Oracle9i

Oracle実践研修2

領域見積

2006.10.17

自己紹介

コミュニケーションテクノロジーズ株式会社
システム開発部
畠山基裕
連絡先：hata@comtec.co.jp

[Oracleの経験]

- Oracle Applications(現E-business Suite)を約4年
  - 販売系2つ、生産系3つ
  - セットアップ、機能評価、Extension設計/テスト、チューニング、運用支援
- その他、システム開発

この講座について

- プロ講師ではない
- 独自テキスト
  - 1から10まですべて書いているわけではない
  - 実務経験＋マニュアル＋ネット情報
- マニュアル、ネットから情報を得て自分で解決することが重要
  - OTNの講座やフォーラム、その他

カリキュラムの確認

- DB全体の設計　　1時間
- 表領域の設定　　1時間
- テーブルの設計　　1時間
- インデックスの設計1時間
- 領域監視　　1時間
- 断片化　　1時間

突然ですが8の復習

- テーブル生成のパラメータ

	initial	next	pctincrease
TableA	2BLK	4BLK	50%
TableB	2BLK	2BLK	50%

ディクショナリ管理

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

[1],[2],…:ブロック　[1+2],[5+6+7+8],[11+..+16]:エクステント

[1+2+5+6+7+8+11+12+..+16]:セグメント(Table/Index)

アンケート

- 8の復習内容について
  - 知らなかった
  - 聞いた記憶はあるが、よくは理解してない
  - 知っている
- INITRANS、MAXTRANS、PCTFREE、PCTUSEDについて
  - 知らない
  - 聞いた記憶はあるが、よくは理解してない
  - 知っている
- 使用しているDB=7/8/8i/9i/10g?

ブロックサイズの決め方1

	ブロックサイズ小	ブロックサイズ大
ブロックに納められるレコード数	少ない	多い
ブロックI/Oのコスト	低めになる	高めになる
フルスキャンのI/Oのコスト	DB_FILE_MULTIBLOCK_READ_COUNT(*)に依存、同じ値ならブロックサイズ大の方がコストが低い	
キャッシュヒット率	高めになる	低めになる
トランザクション競合発生の可能性	小さい	大きい
行連鎖発生の可能性	大きい	小さい
セグメント圧縮の効果	小さい	大きい
適しているシステム形態	OLTP系	DSS/DWH系

(*)1回のI/Oで読み取られるブロックの最大数

ブロックサイズの決め方２

- 初期化パラメータDB_BLOCK_SIZE に指定
- 2,048/4,096/8,192/16,384/32,768バイトのいずれかから選択。32,768バイトは64bit系の一部のOSのみ。
- 通常、４Kか８K。
- DB作成後は変更不可→表領域で設定可

ブロックサイズの決め方3

- 同一ブロックに対し複数セッションから同時更新TX発生
- キャッシュヒット率を高めたい

→小さめにする

- LOB型やLONG/LONG RAW型を多く利用
- レコード長の長いテーブルが多い
- テーブルやインデックスの圧縮機能を利用したい

→大きめにする

☆中庸的なケースの場合は8,192バイトでよい

ブロックサイズの決め方4

マルチブロック

Oracle9*i*では表領域ごとにブロックサイズを選択することができる。

表領域ごとにブロックサイズを指定するには

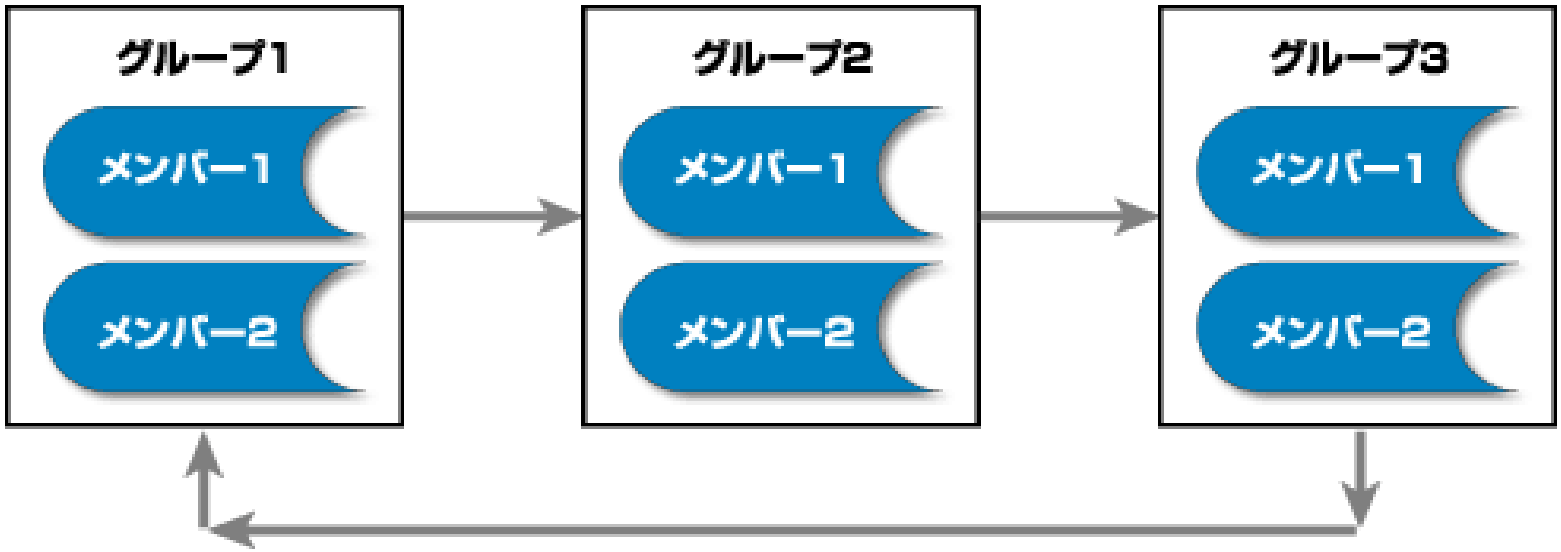
SQL> create tablespace tbs1 datafile … *blocksize 16k*;

注)DB_nK_CACHE_SIZEの設定が必要(デフォルトはゼロ)

制御ファイル作成における考慮事項

- 3重化を推奨→別々のディスク装置に配置（1つ壊れればDB停止となるが、複数あればリカバリーが楽）
- 通常、数M程度なので、容量については心配ない
- ちょっと見てみる(.ctl、EM)
  - c:\oracle\oradata\smpl\xxxxxx.ctl
  - c:\oracle\admin\smpl\init.ora

REDOログファイルの作成における考慮事項1



- ある時点の書き込み対象はある1つのグループ
- 同じグループのメンバーには同じ内容が書かれる

REDOログファイルの作成における考慮事項2

- 2つ以上のグループが必要
- 同じグループのメンバーには同じ内容が書かれる
- 同じグループのメンバーは別々のディスクに配置
- TX処理中のログスイッチ(グループ切替)はそのTXのパフォーマンスを落とす

REDOログファイルの作成における考慮事項3

- 容量見積

  書かれるデータ量は、対象のSQLや更新量などによって大きく変わるため、事前に正確に見積もることはまず不可能。

  小規模システムであれば1～10MB
  中規模システムであれば10～100MB
  大規模システムであれば100MB以上
  の 大きさでまず作成し、調整する

OMF構成(Oracle Managed Files)

- Oracle9i新機能
- 指定ディレクトリにまとめて表領域や制御ファイル、REDOログファイルを配置することで、DB管理の手間を大幅に省くことができる
  (DB_CREATE_FILE_DEST)
- 例）SQL> create tablespace tbs1;
- ディスクが1つしかないような小規模システム 向き
- 制御ファイル、REDOログファイルは多重化も可能
  (DB_CREATE_ONLINE_LOG_DEST_n；nは最大5)

表領域分割・配置の観点1

- 1)管理性
  - 断片化、管理工数削減、わかりやすい
- 2)耐障害性
- 3)パフォーマンス

表領域分割・配置の観点2

1)管理性の観点

- システム表領域、UNDO表領域、一時表領域は独立して作成
- OMF（Oracle Managed Files）を利用する
- 読取専用のセグメントを別表領域に分ける
- 格納オブジェクトで分ける（テーブルとインデックスなど）
- ユーザー（スキーマ）で分ける
- 業務用途で分ける

表領域分割･配置の観点3

2)耐障害性の観点

- ディスクの分割
- 同一ディスク内でもある程度ファイルを分けて配置する
- データの重要度に応じて耐障害性の高いディスク装置を利用する

表領域分割・配置の観点4

3)パフォーマンスの観点～以下をうまく分散

- テーブルと、そのテーブルに対するインデックス
- 結合対象のテーブル同士
- パラレル処理の対象になるテーブルやインデックス
- システム表領域
- UNDO表領域
- 一時表領域

ローカル管理表領域1

- Oracle8iから採用された、新しいエクステント管理の方法を採用した表領域
- データファイルのヘッダ部分に64KBの領域を取り、この領域でビットマップを利用してエクステントを管理(それまではシステム表領域)
- Oracle9iR1からローカル管理表領域がデフォルトの表領域に。Oracle9iR2からはシステム表領域をローカル管理にすることができるように。システム表領域のデフォルトはディクショナリ。

システム表領域をローカル管理にした場合の制限

補足

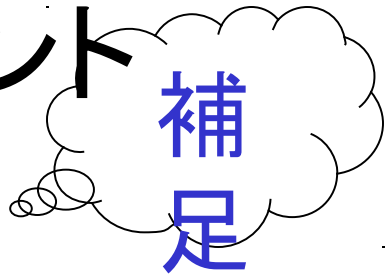
- データベースには、ディクショナリ管理表領域を作成できない。
- ローカル管理表領域はディクショナリ管理表領域に移行できない。
- 既存のディクショナリ管理表領域は、READ ONLY モードの場合にのみデータベースに残すことができる。READ WRITE モードには変更できまない。

ローカル管理表領域2

メリット

- パフォーマンスの向上
  - エクステント管理をするのにディクショナリ管理ではsystem表領域にI/Oが集中
  - エクステントが増えてもパフォーマンスが悪化しない(各データファイルでビットマップ管理だから)
- セグメントの設計・管理の簡素化
- 断片化の削減
- 表領域のコアレスが不要
  (ALTER TABLESPACE tbsp COALESCE;)

ローカル管理のエクステント



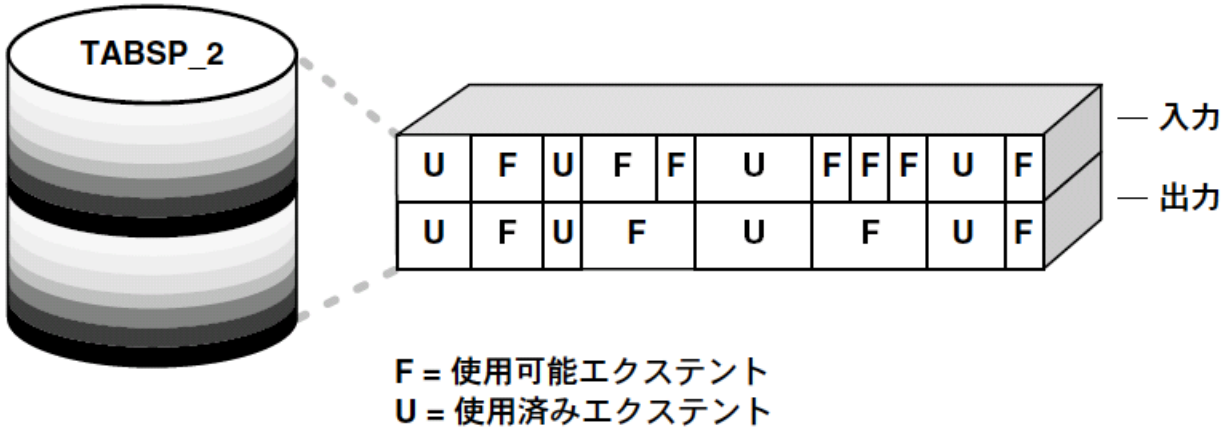
- **AUTOALLOCATE**
  - **64KB、1MB、8MB、64MBをOracleが自動確保**
- **UNIFORM**
  - **表領域作成時に指定した固定サイズで確保**

コアレス

補足

- ディクショナリ管理
  - 連続した空きエクステントを結合しなければ連続していると認識できない→コアレスを実施
- ローカル管理
  - 連続した空きエキステントを結合しなくても認識可能

図 11-1 空き領域の結合



ローカル管理表領域3

デメリット

- ダイレクト処理のパフォーマンスが落ちる
  - 従来型よりは大幅に高速
- きめ細かいエクステント管理ができない
  - 細かいサイズ指定ができないので未使用領域が増える

ローカル管理表領域4

データ表領域の見積り

- データファイルに格納するテーブルやインデックスの見積合計サイズに余裕値を加えた値を算出する
- AUTOALLOCATE指定の場合は64KBの倍数に、UNIFORM指定の場合はUNIFORM SIZEの倍数になるように切り上げ
- 表領域のヘッダとエクステント管理用のビットマップの領域として64KBを加算

ローカル管理表領域5

自動拡張（AUTOEXTEND)

- 表領域を作るときはファイルサイズを指定
- 自動拡張なら自動でファイルサイズを拡張するので便利
- しかし、ファイル単位での断片化が生じる

✂→保険として設定してもよいが、きちんと管理すべき

システム表領域

表領域の見積り

- 200M～400M程度
- ストアドプロシジャ、レプリケーションの利用などが多い場合は、多めに取る
- 必要以上に大きくならないのでAUTOEXTENDにしてもよいかも。

UNDO表領域1

- (8i)RBSを配置する表領域のこと
- 9iでは自動管理可能、作成も自動
  - 管理が容易
  - ORA-01555が抑えられる
  - 領域不足エラーが抑えられる(出にくい)
  - フラッシュバッククエリーが利用できる
- 手動管理も選択可
  - パフォーマンスチューニング

UNDO表領域2

表領域の見積り

(A × B ＋ 64 ÷ C ＋ D × 2) × C + 余裕値(単位:キロバイト)

A:UNDO_RETENTIONの値(フラッシュバック用データ保存期間;デフォルト900秒;0の場合はA=1に)

B:V$UNDOSTATビューのUNDOBLKSの値を600で除算した値

一番更新が多い時間帯のレコードを元にすること

事前見積の場合は1秒あたりの発生UNDOブロック数を仮定

C:ブロックサイズ(2/4/8/16/32のいずれか)

D:V$UNDOSTATビューのMAXCONCURRENCYの値

事前見積の場合は最大同時トランザクション数を仮定

UNDO表領域2

表領域の見積り

- 初期UNDOセグメントの数について

  初期化パラメータSESSIONSを参照する。初期値は2～10の範囲に設定される。基本的にOracle任せで構わないが、起動直後から大量の同時トランザクションが発生することがわかっているような場合はSESSIONSの値を大きめに取ること。

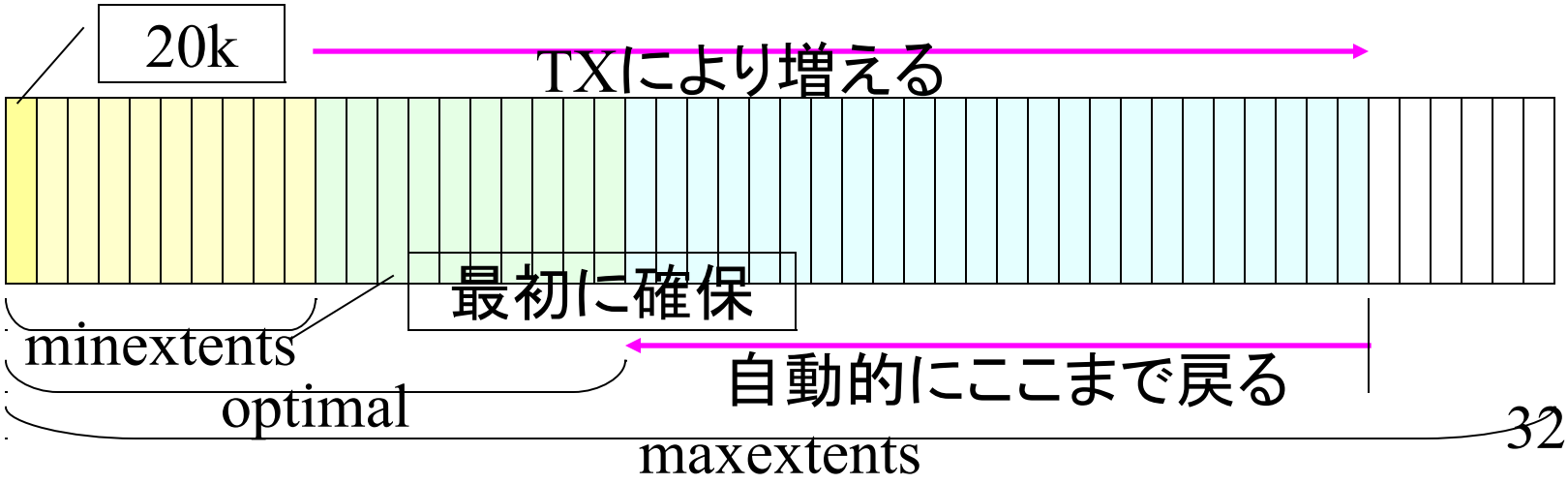
least(greatest(1.1 * SESSIONS / 5, 2), 10) (個)

注）leastはリスト中の一番小さな値を、greatestはリスト中の一番大きな値を返す関数です。

ロールバックセグメント(1)

補足

- **作成パラメータ例**
  - **INITIAL 20K　←1つめのextent size**
    NEXT 20K　←2つめ以降のextent size
    OPTIMAL 400K　←拡張後、このサイズに戻る
    MINEXTENTS 10　←最小エクステント数
    MAXEXTENTS 50　←最大エクステント数



ロールバックセグメント(2)

補足

SEGMENT_NAME	TABLESPACE_NAME	BYTES	BLOCKS	EXTENTS
SYSTEM	SYSTEM	409600	200	8
RB_TEMP	SYSTEM	1126400	550	11
RB1	RBS	614400	300	3
RB2	RBS	614400	300	3
RB3	RBS	614400	300	3
RB4	RBS	614400	300	3
RB5	RBS	614400	300	3
RB6	RBS	614400	300	3
RB7	RBS	614400	300	3
RB8	RBS	614400	300	3

一時表領域

- 主にメモリで処理できなかったソートの為の領域として使用
- 一番重いソート処理の対象になるデータ容量の2倍程度に余裕値を加えた容量を確保
- 作成時、テンポラリを指定
  - CREATE TEMPORARY TABLESPACE ~
- エクステントサイズはSORT_AREA_SIZEの倍数+ブロックサイズにする。OLTP系で数MB、DSS系で数十～数百MB。自動割当不可。

運用開始後の実際

- 足りなければ、再作成(システム表領域以外)
- テーブル、インデックス用の表領域はこまめにチェックする(Diskの容量不足に対処するため)
- AUTOEXTENDに頼るのはよくないが、保険にする分には問題ない(特に必要以上に大きくならないものについては)
- そもそも、事前に正しい情報がわかるのか??(テーブルごとの件数や増分、トランザクション数、ソートに使う容量、処理の多少、ロールバックの多少)

ハード導入のための見積(1)
~ここまでの内容を整理

- ブロックサイズ
  - 2、4、8、16、(32)KB～UNDO、データファイル算出で使用
- REDOログファイル
  - 1~10MB、10~100MB、100MB以上×メンバー数×グループ数
- システム表領域
  - 200MB～400MB
- UNDOファイル領域
  - 計算式により算出

以上は、勘と経験も大切：全社での経験値の積み重ね

表領域の設計

ハード導入のための見積(2)
~これからの内容を整理

- テーブル容量
  - 計算式により算出、これの合計がデータ表領域のサイズ。
- インデックス容量
  - 計算式により算出、これの合計がデータ表領域のサイズ

✓ 以上の合計が初期に必要となるディスク容量

✓ レコード数により数年間の増分をコントロール

✓ やはり、勘と経験も大切：全社での経験値の積み重ね

大まかな手順

- （手順1） 1レコードの平均長を求める
- （手順2） 1ブロックに収まるレコード数を求める
- （手順3） CEIL(想定レコード数÷手順2の値)×ブロックサイズがテーブル容量となる
  ※CEIL；一番近い整数に切り上げる

必要なブロック数

もっと簡単にいえば….

レコード長×レコード件数

だがブロックサイズに制限されるので上のように計算

レコード平均長1

- レコードヘッダのサイズ
  レコードヘッダのサイズは3バイト
- 列ヘッダのサイズ
  対応する列のデータ長が250バイト以下(NULL含む)の場合は1バイト、251バイト以上の場合は3バイト
- 列データのサイズ
  ・・・・次へ

図1:レコードの構造

テキストP102～103

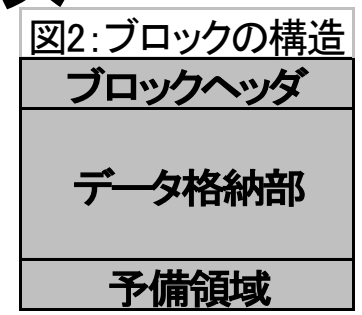
詳細は資料提供

レコード平均長2

表1:データ型による実消費サイズ

データ型	固定/可変	データが格納された時の長さ
CHAR	固定/	バイト数指定時はテーブル定義の長さの固定長
VARCHAR2	可変	実際に格納されているデータの長さ
NCHAR	固定	テーブル定義文字数の2倍(AL16UTF16指定時)
NVARCHAR2	可変	格納文字数の2倍(AL16UTF16指定時)
NUMBER	可変	長さ = 1 + CEIL (n / 2) nは格納された数値の整数部・小数部を合わせた総桁数。n>38の場合は38 有効桁数38桁未満の負数の場合は更に1バイト加算
DATE	固定	7バイト
TIMESTAMP	可変	秒の小数部にデータがある場合:11バイト固定 秒の小数部にデータがない場合:7バイト固定
TIMESTAMP WITH TIME ZONE	固定	13バイト
TIMESTAMP WITH LOCAL TIME ZONE	固定	11バイト
INTERVAL YEAR TO MONTH	固定	5バイト
INTERVAL DAY TO SECOND	固定	11バイト
RAW	可変	実際に格納されているデータの長さ
LONG	可変	実際に格納されているデータの長さ
LONG RAW	可変	実際に格納されているデータの長さ
BLOB/CLOB/NCLOB	指定による	DISABLE IN ROW指定時:20バイト ENABLE IN ROW指定で行内に格納時:実データ長+36バイト ENABLE IN ROW指定で行外(LOBテーブル)に格納時:実データ長により36～86バイト。見積時は余裕を見て86バイト固定で計算してください LOBテーブルの見積については後述 EMPTYの場合はデータ長の部分を0として計算
BFILE	固定	530バイト
ROWID	固定	6バイト(～Oracle7) 10バイト(Oracle8～)

ブロックに収まるレコード数



- ブロックヘッダのサイズ
  ヘッダのサイズ = 90 ＋ (INITRANS － 1)
- 予備領域のサイズ
  予備領域のサイズ = CEIL((表領域のブロックサイズ － ヘッダのサイズ) × PCTFREE)
- データ格納部のサイズ
  データ格納部のサイズ = 表領域のブロックサイズーヘッダのサイズー予備領域のサイズ
- TRUNC(データ格納部のサイズ ÷ 平均レコード長)

※INITRANSはテーブルパラメータ:同時TX数の初期値

※PCTFREEはそれ以上insertさせない割合:updateの増分を格納する

その他(1)

- **平均レコード長がデータ格納部より長い場合の見積方法（行連鎖）**
  - CEIL(平均レコード長 ÷ データ格納部の長さ) × 想定レコード数 × ブロックサイズ
  - PCTFREEは実際の値に関わらず0として計算
  - レコードサイズのばらつきが大きいと、かなり大きめの値になるので注意

その他(2)

- **LOBテーブルの見積方法**
  1. CHUNKの値をブロックサイズの倍数に切り上げる=実際のI/Oサイズを求める
  2. 平均LOB長を手順1の倍数に切り上げる
  3. LOB格納領域のサイズを求める
  4. RETENTION(PCTVERSION)領域を求める

http://otndnld.oracle.co.jp/skillup/oracle9i/3_1/index.html
を参照のこと

※CHUNK：LOBのI/Oサイズ

実習

- 別紙資料で計算
- OTN提供のExcelシートで自動計算
  - http://otn.oracle.co.jp/idba/availability/techlisting.html#ds
    の「領域サイズ見積計算」
  - ユーザ別/データベース管理者/可用性/9i8i可用性アーカイブ/データストレージ
- OTNのオンライン計算
  - http://otn.oracle.co.jp/document/estimate/index.html
- 確認

レコード長を短くするテクニック

- 可変長データ型の利用
- SJISの利用（日本語データの格納効率）
- NULLが格納されやすい列をまとめて後方に定義する→NULLの場合、列ヘッダが省略される（値を持つ2つの列の間にNULL値項目があると1バイト使用する）

テーブル作成パラメータ

パラメータ	ディクショナリ管理	ローカル管理
INITIAL	初期で確保するエクステントのサイズ。例えば100MBのINITIALを指定すると、1個の100MBのエクステントを確保します。	初期で確保するエクステントの総サイズ。例えば1MBのUNIFORM指定の表領域で100MBのINITIALを指定すると、100個の1MBのエクステントを確保します。
NEXT	二つ目のエクステントのサイズ。例えば70MBのNEXTを指定すると、二つ目のエクステント70MBの大きさで作成されます。	MINEXTENTSが2以上の時にINITIALとNEXTの合計のサイズになるよう一つ以上のエクステントが確保されます。例えば1MBの UNIFORM指定の表領域でMINEXTENTS 2、INITIAL 100MB、NEXT 70MBと指定すると、170個の1MBのエクステントを確保します。
PCTINCREASE	三つ目以降のエクステントを確保する際の、一つ前のエクステントサイズからの増分。デフォルトは50(%)。例えばNEXT 100MB、PCTINCREASE 40という指定の場合、三つ目のエクステントはNEXT値の40%増しで140MB、四つ目のエクステントは三つ目のエクステントの40%増しで 196MBとなります。	MINEXTENTSが3以上の時に、ディクショナリ管理と同じような計算方法で一つ以上のエクステントが確保されます。デフォルトは0 (%)。例えば1MBのUNIFORM指定の表領域でMINEXTENTS 4、INITIAL 100MB、NEXT 100MB、PCTINCREASE 40と指定すると、100 + 100 + 140 + 196 = 536個の1MBのエクステントを確保します。
MAXEXTENTS	指定した数より多くのエクステントが作成できなくなります。	意味はディクショナリ管理と同じですが、指定しても無視され、常にUNLIMITEDになります。

大まかな手順

- （手順1） 1レコードの平均長を求める
- （手順2） 1ブロックに収まるレコード数を求める
- （手順3） CEIL(想定レコード数 × 1.05 ÷ 手順2の値) × ブロックサイズ＝インデックス容量
- （手順4） 空のテーブルに対してインデックスを作成する場合は手順2の値×1.3＝インデックス容量

CEIL ;一番近い整数に切り上げる
1.3 ;経験値(1.1でも収まる場合がある)

レコード平均長1

- レコードヘッダのサイズ
  レコードヘッダのサイズは3バイト
- ROWIDのサイズ
  Oracle9iの本来のサイズは10バイトだが、索引レコード中のROWIDは6バイトのフォーマットで格納。ただし、グローバルのパーティション・インデックスの場合は10バイト。一意でない索引の場合は更に1バイト必要。

図1:B-Treeインデックスのレコードの構造

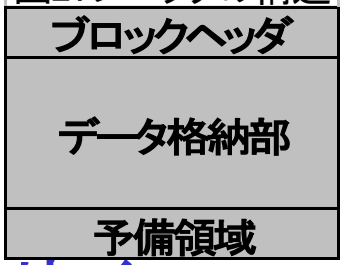
レコード平均長2

- オーバーヘッドサイズ
  (127バイト以下の列数 × 1) ＋ (128バイト以上の列数 × 2) ＋ 10 バイト
- 列ヘッダのサイズ
  対応する列のデータ長が250バイト以下の場合は1バイト、251バイト以上の場合は3バイト
- 列データのサイズ
  データ型による(テーブル設計と同じ)

※列ヘッダ+列データの合計が9バイトに満たない場合は9バイトに切り上げ

ブロックに収まるレコード数

図2:ブロックの構造



- ブロックヘッダのサイズ

ヘッダのサイズ = 96 ＋ (24 × INITRANS)

- 予備領域のサイズ

予備領域のサイズ = CEIL((表領域のブロックサイズ － ヘッダのサイズ) × PCTFREE)

- データ格納部のサイズ

データ格納部のサイズ = 表領域のブロックサイズ－ヘッダのサイズ－予備領域のサイズ

- TRUNC(データ格納部のサイズ ÷ 平均レコード長)

※索引のPCTFREEは、既存の索引エントリ間に新たな索引エントリが挿入されたときに備えて各ブロックに空けておく領域を指定するもの

その他

- LOBインデックスの見積方法
  - LOBテーブルと同じ表領域にOracleが自動確保
  - サイズはLOBテーブルの5%
- ビットマップ・インデックスの見積方法
  事前に見積もるのは困難。B-Treeよりは小さくなるようなので、B-Treeの方法で見積もっておけばオーバーはしない。

サンプルと実習

- 別紙資料で計算
- OTN提供のExcelシートで自動計算
  - http://otn.oracle.co.jp/idba/availability/techlisting.html#dsの「領域サイズ見積計算」
- OTNのオンライン計算
  - http://otn.oracle.co.jp/document/estimate/index.html
- 確認

Oracleの物理構造と断片化

階層	断片化現象
データファイル（表領域）	・ファイルレベルの断片化 ・データファイルレベルの未使用領域の発生
セグメント	・位置の高いハイウォーターマーク ・セグメントレベルの未使用領域の発生 ・階層の深いインデックス
エクステント	・不連続のエクステント
ブロック	・行移行 ・行連鎖 ・ブロック内の未使用領域の発生

ファイルレベルの断片化

- ディスクが1つのシステムで発生しやすい
- 自動拡張で断片化が発生する
- Windowsであればデフラグツールを利用して調査

ハイウォーターマーク

- 最高水位標
- 過去にデータが格納されたことのある一番高い（最後の）位置を示す指標
  - データファイル(表領域)
  - セグメント

データファイルレベルのHWM以降の未使用領域

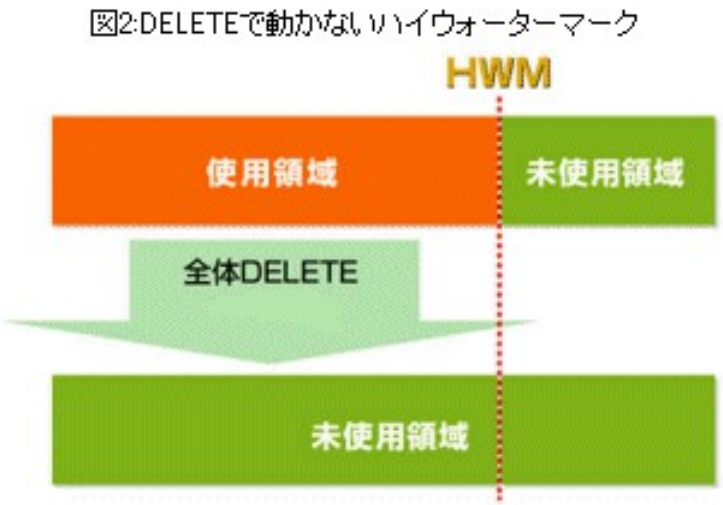
- データ検索、更新には影響しない
- 将来のデータ増加に対応するためのものは悪くはない
- バックアップ・リストアに時間がかかる
- SQL→http://otndnld.oracle.co.jp/skillup/oracle9i/5_1/index.html(＜表領域のデータファイルごとのHWM以降の未使用領域のサイズを求める＞、＜ある表領域の未使用領域の総サイズを求める＞)

データファイルレベルのHWM以前の未使用領域

- 未使用エクステント
- SQL→http://otndnld.oracle.co.jp/skillup/oracle9i/5_1/index.html(＜HWM以前のセグメント間の未使用領域を求める＞)

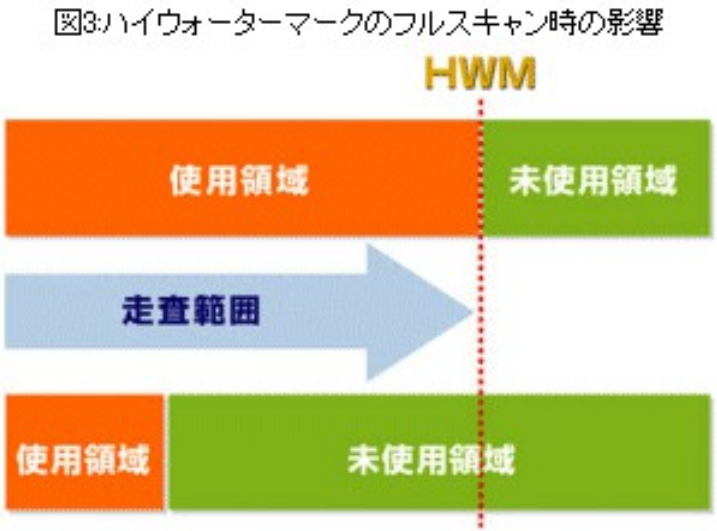
セグメントレベルの断片化

- HWMは自動的に小さくはならない
  - 全レコードをdeleteしても同じ位置
    - 全件検索に影響
    - ダイレクト・ロード、ダイレクト・ロード・インサートに影響



全件検索への影響

- HWMの位置までスキャンする



ダイレクト処理への影響

- HWMの位置以降に書き込む

HWMの位置を調べる

- DBMS_SPACEパッケージのUNUSED_SPACEプロシージャを利用
- SQL→http://otndnld.oracle.co.jp/skillup/oracle9i/5_1/index.html(＜HWMの位置を調べる＞)

領域監視

セグメントレベルのHWM以降の未使用領域

- データファイルと同じような影響
- DBMS_SPACEパッケージのUNUSED_SPACEプロシージャを利用
  - SQL→http://otndnld.oracle.co.jp/skillup/oracle9i/5_1/index.html(＜HWM以降の未使用領域を求める(1)＞)
- ANALYZE後ディクショナリ参照
  - SQL→http://otndnld.oracle.co.jp/skillup/oracle9i/5_1/index.html(＜HWM以降の未使用領域を求める(2)＞)

セグメントレベルのHWM以前の未使用領域

- 未使用エクステント
- SQL→http://otndnld.oracle.co.jp/skillup/oracle9i/5_1/index.html(＜HWM以前の未使用領域を求める＞)

エクステントレベルの断片化

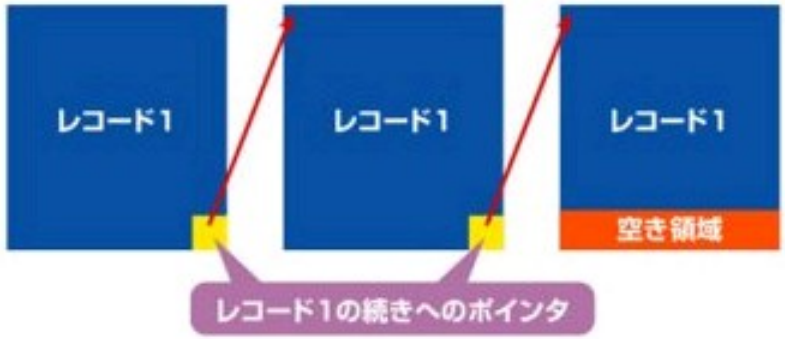
- ディクショナリ管理の場合、不連続の未使用エクステントが発生する
- SQL→http://otndnld.oracle.co.jp/skillup/oracle9i/5_1/index.html(＜エクステントが連続しているかどうかを調べる＞)
- ローカル管理では気にしなくて良い

ブロックレベルの断片化

- 行移行と行連鎖

図6:行移行

図7:行連鎖



行移行・行連鎖の調査

- ANALYZE後、ディクショナリ(user_tablesなど)のCHAIN_CNT列を見る
- 行移行と行連鎖の違いは知ることは出来ない
- しっかり見積ること

断片化解決の概要

表1：Oracleの物理構造と断片化、解決方法

階層	断片化現象	解決方法（再編成方法）
データファイル（表領域）	ファイルレベルの断片化	OSコマンドによるデフラグ
	データファイルレベルの未使用領域の発生	表領域レベルの再編成
		データファイルの縮小
セグメント	位置の高いハイウォーターマーク	セグメントレベルの再編成
	セグメントレベルの未使用領域の発生	エクステントの切り捨て
		セグメントレベルの再編成
	階層の深いインデックス	セグメントレベルの再編成
エクステント	不連続のエクステント	セグメントレベルの再編成
		表領域レベルの再編成
ブロック	行移行	セグメントレベルの再編成
		行移行しているレコードのみの再編成
	行連鎖	ブロックサイズ変更して再編成
	ブロック内の未使用領域の発生	セグメントレベルの再編成

データファイルレベル

- Windowsではデフラグツールを利用
- UNIX、Linuxではあまり気にしなくてよい

表領域レベルでの再編成

1. 表領域ごとExport
2. (パラメータを変更したい場合)オブジェクトの削除＆再作成
   - 表領域を削除した場合、Import前に再作成が必要(表領域のcreate文がexportファイルに入ってないため)
3. Import

※HWM以降を縮小することもできる

```
SQL> alter database datafile 'c:\temp\test.dbf'
resize 100m;
```

セグメントレベルの断片化解消

- テーブル、インデックスのExport/Import
- テーブルの表領域間の移動
  - SQL> alter table table1 move tablespace tbs1;
  - 移動と同時に再編成してくれる
  - tablespace句を省略すると元の表領域に再編成した形で生成してくれる
- データを削除してもよければ、truncate table ～
- インデックスの再編成
  - SQL> alter index index1 rebuild;

ブロックレベルの断片化解消

- 行移行の再編成
  - セグメント、表領域レベルの再編成で解消する
  - 一部のレコードだけが行移行しているなら、それだけ解消することも可能
    →http://otndnld.oracle.co.jp/skillup/oracle9i/6_1/index.html(**行移行の再編成**)
- 行連鎖の再編成
  - 表領域レベルの再編を行うが、その際、ブロックサイズを大きくする

最後に1

- ファイル配置→Excel資料
- OEMの表領域マップ
  - 表領域選択→ツール(T) →Tuning Pack→表領域マップ
  - ツール→表領域分析(再編成/セグメント再構成/エクステント結合)

最後に2

- 旧→新システムへの移行では、データ移行作業が発生
- 詳細見積をせず、全データをテスト的に移行してみる(HW購入用の見積は必要→経験値を集積しておく/DISKの値段から検討)
- DBA_SEGMENTS、DBA_EXTENTSのBYTESカラムで、容量を確認→配置、作成

最後に3

- OTNをよくチェックする(http://otn.oracle.co.jp/)
- 例えば「はじめてのOracle9iデータベース:第5章 バックアップ・リカバリを体験しよう」
- 今回の元資料「Oracle9i物理設計」http://otn.oracle.co.jp/skillup/oracle9i/index.html

終了