_obj_privs
as
select
a.user_name grantee, f.user_name owner, e.table_name object_name,
e.table_type object_type, c.user_name grantor,
replace(d.priv_name, '_', '') priv_name,
decode(b.with_grant_option, 0, 'N', 'Y') grantable
from system_sys_users_ a, system_sys_grant_object_ b,
system_sys_users_ c, system_sys_privileges_ d,
system_sys_tables_ e, system_sys_users_ f
where c.user_name <> 'SYSTEM_' and b.grantee_id = a.user_id
and b.grantor_id = c.user_id and b.priv_id = d.priv_id
and b.obj_id = e.table_id and e.user_id = f.user_id
--order by grantor, owner, object_type, object_name, priv_name ;
```

-- 非オブジェクト権限

```
iSQL> create or replace view adm_sys_privs
as
select a.user_name grantee, c.user_name grantor,
replace(d.priv_name, '_', '') priv_name
from system_sys_users_ a, system_sys_grant_system_ b,
system_sys_users_ c, system_sys_privileges_ d
where c.user_name <> 'SYSTEM_' and b.grantee_id = a.user_id
and b.grantor_id = c.user_id and b.priv_id = d.priv_id ;
```

重要列)

GRANTEE	権限を与えられたユーザー
OWNER	対象OBJECTの所有ユーザー
OBJECT_NAME	OBJECT名
OBJECT_TYPE	OBJECT TYPE
GRANTOR	権限を与えたユーザー
PRIV_NAME	権限名
GRANTABLE	与えられた権限を再び与えることができるかどうか

● ADM_TABLESPACES : テーブルスペース詳細情報

実行例)

```
iSQL> select name, max_size, total, alloc_size, autoextend from
adm_tablespaces
where name like 'TBS%';
NAME MAX_SIZE TOTAL ALLOC_SIZE AUTOEXTEND
-----
TBS_ALTIMIS_DATA 4000 560 454 Y
TBS_TEST_IDX 16 16 4 N
```

実行例 - ディスク テーブルスペース使用量)

```
-- DISK Tablespaces Usage
select
a.name, a.max_size, a.alloc_size,
round(sum(b.extent_total_count * a.extent_page_count * a.page_size) /
(1024*1024), 2) used_size
from adm_tablespaces a, v$segment b
where a.tbs_id = b.space_id
group by a.name, a.max_size, a.alloc_size;
NAME MAX_SIZE ALLOC_SIZE USED_SIZE
-----
SYS_TBS_DISK_DATA 2058 376 175
TBS_ALTIMIS_DATA 4000 454 428.25
```

実行例 - メモリテーブルスペース使用量)

```
-- MEM Tablespaces Usage
select
a.name, a.max_size, a.alloc_size,
round(sum(fixed_alloc_mem + var_alloc_mem) / (1024*1024), 2) used_size
from adm_tablespaces a, v$memtbl_info b
where a.tbs_id = b.tablespace_id
group by a.name, a.max_size, a.alloc_size;
NAME MAX_SIZE ALLOC_SIZE USED_SIZE
-----
SYS_TBS_MEM_DATA 4096 1288 1247.56
```

実行例 - UNDOテーブルスペース使用量)

```
-- UNDO Tablespaces Usage
select
a.name, a.max_size, a.alloc_size,
(select round(sum(USED_PAGE_COUNT) * 8/1024,2) from v$undo_tbs)
used_size
from adm_tablespaces a
where
name in ('SYS_TBS_DISK_UNDO')
group by a.name, a.max_size, a.alloc_size;
NAME MAX_SIZE ALLOC_SIZE USED_SIZE
-----
SYS_TBS_DISK_UNDO 2048 1 0.05
```

VIEW)

```
iSQL> create or replace view adm_tablespaces
as
select b.name , a.tbs_id , a.max_size
, round((b.total_page_count * b.page_size) / (1024*1024),0) total_size
, round((b.allocated_page_count * b.page_size) / (1024*1024),0) alloc_size
, a.autoextend , b.extent_page_count , b.page_size
from ( select 1 tbs_id, ( select round(value1 / (1024*1024),0)
from v$property where name = 'MEM_MAX_DB_SIZE' ) max_size,
'Y' as 'AUTOEXTEND' from dual
union all
select spaceid tbs_id,
sum(decode(autoextend,0,currsz,maxsize)) * 8 / 1024 max_size,
decode(max(autoextend),1,'Y','N') as 'AUTOEXTEND'
group by spaceid ) a,
v$tablespaces b
where a.tbs_id = b.id
```

重要列)

NAME	テーブルスペース名
TBS_ID	テーブルスペースID
MAX_SIZE	最大設定サイズ(MB)
TOTAL_SIZE	現在全体サイズ(MB)
ALLOC_SIZE	SEGMENTに割り当てられた全体サイズ (MB)
AUTOEXTEND	自動拡張するかどうか
EXTENT_PAGE_COUNT	EXTENTされるpage数
PAGE_SIZE	Pageサイズ(Byte)

- ADM_REPLICATION : 二重化送受信関連詳細情報

実行例)

```
iSQL> select * from adm_replication;
REP_NAME PEER_IP REP_GAP XSN SENDER_STAT RECEIVER_STAT
-----
REP_MEM 10.0.0.2 0 14494722 ON ON
REP_MEM 192.168.4.112 (*) 0 14494722 ON ON
REP_DISK 10.0.0.2 n/a 14494722 OFF ON
```

VIEW)

```
iSQL> create or replace view adm_replication
as
select a.replication_name rep_name,
       d.host_ip || decode(d.host_ip, b.peer_ip, ' (*)', NULL) peer_ip,
       nvl(to_char(e.rep_gap), 'n/a') as rep_gap,
       a.xsn, decode(b.peer_port, NULL, 'OFF', 'ON') as sender_stat,
       decode(c.peer_port, NULL, 'OFF', 'ON') as receiver_stat
from system_sys_repl_hosts_d , system_sys_replications_a
left outer join v$repsender b on a.replication_name = b.rep_name
left outer join v$repreceiver c on a.replication_name = c.rep_name
left outer join (
  select rep_name, max(rep_gap) rep_gap from v$repgap
  group by rep_name
) e on a.replication_name = e.rep_name
where a.replication_name = d.replication_name
order by rep_name ;
```

重要列)

REP_NAME	二重化名
PEER_IP	二重化対象HOST IP, “*”は現在ACTIVEなIP
REP_GAP	未伝送ログ量(Byte)
XSN	二重化送信スレッドが送信したログSN
SENDER_STAT	送信スレッド動作有無
RECEIVER_STAT	受信スレッド動作有無

- ADM_LOCK : LOCK 関連照会

実行例)

```
iSQL> select tx_id, blocked_by, status, update_time, logfileno, query from
adm_lock;
TX_ID BLOCKED_BY STATUS UPDATE_TIME LOGFILENO QUERY
-----
16864 BEGIN 09/03 17:20:44 319 -- TX_REPLICATION --
186624 16864 BLOCKED 09/03 17:21:16 319 insert into TB_TEST1 (C1, C10)
values (1, SYSDATE)
```

VIEW)

```
iSQL> create or replace view adm_lock
as
select a.trans_id tx_id, d.wait_for_trans_id blocked_by,
       decode(b.status, 0, 'BEGIN', 1, 'PRECOMMIT',
              2, 'COMMIT_IN_MEMORY', 3, 'COMMIT', 4, 'ABORT',
              5, 'BLOCKED', 6, 'END' ) status,
       to_char(to_date('1970010109','YYYYMMDDHH'))
       + b.first_update_time / (1*60*60*24), 'MM/DD HH:MI:SS') update_time,
       b.first_undo_next_lsn_fileno logfileno, a.table_oid, a.lock_desc,
       decode(b.log_type, 1, '-- TX_REPLICATION --', c.query) query
from v$lock a
left outer join v$lock_statement c on c.tx_id = a.trans_id
left outer join v$lock_wait d on d.trans_id = a.trans_id,
v$transaction b
where a.trans_id = b.id ;
```

重要列)

TX_ID	トランザクションID、(二重化によるTX包含)
BLOCKED_BY	該当TXが待機しているTX
UPDATE_TIME	COMMITできない変更TX開始時間
TABLE_OID	テーブル名は、system_sys_tables_で TABLE_OIDと言う条件で検索
LOCK_DESC	LOCK MODE Ex. IS_LOCK, IX_LOCK, X_LOCK
LOG_TYPE	値が1であるなら二重化スレッドによるTXで V\$LOCK、V\$TRANSACTIONには存在するが、 V\$STATEMENTには除外になるTXのことを意味

付録 A
Trace Log
> Trace Log

付録 A >> Trace Log

アルティベースサーバーで実行されるSQL文の情報、エラーメッセージ種類またはSQL文の実行 時間などをaltibase_boot.logファイルに記録できるようにするプロパティ(Property)で、以下のような内容をaltibase_boot.logファイルに記録することができます。基本値は0で、任意の trace logを使用するため1を設定します。プロパティ(Property)ファイルに設定された値はALTER SYSTEM文を用い、変更できます。

TRCLOG	説明
TRCLOG_SET_CONFLICT_LOG	altibase_rp.logにreceiver側で発生したconflict message 記録
TRCLOG_SET_INSERT_SM_LOG	altibase_rp.logにreceiver側でinsertXLog時、SMで発生するerror message記録
TRCLOG_DML_SENTENCE	altibase_boot.logに実行したDMLとwhere節のpredicate分類、状態記録
TRCLOG_DETAIL_PREDICATE	explain plan 機能使用時にwhere節のpredicate分類、状態も共にdisplay
TRCLOG_SEQ_ITERATOR	sequential fetch時にsequential iteratorのtrace 情報記録
TRCLOG_SET_LOCK_TIME	Lock 設定時間記録
TRCLOG_SET_HBT_LOG	現在HeartBeat Threadの動作有無無(データ通信障害感知)が判断できます。また、HeartBeat Threadに登録されている全てのホストに対するリストを周期的にaltibase_rp.logに記録
TRCLOG_PSM_ERROR_LINE	PSM関連SQL文(create、procedure、execute Procedure など)実行時にエラーメッセージをaltibase_boot.logに記録

付録 B
データ型
比較
> データ型比較

付録 B >> データ型比較

DRDBMS データ型	説明	アルティベース データ型	説明
CHAR	固定長文字型 最大2000 byte	CHAR	固定長文字 最大32K
VARCHAR2	可変長文字型 最大4000 byte	VARCHAR	可変長文字型 最大32K
NCHAR	UNICODE固定長 文字型 最大2000 byte	-	CHAR使用
NVARCHAR2	UNICODE固定長 文字型 最大2000 byte	-	VARCHAR使用
CLOB	単一byte文字型 最大4G	CLOB	部分サポート最 大4G
NCLOB	UNICODE文字型 最大4G	-	VARCHAR使用
LONG	文字型最大2G	-	VARCHAR使用
NUMBER (p, s)	数字型 精密度p 1~38, スケールs -87~127	NUMBER	数字型、精密度p 1~38, スケールs -84~126
NUMERIC	-	NUMERIC	Numberと同一
INT	-	INTEGER	4 byte 整数型
DECIMAL	-	BIGINT	8 byte 整数型
DOUBLE PRECESION	-	DECIMAL	Numberと同一
FLOAT	-	DOUBLE	8 byte実数型
REAL	-	FLOAT	Numberと同一
	-	REAL	4 byte実数型
SMALLINT	-	SMALLINT	2 byte 整数型
DATE	日付型 BC 4712年1月1日 ~ AD 9999年12月 31日	DATE	日付型
TIMESTAMP	Microsecondを 包含したdate	DATE	Microsecondを包 含

BLOB	大容量バイナリ オブジェクト型 最大 4G	BLOB	大容量バイナリ オブジェクト型 最大4G
BFILE	大容量バイナリ ファイル型 最大4G	-	
RAW (size)	元始バイナリ型 最大2000 byte	-	BLOB使用
LONG RAW	可変長さ元始 バイナリ型 最大 2G	-	BLOB使用
-	-	GEOMETRY	空間型